

INFORMATION SYSTEM SECURITY

DARK WEB

Lab on Bitcoin

Author(s):

Gabriel PADIS

Anastasia DUCHESNE

Teacher(s):

MR. HAMONT

Contents

1	Launching bitcoind	2
2	Getting BTC	3
3	Working with a new address	4
3.1	Creating a new address	4
3.2	Displaying transactions	4
3.3	Balance at the end of transaction	6
3.4	Custom transaction	6
4	Signing a transaction	7
5	Sending a signed transaction	8
6	Sources	8

1 Launching bitcoind

`./bitcoin/bin/bitcoind`

This daemon is used to connect to the blockchain. Here it tries to initialize the local instance of it by retrieving all the bitcoin transactions ever made to save them on this machine to interact with them.

It can take a lot of time because the total amount of transactions is roughly equal to 90GB. So downloading all of it and checking it can take can be long.

```
2019-02-20T14:54:40Z init message: Rewinding blocks...
2019-02-20T14:54:40Z block index 30ms
2019-02-20T14:54:40Z init message: Loading wallet...
2019-02-20T14:54:40Z [default wallet] nFileVersion = 170100
2019-02-20T14:54:40Z [default wallet] Keys: 0 plaintext, 0 encrypted, 0 w/ metadata, 0 total. Unknown wallet records: 0
2019-02-20T14:54:40Z [default wallet] Performing wallet upgrade to 169900
2019-02-20T14:54:42Z [default wallet] keypool added 2000 keys (1000 internal), size=2000 (1000 internal)
2019-02-20T14:54:42Z [default wallet] Wallet completed loading in 4120ms
2019-02-20T14:54:42Z [default wallet] setKeyPool.size() = 2000
2019-02-20T14:54:42Z [default wallet] mapWallet.size() = 0
2019-02-20T14:54:42Z [default wallet] mapAddressBook.size() = 0
2019-02-20T14:54:42Z UpdateTip: new best=000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f height=0 version=0x00000001 log2_work=32.000022 tx=1 date='2009-01-03T18:15:05Z' progress=0.000000 cache=0.0MiB(0txo)
2019-02-20T14:54:42Z mapBlockIndex.size() = 1
2019-02-20T14:54:42Z nBestHeight = 0
2019-02-20T14:54:42Z Bound to [::]:8333
2019-02-20T14:54:42Z Bound to 0.0.0.0:8333
2019-02-20T14:54:42Z init message: Loading P2P addresses...
```

Figure 1: Initialization

Here it fails to do so because we have no internet connection:

```
2019-02-20T14:54:44Z ERROR: DeserializeFileDB: Failed to open file /home/secureit/y/.bitcoin/peers.dat
2019-02-20T14:54:44Z Failed to open mempool file from disk. Continuing anyway.
2019-02-20T14:54:44Z torcontrol thread start
2019-02-20T14:54:44Z Invalid or missing peers.dat; recreating
2019-02-20T14:54:44Z init message: Loading banlist...
2019-02-20T14:54:44Z ERROR: DeserializeFileDB: Failed to open file /home/secureit/y/.bitcoin/banlist.dat
2019-02-20T14:54:44Z Invalid or missing banlist.dat; recreating
2019-02-20T14:54:44Z init message: Starting network threads...
2019-02-20T14:54:44Z init message: Done loading
2019-02-20T14:54:44Z msghand thread start
2019-02-20T14:54:44Z opencon thread start
2019-02-20T14:54:44Z addcon thread start
2019-02-20T14:54:44Z dnsseed thread start
2019-02-20T14:54:44Z Loading addresses from DNS seeds (could take a while)
2019-02-20T14:54:44Z net thread start
```

Figure 2: Initialization

During the lab, we're going to use the regtest mode. In order to launch it, we use the following command line :

```
security@debian:~$ ./bitcoin-0.17.1/bin/bitcoind -regtest -daemon
Bitcoin server starting
```

Figure 3: Launching bitcoin server regtest

While launching `-daemon` with `-regtest` option, we're creating a regtest directory. Regtest is an option for "regression test mode". It is not a network, and is for private use only. In this mode, you're simulating bitcoin transactions only. Unlike Testmode which is a network, you have a full control over the environment, since you're choosing when to create your blocks on a number of nodes that you've decided.

