Week 1 Write-UP

- # Web
- ## Classic Childhood Game

  通过关于信息，检索到本题改自 https://github.com/Vinlic/h5-mota 项目。对比所有 JS 脚本发现只有 *Events.js* 在题目所给的版本中新增了少许代码。将该代码执行函数由打败魔王的 case 移动至游戏刚开始与仙子对话的 case，重启游戏，移动到仙子前直接触发 flag。

- ## Become A Member

  第一关卡了很久，一开始以为身份认证是加 *Authorization* 头或者 *Cookie*，试了好几次密码都过不去，甚至有想跑字典的冲动。后来想到 User-Agent 也有“身份”的意思，指定 *User-Agent*，过了。最后要求 json 登录，POST 过去 404 了，可能不存在 POST 方法？ 问了 Google 才知道 GET 方法没有强制要求不能带 body，用 GET 方法并指定 *Content-Type* 拿到 Flag 了。

- ## Guess Who I Am

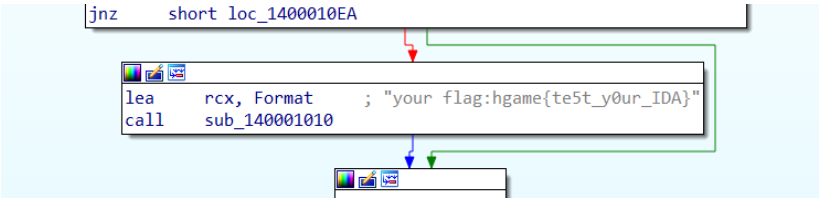
  从 Vidar 主页的 *index.33923334.chunk.js* 脚本中提取成员信息，并保存为 json 格式。编写的 Python 脚本存在家里的电脑上了，~~不知道明天下午八点前可不可以回到家打开电脑拿出来。~~事情好多，明天才能回去，今晚是拿不到了。这道题下发的题目与 Cookie 有关，答对了会有新的 Cookie 下发，然后用新的 Cookie 去请求下一题。是在考察爬虫中 Cookie 的保存和使用吗？

- ## Show Me Your Beauty

  前端是白名单过滤，用 Burp Suite 改请求绕过。后端是黑名单匹配，大小写可以绕过，命名为 *a.PhP* 即可。之后用蚁剑连上上传的 webshell 找 flag。

- # Reverse
- ## test your IDA

  IDA 安装完毕后，打开就是 flag。



- # Pwn
- ## Test NC

  nc 连上，直接 *cat /flag*

- # Crypto

- RSA

  使用 http://www.factordb.com/ 对 n 分解，得

  p =
  1123913498780499358676355902818724505765255021951520176864477073386908818532074093845017881613839484432972331143354989949979577565592126166408799709729481 3

  与

  q =
  120229126614209415925697517318026393750884274634301622521130826196178370109130025154502236569428363780411221638333590979109356384234640062528142669591289 53

  之后使用 Python 按常规解法即可得到 Flag。脚本如下：

```
1.  import libnum
2.  from Crypto.Util.number import long_to_bytes
3.  c=110674792674017748243232351185896019660434718342001686906527789876264976328686
    134101972125493938434992787002915562500475480693297360867681000092725583284616 35
    354342238848920811454500713860654367804079865183602743338328217708103415158993 50
    242920172072090568292501522191835184003648711095598256792735022749555 82
4.  n=135127138348299757374196447062640858416920350098320099993115949719051354213545
    596643216739555453946196078110834726375475981791223069451364024181952818056802 08
    956706492651029412459417447812321651660036833476384920694294282471153133423910 68
    074540863892111391530236622661259374816695207718793550899976711250207 89
5.  p = 1123913498780499358676355902818724505765255021951520176864477073386908818532
    074093845017881613839484432972331143354989949979577565592126166408799709729481 3

6.  q = 1202291266142094159256975173180263937508842746343016225211308261961783701091
    300251545022365694283637804112216383335909791093563842346400625281426695912895 3

7.  e = 65537
8.  d = libnum.invmod(e, (p - 1) * (q - 1))
9.  m = pow(c, d, n)
10. print(long_to_bytes(m))
```

- 神秘的电话（卡住了）

  对音频文件进行摩尔斯解码，得到 “*0223E#PRIIBLY##HONWA#JMGH#FGKCQAOQTMFR*”

