Peer to Peer Systems and Blockchains
Academic Year 2020/2021
Final Term

Gianmaria di Rito 544013

June 2021

1 Ethereum e Smart Contracts

Given the smart contract Mayor.sol associated to this assignment, the student is asked to answer to the following questions:

1. Implement the `open_envelope()` and `mayor_or_sayonara()` functions. Feel free to make any change to the smart contract state attributes.

   Look at **mayor.sol** in the folder.

2. Evaluate the cost in gas of the functions provided by the smart contract;

   (a) provide 2 or 3 cost variations of the `mayor_or_sayonara()` function, for example by fixing the quorum and varying the number of losing voters, or varying the quorum, or any other method.

   Cost variation of the `mayor_or_sayonara()` function:

Participants	In favor	Against	Transaction cost	Execution cost
2	1	1	47187 gas	40915 gas
5	3	2	61894 gas	70622 gas
10	1	9	120524 gas	219775 gas

   (b) cost in gas of the other functions provided by the smart contract:

Name	Transaction cost	Execution cost
compute_envelope	23664 gas	1496 gas
cast_envelope	54895 gas	31447 gas
open_envelope	159272 gas	137616 gas

3. What are the security considerations and potential vulnerabilities related to the `mayor_or_sayonara()` function? List and explain them.

   - **Reentrancy vulnerabilities**: A possible vulnerability within the `Mayor_or_sayonara()` function there would have been if, to refund the voters who expressed the "losing vote"(nay in case of confirmation, yay in case of rejection), and to pay the souls to the mayor in case of victory and to the escrow address in case of defeat, the function **call.value()** would be used instead of the more secure **transfer()**. This is because we would be exposed to re-entrancy vulnerability. A function is reentrant if its execution
     - can be interrupted in the middle
     - initiated over (re-entered)
     - both runs complete without errors

     In general re-entrancy may result in unexpected behaviors and loops of invocations which eventually consume all the gas. In solidity the functions are not reentrant and the task of making them reentrant is left to the developer. In fact, by exploiting the not re-entrancy of the `Mayor_or_sayonara()` function, for example, an attacker could have built a loop of calls so as to exhaust all the gas. A solution to this vulnerability is to use **transfer()** which sets a gas limit of 2300 units. This fixed gas limit prevents the receiver to execute too much code and therefore avoids the consumption of all available gas.
   - **Front Running vulnerabilities**: Ethereum nodes pool transactions and form them into blocks. All the transactions are visible in the mempool for a short before they are executed, i.e. inserted in a block by a miner. Hence, their public content can be "observed" by attackers. The miner who solves the block also chooses which transactions execute based on the gasPrice of each transaction. The attacker can get the data from transactions before they are inserted in a block, can create a transaction of their own with a higher gas price so that the attacker's transaction is included in a block before the original. In this way, a potential attacker can become aware of the result of the vote before it is inserted into the blockchain and made definitive and then use this information for his own personal purposes.

4. The `compute_envelope()` function is an helper to compute an envelope. Why is its presence an issue if the smart contract would be deployed to the Ethereum network?

   A pure function like `compute_envelope()` is a function in which the output is entirely dependent on the input parameters. They can never read from, nor store data to the blockchain. This makes them suitable as utility functions. However, if the smart contract would be deployed to the Ethereum network if you're making a call to the pure function from another smart contract as part of a transaction execution, the execution will be done by the whole network and therefore, the input parameters of the function will be visible to everyone. To keep something private, the data should be encrypted first and should never be sent the key to the blockchain. This leads to understand that the pure modifier does not make input data secret.

2 BITCOIN and the Lightning network

Answer the following questions:

1. One of the main problem that the inventors of Bitcoin faced is that of "double spending". Imagine that you have 1 BTC in your wallet, you decide to buy a car, go to the car dealer, and you pay with your BTC. After having received the car, you decide to buy a sailboat, but you do not have any more money. Would you be able to buy the sailboat by somehow 'doubles spending' the 1 BTC you had earlier? Show how an attacker can perform a double spending attack and which are the countermeasures that Bitcoin defines to protect the system from this attack.