2 Getting BTC

The command we have to run in order to get a balance of 50BTC on our account is :

```
./bitcoin-cli -regtest generate 101
```

This way we generate 101 blocks, which is the needed amount to get access to the coinbase transaction from block 1. This means that we need to generate 100 blocks in order to have 100 confirmations, only then we'll be able to access to the transaction from the first block.

The first 150 blocks pay a reward of 50BTC, which means that we only need to get access to the first block.

```
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest generate
[
  "1577e2037a954b3dfb6c7f0c10627e31bed8a3e509972997252da65cad843b6f",
  "01803d62bb86a7fdc9fe1535bf74607b7690fe291c8c31069f57dd3e21fa4cc7",
  "68e743df62f67e8f44da9890df0cdd0ada7161a4cce57dfc38550fd720ef38b2",
  "5568c5e91290006a2cd4a5e117f44c6308c46d6d9a953e46cf725fa939f57ee7",
  "4f6f1578d1b154b2bbb4d873384b814a9a66efdb585401d7467375c81d30f81d",
  "5ff1830cb763418b8f20267ee5d0661003a32ba9a6ab9357ca5bf4e1bd409fc2",
  "45dd390a482bbe3292da8db000eaaca966107d800a3b8d77b3f1b4884f2f0b7",
  "5a6e745f770505fa4482c3eeecf099006811093bcd5729e92d362c75b21825c",
  "03ed94e9710dc43c805e74e79ff9ee818a30622cc44a9dcc576024411883dbe7",
  "4107df06062aeca4e42e6d73841de4e0da2bc3914ffad3b4788a301e4ac9eb68",
  "382a1c7ca878acfaa6ed52fe464c1f395d9dc2bc83d0e1db0b2f65942a6a366d",
  "3153371826aafb9d048d4f9ccb71b7160e72063e7cc0c7f59b18fdc06e028797",
  "7fdfa998bd6624c65da5b7ad5b7abadfcf12a0f35b9b89fb13ecfd4939b9a480",
  "2a4b3eeb2436b1ccc2cc7414af0d759380f7d4cd85ff2cfa64f1828a21891e2d",
  "7313d928bd8a6ae1ec60c5413e601bd1910088134954c388c2cab67745989edb",
  "56cb8b15690ebc5ce7f1f2a3f0a84973a66c94367230b5a304f2aa242897d171",
  "3c944ca17e32ab6b098deaca580b0efcde36986e46e736617df4e73f981341e9",
  "2a18408f58376d5a233597146de70ef4418a9e0aae036960caa87c5dcab34a3f",

```

Figure 4: Generating 101 blocks

As you can see, with the command line `./bitcoin-cli -regtest getbalance` we now have 50BTC on our account :

```
root@debian:/home/security/bitcoin-0.17.1/bin# ./bitcoin-cli -regtest getbalance
50.00000000
```

Figure 5: Account balance

3 Working with a new address

3.1 Creating a new address

First, we have to create a new address with :

```
./bitcoin-cli -regtest getnewaddress
```

We store the output of the following command line in a shell variable called "NEW_ADDRESS".

```
security@debian:~/bitcoin-0.17.1/bin$ NEW_ADDRESS=$(./bitcoin-cli -regtest getnewaddress)
security@debian:~/bitcoin-0.17.1/bin$ echo $NEW_ADDRESS
2N4iJWrCyikDznpvDwZREMSG4fGek6eucuf
```

Figure 6: Setting a new address

Then, we transfer 10BTC to the new address.

```
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest sendtoaddress $NEW_ADDRESS 10.0
8c373a514ca3853f0ce8948f261415330563e0ee20197b44580921d7d10f8f97
```

Figure 7: Transferring 10BTC

The output is the transaction id.