  对 txt 进行 base64 解码，结果为‘*几个星期前，我们收到一个神秘的消息。但是这个消息被重重加密，我们不知道它的真正含义是什么。唯一知道的信息是关于密钥的：“只有倒着翻过十八层的篱笆才能抵达北欧神话的终点”。*’

  提取关键信息，“18”，“倒着”，“篱笆”

  对摩尔斯解码字符串进行栅栏密码解密，18 为密钥，转小写，flag 错误

  对摩尔斯解码字符串倒转后进行栅栏密码解密，18 为密钥，转小写，flag 错误

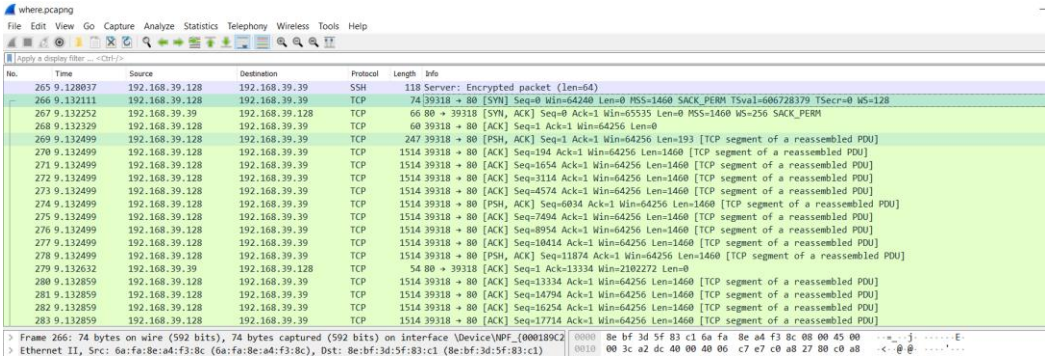
  对摩尔斯解码字符串进行栅栏密码加密，18 为密钥，转小写，flag 错误

对摩尔斯解码字符串倒转后进行栅栏密码加密，18 为密钥，转小写，flag 错误

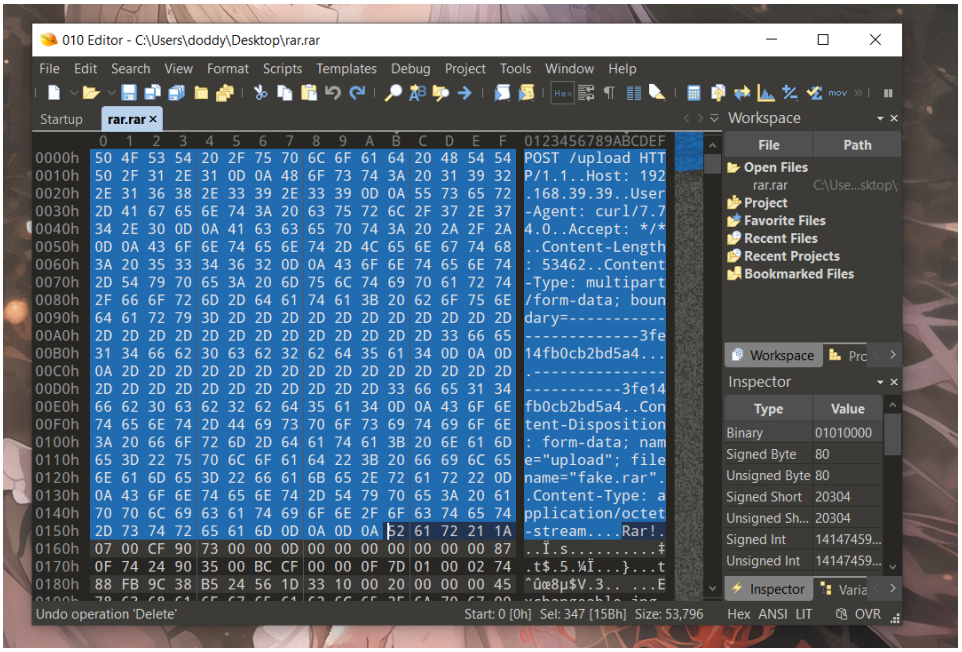
卡住了

- Misc
- Sign In
  签到题，直接 base64 解码得到 flag。

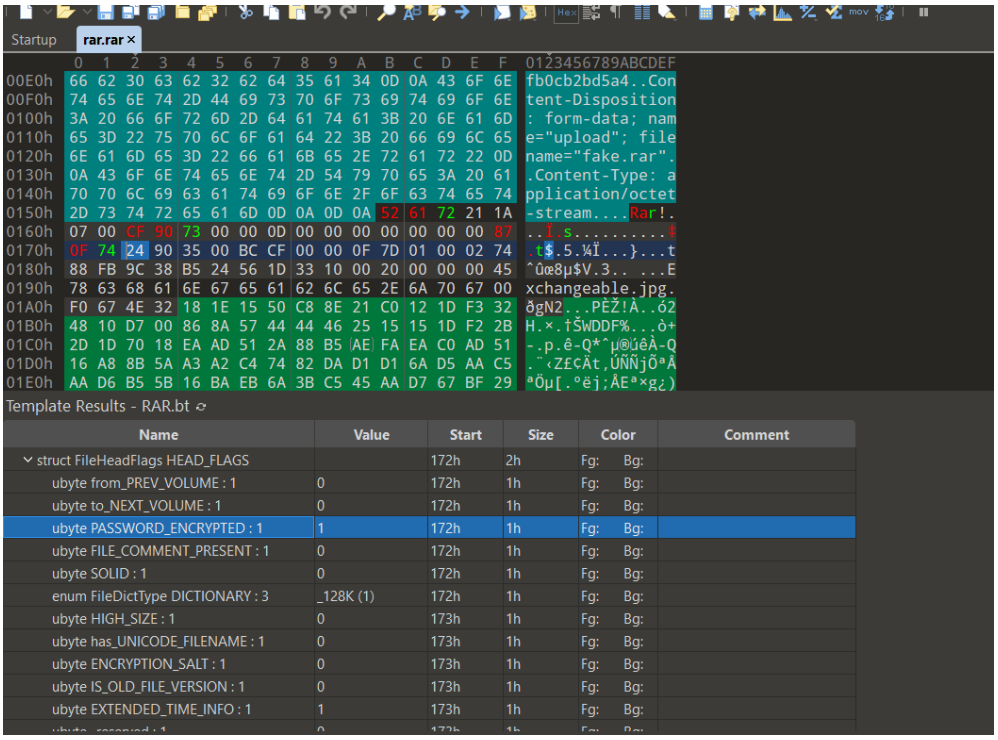
- Where am I
  Wireshark 打开附件，发现有明文的 HTTP 流。



提取整个流并以 Raw 格式保存，使用 010 Editor 去掉 HTTP 头和文件上传的 body 头。根据文件头的标志得知上传的是一个 Rar 文件。



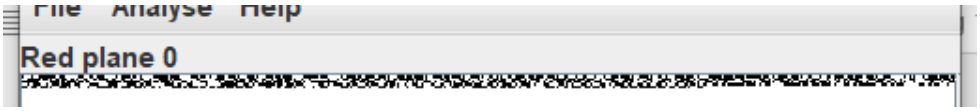
使用 010 editor 找到加密位并改为 0，以破除伪加密。



保存后解压得到一张纯黑照片。右键属性发现含有 Exif 信息，接下来就是水到渠成的了。
Google 上找一个 Exif 在线查看的网站即可得到经纬度信息。
提交 flag 的时候注意一下经纬度顺序的问题。

- 神秘的海报

  StegSolve 打开附件，发现红绿蓝的第 0 通道图像顶部有数据。



  提取，得到隐写：

  “Sure enough, you still remember what we talked about at that time! This is part of the secret: `hgame{U_Kn0w_LSB&W`
  I put the rest of the content here, https://drive.google.com/file/d/13kBos3Ixlfwkf3e0z0kJTEqBxm7RUk-G/view?usp=sharing, if you directly access the google drive cloud disk download in China, it will be very slow, you can try to use Scientific Internet access solves the problem of slow or inaccessible access to external network resources. This is my favorite music, there is another part of the secret in the music, I use Steghide to encrypt, the password is also the 6-digit password we agreed at the time, even if someone else finds out here, it should not be so easy to crack ((hope so ”

