ECE PARIS · LYON
ÉCOLE D'INGÉNIEURS

ING4 SI01

Information System Security

Dark Web

Lab on Bitcoin

Author(s):
Gabriel PADIS
Anastasia DUCHESNE

Teacher(s):
Mr. Hamont

Paris - Semester 7 - March 5, 2019

Contents

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.

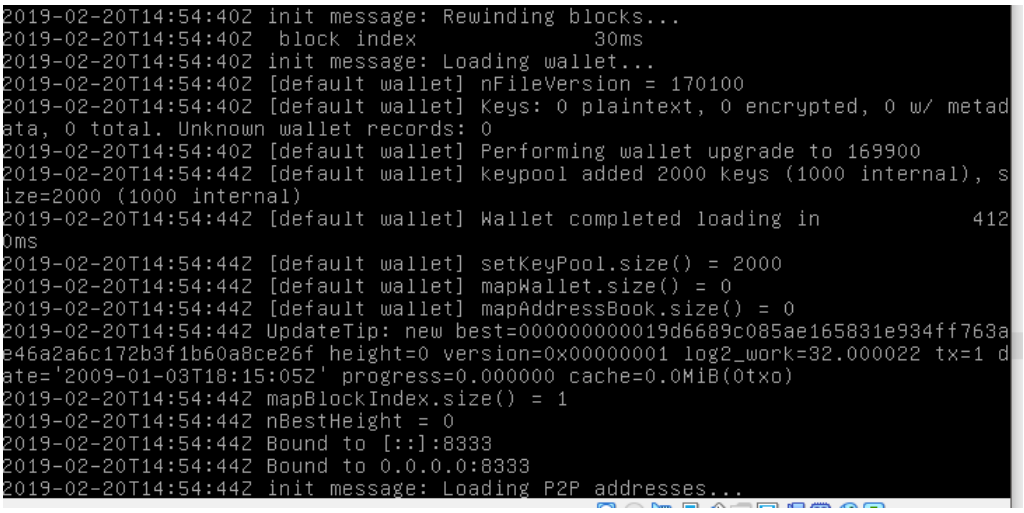

Figure 1: Initialization

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

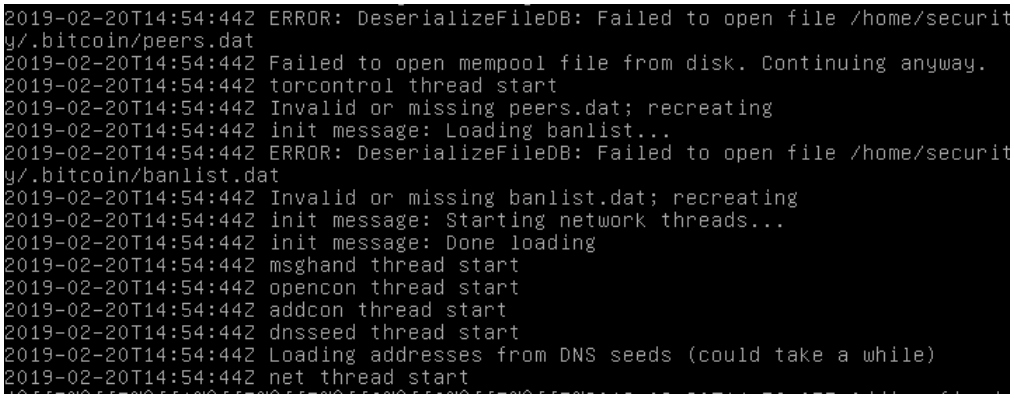

Figure 2: Initialization

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

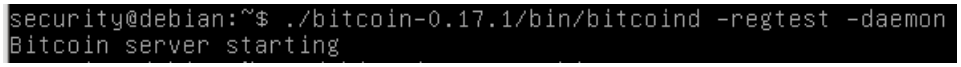

Figure 3: Launching bitcoin server regtest

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.

