

# HW - Balance Program


201714089

융합보안학과 이재승

## 결과

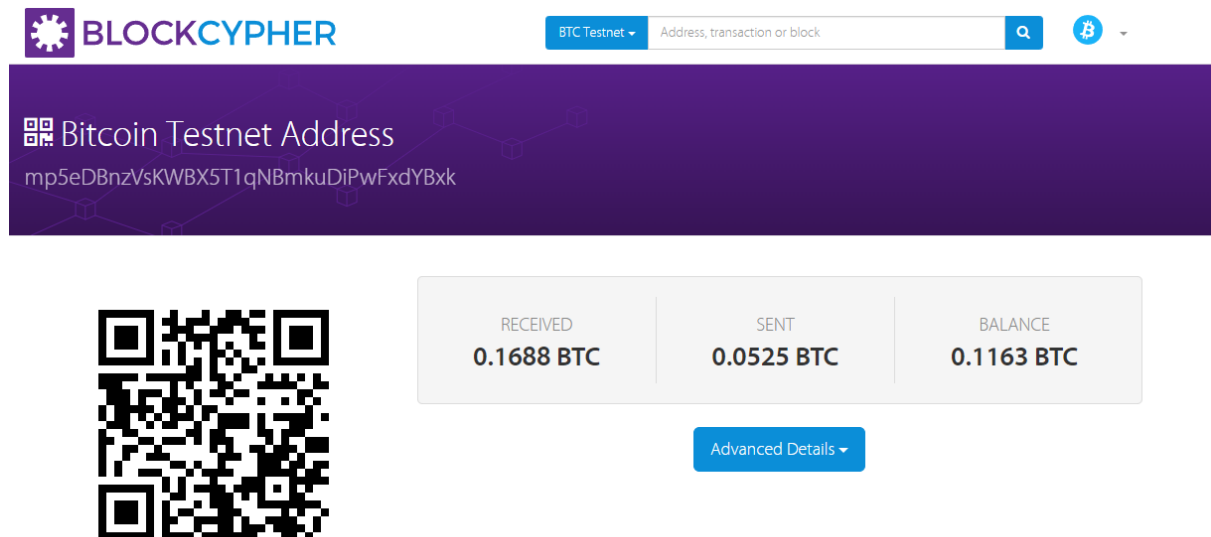
BTC Address mp5eDBnzVsKWBX5T1qNBmkuDiPwFxdYBxk | BlockCypher

BTC Address mp5eDBnzVsKWBX5T1qNBmkuDiPwFxdYBxk has had 5 transactions and has a balance of 0.1163 BTC (0.1688 BTC received and 0.0525 BTC sent).

 <https://live.blockcypher.com/btc-testnet/address/mp5eDBnzVsKWBX5T1qNBmkuDiPwFxdYBxk/>



위 사이트에서 testnet에서의 api 테스트가 가능합니다. 문제의 address를 조회하면 아래와 같습니다.



| RECEIVED   | SENT       | BALANCE    |
|------------|------------|------------|
| 0.1688 BTC | 0.0525 BTC | 0.1163 BTC |

Advanced Details ▾

testnet에서 alice의 address를 조회한 결과

과제로 작성한 코드를 실행해보면 아래와 같이 balance 값을 구해낼 수 있습니다.

```
(base) └─(kali㉿kali)-[~/workspace/kgu]
└─$ python btc_balance.py
[*] 2090000 / 2104345
[vout] 0ef83f003cfdb73e678e031a130ac9be83b710da189cae03cd94de571952583f[1] 0.10000000
[vout] 41660290939f71638ab06f42d83a52264cb929ea06dca15b586b97a3aa71fa90[1] 0.05250000
[*] 2095000 / 2104345
[vin] 41660290939f71638ab06f42d83a52264cb929ea06dca15b586b97a3aa71fa90[1] 0.05250000
[vout] 0e74a5c16f40f15001372f559d6dd55665a87aa41621791f723fe9db6674e99d[1] 0.00580000
[vout] 39d08f87b44b1679de8da61a90f27344de768a98f800c660955b40c8d1ee51d[0] 0.01050000
[*] 2100000 / 2104345
0.1163
```

코드 실행화면

## 동작 과정

1. 전체 블록을 탐색하면서 tx 마다 vin 및 vout 검사
2. UTXO 리스트에 있는 tx를 vin에서 발견 시 UTXO에서 제거
3. vout에 alice address에 해당하는 out이 있으면 UTXO에 추가
4. 모든 블록을 순회 후 UTXO 전체 합계