  打开不存在的网址下载，使用 *stegcracker* 和六位纯数字的字典进行爆破，得到密码 123456 和 flag 的另一部分。

- e99p1ant_want_girlfriend

图片的高度被修改过，网上找个好用的脚本跑一下，把正确的宽度和高度跑出来，010 editor 修改正确的宽度和高度再用图片查看器打开，flag 就在图片底下。

- # Blockchain
- ## Checkin（卡住了）

学了一点散装的智能合约和 Geth RPC 协议，大概是有点头绪了。但是不知道传参的时候错在哪了，总之没做出来。

糊了一个 python 脚本用于编码指令

```python
from web3 import Web3
from solcx import install_solc
from solcx import compile_source
install_solc(version='latest')
compiled_sol = compile_source(
    '''''
    pragma solidity 0.8.17;

    contract Checkin {
        string greeting;

        constructor(string memory _greeting)  {
            greeting = _greeting;
        }

        function greet() public view returns (string memory) {
            return greeting;
        }

        function setGreeting(string memory _greeting) public {
            greeting = _greeting;
        }

        function isSolved() public view returns (bool) {
            string memory expected = "HelloHGAME!";
            return keccak256(abi.encodePacked(expected)) == keccak256(abi.encodePacked(greeting));
        }
    }
    ''',
    output_values=['abi', 'bin']
)
contract_id, contract_interface = compiled_sol.popitem()
abi = contract_interface['abi']
address='0x088E182C5d5F4AD690dF3a4E9B5Bb0Be028C962a'
w3 = Web3(Web3.HTTPProvider("http://week-1.hgame.lwsec.cn:32593/"))
api = w3.eth.contract(address=address, abi=abi)
print(api.encodeABI(fn_name="setGreeting", args=["HelloHGAME!"]))
```

得到编码的 data：

"0xa41368620000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000b48656c6c6f4847414d4521000000000000000000000000000000000000000000000000"

nc 连接，按 1 生成自用地址。打开水龙头给地址转 1 ETH，nc 按 2 生成合约(?)地址。

使用 curl 向 RPC 调用 eth_call 指令，from 处填写第一步得到的地址，to 处填写合约地址：

root@ser030937873700:~# curl --data '{"jsonrpc":"2.0","method": "eth_call", "params": [{"from": "0xc1A91bb5AC3987CEf10A06A804FB2Ca9628e6361", "to": "0x7f52C9c5c8Be0F804eA99267eAFCFffBF4120869", "data": "0xa41368620000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000b48656c6c6f4847414d45210000000000000000000000000000000000000000000"}, "latest"], "id": 8}' -v -H "Content-Type: application/json" http://week-1.hgame.lwsec.cn:30032

返回 0x （函数无返回值）

调用 greet 函数，查看 greeting 变量，得到结果
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000568656c6c6f000000000000000000000000000000000000000000000000000000

ABI 解码后为：

[

"hello"

]

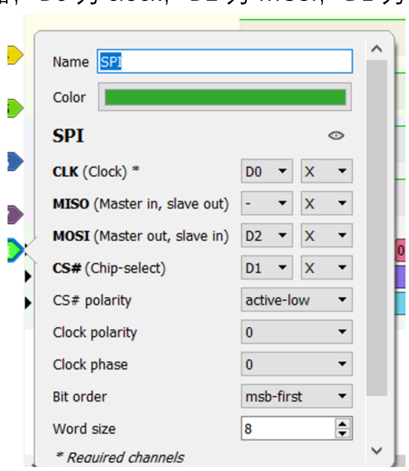
greeting 变量的值未达到期望，不知道是什么原因。没时间了，等官方 Write UP 看看。

尝试过将 *eth_call* 方法改为 *eth_sendTransaction*，报错 *jsonrpc method is not allowed*.