Figure 4: Generating 101 blocks

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

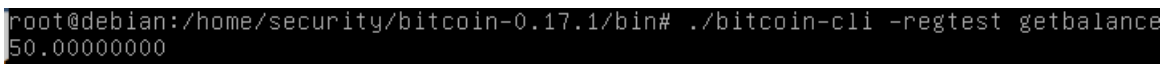

Figure 5: Account balance

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=$(./bitcoin-cli -regtest getne
waddress)
security@debian:~/bitcoin-0.17.1/bin$ 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 "**linstunspent**" 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": "2NAx2dNnqpUdKpAHPWeZdSTL9uemdQxWLaC",
    "redeemScript": "001497fbfa31938cf2c4b692e791959c80061be2070b",
    "scriptPubKey": "a914c231b5a297a438312c4162c70b2886fab3f8651b87",
    "amount": 39.99996260,
    "confirmations": 0,
    "spendable": true,
    "solvable": true,
    "safe": true
  },
```

Figure 8: First transaction

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 bitoins 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.

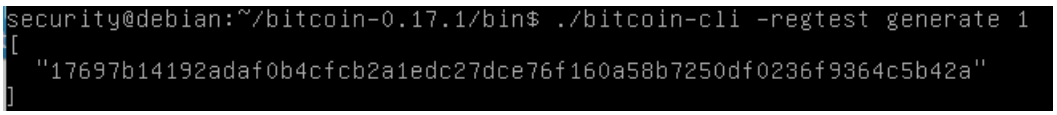

Figure 10: Generating one block

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

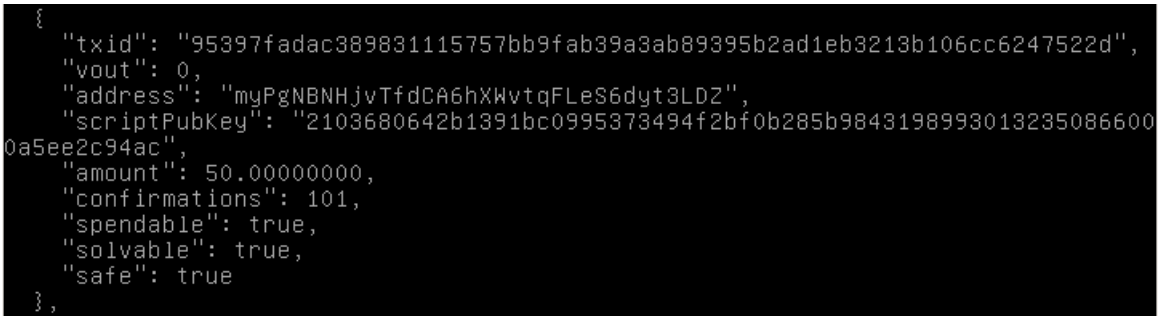

Figure 11: Third transaction

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

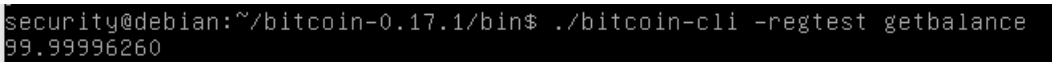

Figure 12: Balance after transaction

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 blanace wasn't equal to 50BTC but only 49.99996260 (10.0 + 39.99996260).

3.4 Custom transaction

We create a new address :

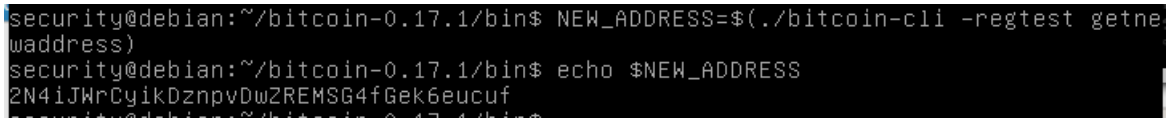

Figure 13: Setting a new address

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

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.

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

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