3.2 Displaying transactions

In order to display the unconfirmed transaction, we use the function "**listunspent**" with the **argument 0**. If the argument isn't specified, the returned list will be empty since there is no transactions confirmed.

```
./bitcoin-cli -regtest listunspent 0
```

There are two transactions :

```
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest listunspent 0
[
  {
    "txid": "8c373a514ca3853f0ce8948f261415330563e0ee20197b44580921d7d10f8f97",
    "vout": 0,
    "address": "2NAX2dNngpUdKpAHPWeZdSTL9uemdQxWLaC",
    "redeemScript": "001497fbfa31938cf2c4b692e791959c80061be2070b",
    "scriptPubKey": "a914c231b5a297a438312c4162c70b2886fab3f8651b87",
    "amount": 39.99996260,
    "confirmations": 0,
    "spendable": true,
    "solvable": true,
    "safe": true
  },
]
```

Figure 8: First transaction

```

{
  "txid": "8c373a514ca3853f0ce8948f261415330563e0ee20197b44580921d7d10f8f97",
  "vout": 1,
  "address": "2N4iJWrCyikDznpvDwZREMSG4fGek6eucuf",
  "label": "",
  "redeemScript": "0014fc22b30ef0f44c8925f7eb8b03826781687fb000",
  "scriptPubKey": "a9147dc864be46f8b5f32c664da45759cf2a08a43c7b87",
  "amount": 10.00000000,
  "confirmations": 0,
  "spendable": true,
  "solvable": true,
  "safe": true
}

```

Figure 9: Second transaction

This gives us the following information :

- **txid** : transaction id
- **vout** : specific output from a transaction
- **address** : the address of the transaction. The second one matches with the one stored in `NEW_ADDRESS`.
- **amount**
- **confirmation** : number of confirmations. Since there was no block generated, it is set at 0.

The first transaction is a change output created by the `sendtoaddress` function. It is a transaction that returns bitcoins to the sender, preventing the transaction value from going through too many transaction fees. The second transaction is the one we're sending to the new address, which is like a second account for us. If it was someone else's account, we won't be able to see the second one.

Then, we generate a new block in order to confirm our transaction.

```

security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest generate 1
[
  "17697b14192adaf0b4cfcb2a1edc27dce76f160a58b7250df0236f9364c5b42a"
]

```

Figure 10: Generating one block

This action adds 50BTC on our account as well. Thus done, `./bitcoin-cli -regtest listunspent` displays the above two transactions, with confirmation set at "1" instead of "0", but also a third one :

```
{
  "txid": "95397fadac389831115757bb9fab39a3ab89395b2ad1eb3213b106cc6247522d",
  "vout": 0,
  "address": "myPgNBHjvTfdCA6hXWvtqFLeS6dyt3LDZ",
  "scriptPubKey": "2103680642b1391bc0995373494f2bf0b285b9843198993013235086600
0a5ee2c94ac",
  "amount": 50.00000000,
  "confirmations": 101,
  "spendable": true,
  "solvable": true,
  "safe": true
},
```

Figure 11: Third transaction

It is the coinbase transaction, the first transaction in a block. Its confirmation number is at "101", since we generated 101 blocks to get access to it. Finally, we use "**unset NEW_ADDRESS**" to clear the shell variable.

3.3 Balance at the end of transaction

```
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest getbalance
99.99996260
```

Figure 12: Balance after transaction

When we check the balance after our transaction, we see that its amount changed. It is normal :

- We added +50BTC when we generated a block for the confirmation
- We lost a small amount of BTC during our transaction because of the transaction fees. That's why before generating the confirmation block, our balance wasn't equal to 50BTC but only 49.99996260 (10.0 + 39.99996260).

3.4 Custom transaction

We create a new address :

```
security@debian:~/bitcoin-0.17.1/bin$ NEW_ADDRESS=$(./bitcoin-cli -regtest getnewaddress)
security@debian:~/bitcoin-0.17.1/bin$ echo $NEW_ADDRESS
2N4iJWrCyikDznpvDwZREMSG4fGek6eucuf
```