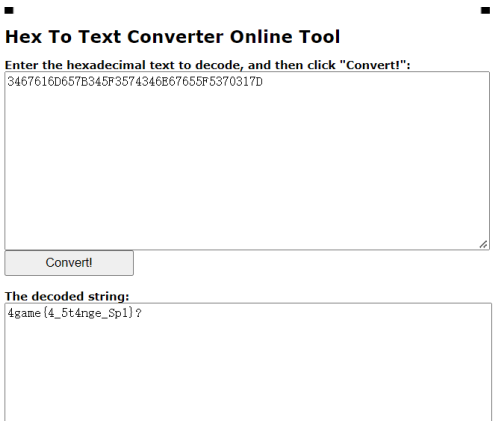
尝试使用 *eth_sendRawTransaction*，卡在 *eth_signTransaction* 报错 *jsonrpc method is not allowed.*

卡住了

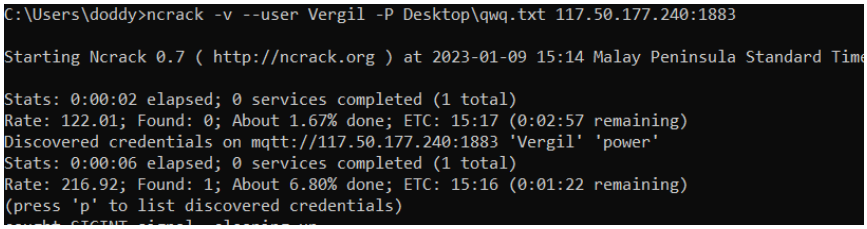
- IoT
- Help marvin
  拖入 PulseView 设置好 SPI 解码器，D0 为 clock，D2 为 MOSI，D1 为 chip-select, 得到一组 Hex 数据。



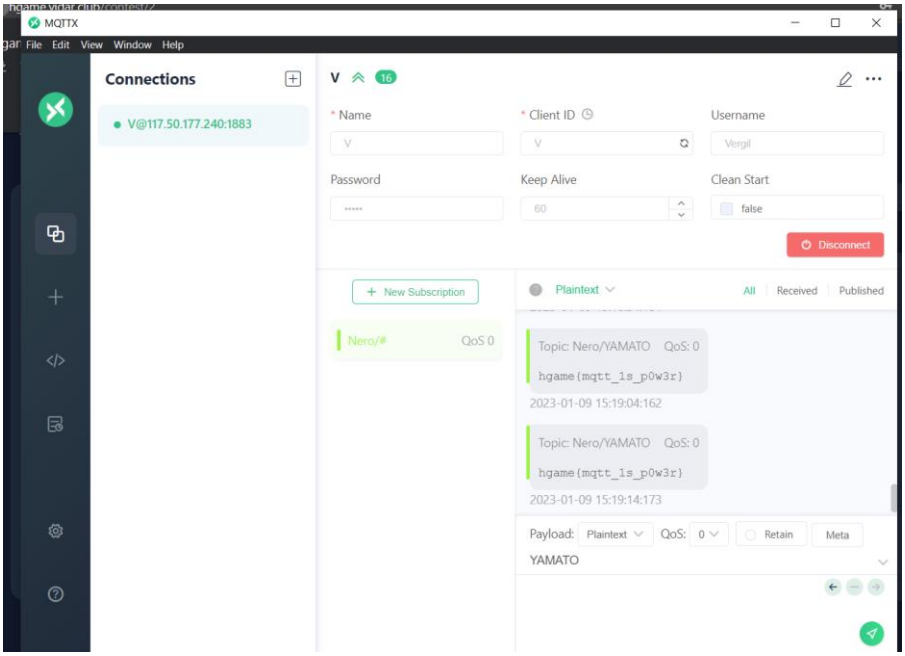
转为 Ascii，不知道为什么第一位字符是 4… 改为 h 后提交成功。



- Help the uncle who can't jump twice
  1883 是 mqtt 的端口号，经 nmap 扫描确实是跑着 mqtt 协议。使用提供的字典，以 *Vergil* 为用户名爆破，得到密码 *power*



接下来找一个好用的 mqtt 客户端，例如 MQTTX。登录后定阅 *Nero/#* , 得到 flag。


MQTTX
File Edit View Window Help
Connections
V@117.50.177.240:1883
V 16
* Name
* Client ID
Username
Password
Keep Alive
Clean Start
false
Disconnect
New Subscription
Nero/#
QoS 0
Plaintext
All
Received
Published
Topic: Nero/YAMATO QoS: 0
hgame{mqtt_1s_p0w3r}
2023-01-09 15:19:04:162
Topic: Nero/YAMATO QoS: 0
hgame{mqtt_1s_p0w3r}
2023-01-09 15:19:14:173
Payload:
Plaintext
QoS:
0
Retain
Meta
YAMATO