## Source Code

```
from bitcoin.rpc import RawProxy

# ...

if __name__ == "__main__":
    api = RawProxy(service_port=18332)
    alice_address = "mp5eDBnzVskWBX5T1qNBmkuDiPwFxdYBxk" # testnet

    balance = Balance(api, alice_address, start_block_height=2090000, debug=True)
    print(balance.calc())
```

**Balance** 클래스를 생성하여 개발했습니다. api를 사용할 수 있는 객체와 alice의 address를 넘겨주면 됩니다. 기본적으로 아무런 정보가 없다면 Genesis block 부터 탐색해야 하지만 연산량을 고려하여 2090000 이후부터 탐색하도록 했습니다. (testnet address 조회 웹사이트에서 정보를 획득했습니다)

**Balance** 클래스의 **calc()** 메소드를 호출하면 탐색이 시작됩니다.

```
class Balance:
    def __init__(self, api, address, start_block_height=0, debug=False):
        self.api = api
        self.debug = debug
        self.address = address

        self.utxo = {}

        self.total_height = api.getblockcount()
        self.start_block_height = start_block_height
        self.next_block_hash = api.getblockhash(start_block_height)

    def calc(self):
        # ...

    def check_vin(self, tx):
        # ...

    def check_vout(self, tx):
        # ...

    def is_lastblock(self, block):
        # ...
```

클래스 내 메소드들은 다음과 같습니다. **\_\_init\_\_()** 메소드에서는 초기화를 담당하고, **calc()** 메소드는 전체 블록을 탐색합니다. **check\_vin()** 메소드와 **check\_vout()** 메소드는 블록 탐색 과정에서 tx의 vin과 vout을 검사합니다. 마지막으로 **is\_lastblock()** 메소드는 현재 블록이 마지막인지 체크하는 메소드입니다. **calc()** 메소드의 반복을 제어합니다.

```

def calc(self):
    while True:
        block = api.getblock(self.next_block_hash, 2)

        for tx in block['tx']:
            self.check_vin(tx)
            self.check_vout(tx)

        if self.is_lastblock(block):
            break

    return sum([float(self.utxo[txid]['value']) for txid in self.utxo.keys()])

```

반복문을 돌면서 블록을 탐색합니다. 초기 `getblockhash()` api를 이용해서 height의 block hash 값을 가져오도록 설계했으나, 실행시간 단축을 위한 방안을 고려하던 중 `nextblockhash` 필드를 통해 api 사용을 줄일 수 있었습니다. tx마다 `check_vin/out()` 메소드를 호출하여 UTXO를 관리합니다. 마지막 블록까지 탐색이 끝나면 지금까지 UTXO의 합계를 구해 반환합니다.

```

def check_vin(self, tx):
    for vin in tx['vin']:
        if 'txid' in vin and (vin['txid'] in self.utxo) and (vin['vout'] == self.utxo[vin['txid']]['index']):
            if self.debug:
                print(f"[vin] {vin['txid']}[{vin['vout']}] {self.utxo[vin['txid']]['value']}")
            del self.utxo[vin['txid']]

def check_vout(self, tx):
    for idx, vout in enumerate(tx['vout']):
        if "address" in vout['scriptPubKey']:
            out_address = vout['scriptPubKey']['address']
            if(self.address == out_address):
                if self.debug:
                    print(f"[vout] {tx['txid']}[{idx}] {vout['value']}")
                self.utxo[tx['txid']] = {
                    "value": vout['value'],
                    "index": idx
                }

```

`check_vin()` 메소드는 기존 UTXO에 있는 tx가 사용되었는지 체크하여 사용되었다면 해당 tx를 UTXO에서 제거하도록 구성했습니다. `check_vout()` 메소드는 제공된 address를 향하는 out이 존재하는지 확인하여 있다면 UTXO에 추가하도록 구성했습니다.

```

def is_lastblock(self, block):
    if self.debug and self.start_block_height % 5000 == 0:
        print(f"[*] {self.start_block_height} / {self.total_height}")

    if 'nextblockhash' in block:
        self.next_block_hash = block['nextblockhash']
        self.start_block_height += 1
        return False

    return True

```

`is_lastblock()` 메소드는 `nextblockhash` 필드의 유무를 통해 현재 블록이 마지막 블록인지 확인합니다. 마지막이 아니라면 다음블록을 설정합니다.