Figure 13: Setting a new address

```

security@debian:~/bitcoin-0.17.1/bin$ TXID=$(./bitcoin-cli -regtest listunspent
| python -c 'import json,sys;obj=json.load(sys.stdin);print obj[0]["txid"]')
security@debian:~/bitcoin-0.17.1/bin$ VOUT=$(./bitcoin-cli -regtest listunspent
| python -c 'import json,sys;obj=json.load(sys.stdin);print obj[0]["vout"]')
security@debian:~/bitcoin-0.17.1/bin$ NEW_ADDRESS=$(./bitcoin-cli -regtest getne
waddress)
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest createrawtransactio
n '{"txid": "'$TXID'", "vout": '$VOUT'}' "" "" {"$NEW_ADDRESS": 49.9999}' ''
0200000001113ff187f191551bf289cd60eb00d8f065800b5cf723980780952827726aa024000000
0000ffffffff01f0ca052a0100000017a9147d6b861eae30222c8385bdf078695cabf3236ca58700
000000
security@debian:~/bitcoin-0.17.1/bin$ RAW_TXT=$(./bitcoin-cli -regtest createraw
transaction '{"txid": "'$TXID'", "vout": '$VOUT'}' "" "" {"$NEW_ADDRESS": 49
.9999}' '')
security@debian:~/bitcoin-0.17.1/bin$ echo $RAW_TXT
0200000001113ff187f191551bf289cd60eb00d8f065800b5cf723980780952827726aa024000000
0000ffffffff01f0ca052a0100000017a9147d6b861eae30222c8385bdf078695cabf3236ca58700
000000

```

Figure 14: Creation of a raw transaction

A raw transaction is a complete transaction in a binary format, represented using hexadecimal. It gives a low-level access to the transaction creation. It is in a serialized format. First, in order to create our raw transaction, we need the **txid** and **vout** parameters of our previous transaction. To gather this information, we apply a python function that retrieves the first txid and vout fields from **listunspent** function output.

Our raw transaction inputs are a transaction ID, output, address, and number of BTC we're sending. We're sending only 49.9999BTC and not 50BTC to include the fee of 0.0001BTC. We need to pricise this since if we don't pay enough attention to it, we can pay huge fee amounts. When we echo the RAW_TX shell variable, we notice that it actually returns a hexadecimal.

4 Signing a transaction

```

security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest signrawtransactionw
ithwallet $RAW_TXT
{
  "hex": "0200000001113ff187f191551bf289cd60eb00d8f065800b5cf723980780952827726a
a024000000004847304402204e5b9702609c193a2d39e3c8006e4a5917797da52437aac19c44abcd
86877c3c022035de4d18bd15e1e73fef649c193a219863daabcd66f491f177ee4f7b680c2eb601ff
ffffff01f0ca052a0100000017a9147d6b861eae30222c8385bdf078695cabf3236ca58700000000
",
  "complete": true
}
security@debian:~/bitcoin-0.17.1/bin$

```

Figure 15: Signing the transaction

We sign the serialized transaction with a private key stored in the wallet.

In order to save the hex, we used the same python functions as before that permits us to retrieve Json information.


```

security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest signrawtransactionw
ithwallet $RAW_TXT | python -c 'import json,sys;obj=json.load(sys.stdin);print o
bj["hex"]'
02000000001113ff187f191551bf289cd60eb00d8f065800b5cf723980780952827726aa02400000
004847304402204e5b9702609c193a2d39e3c8006e4a5917797da52437aac19c44abcd86877c3c02
2035de4d18bd15e1e73fef649c193a219863daabcd66f491f177ee4f7b680c2eb601ffffffff01f0
ca052a0100000017a9147d6b861eae30222c8385bdf078695cabf3236ca58700000000
security@debian:~/bitcoin-0.17.1/bin$ SIGNED_RAW_TXT=$(./bitcoin-cli -regtest si
gnrawtransactionwithwallet $RAW_TXT | python -c 'import json,sys;obj=json.load(s
ys.stdin);print obj["hex"]')

```

Figure 16: Saving the signed hex format

We have created and signed a transaction, but we didn't have send nor confirmed it yet. This means that we haven't spent anything yet, and by unsetting the shell variable `SIGNED_RAW_TX`, we can delete it.

5 Sending a signed transaction

```

security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest sendrawtransaction
$SIGNED_RAW_TXT
6c96c9e7c72fe5b1fa3755897c13447515b85bf4c0bfc15e2d7a657b3f1283e1
security@debian:~/bitcoin-0.17.1/bin$ ./bitcoin-cli -regtest generate 1
[
  "26084aecf087c58ee1aaf163afe173b23acf5154b806d90ebb2b1fee4e6a75ab"
]

```

Figure 17: Sending and confirming the transaction

We send the raw transaction to the connected node. We generate a block to confirm the transaction.

6 Sources

- <https://bitcoin.org/en/developer-examples>
- <https://stackoverflow.com/questions/1955505/parsing-json-with-unix-tools>