   (a) We call the attacker Bob and say that in the normal way he goes to the car dealer and spends his 1 BTC on buying the car. Knowing that each transaction is definitively accepted after 6 confirmations, according to the longest chain rule, he waits for the amount of time necessary for this to happen so that, the car dealer receives his 1 BTC, and he the car.

   (b) Bob wanting to perform a double spending attack by "reversing" the longest chain and being a malicious miner, performs the mining in **stealth mode** and therefore do not broadcast its chain to the rest of the nodes. Within his "stealth blockchain" he excludes the spending he made on the honest branch of the blockchain and therefore he will still have the 1 BTC spent on the car.

   (c) Later, as soon as it creates a chain longer than the main one, it will transmit its version of the chain to the rest of the network which will be forced to accept it due to the longest chain rule. This will lead to invalidating the transactions present only on the abandoned

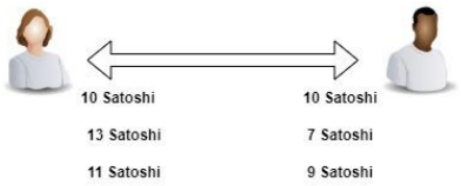
chain including the relative one to the purchase of the car by Bob who will have the car of his property and the 1 BTC spent to buy it, still present in his wallet ready to be spent on the purchase of the sailboat.

So, in order for the attack to be successful Bob needs to be able to create, by himself, a chain longer than the main one in which all the other miners in the world cooperate. It is therefore understood that the probability of success of Bob's double spending attack in Bitcoin is infinitesimal, unless he is able to solve the PoW puzzles at a speed that approaches all the other miners combined. This is only possible if he has a hashing power greater than or equal to 51% of the total power.
Other more naive types of double spending attacks are:

- Send two transactions to the network by spending the same BTC twice. Then both transactions go to the miners MemPool and one of them will be inserted into the next block by an honest miner while the second will be considered invalid by the miner who extracts the next block. Even if the two transactions are validated simultaneously by two different miners and therefore a blockchain fork is created, only one of the two transactions will be successful since sooner or later one of the two branches will have the upper hand over the other and therefore only one of the two blocks (and the corresponding transaction) will be inserted in the chain at long-term.
- the attacker is a malicious miner and tries to mine a block in which he wants to enter the two transactions that spend the same BTC. This attack fails simply because the other miners refuse the block. This is the most surreal of the three attacks given that no miner would waste mining power in a block that will surely be rejected and therefore will not be able to obtain the block reward.

As can be seen from these three cases none of them is easily applicable in Bitcoin thanks to the longest chain rule or the simple consensus mechanism. In fact they are made almost impossible (first case), impractical (second case) or even harmful for the attacker (third case).

2. Alice and Bob have several trade relations and decide to open a channel of the Bitcoin's Lightning network. Initially each of them decides to fund the channel with10 Satoshi. Then, they perform some transactions changing their balances in the channel, as shown in the following picture:



After the last transaction, Alice tries to cheat Bob, by publishing the second transaction, which is more favorable to her, because the state of the channel, after the second transaction, assigns 13 Satoshi to Alice and 11 Satoshi to Bob, while, after the last transaction, 11 are assigned Satoshi to Alice and 9 Satoshi to Bob. Describe how Bob can avoid that the scam is successful, highlighting the functionalities of the blockchain that are exploited to prevent the scam.

The Lightning network protocol ensures that neither party can cheat the other and does so by leveraging the blockchain. The idea behind the mechanism that protects against cheating is based on the fact that both parties must have an interest in publishing the most updated balances of the closing transactions. To enforce this principle, it introduces a disincentive which consists in the fact that if someone violates this rule, the other party can punish him and withdraw all the amount registered in the channel. The anti-cheating protocol of the Lightning protocol uses two mechanisms:

- **Time Lock**: lock bitcoin in the transaction output make them spendable only at some point in the future. There are two type of time lock:
  - CheckLockTimeVerify (CLTV): indicates a time lock that will be released at a specific date / time;
  - CheckSequenceVerify (CSV):indicates a time lock that will be released to N locks from now on.
- **Hash Preimages (Secrets)**: a "secret" or "preimage" is a string randomly generated and impossible to guess. The sequence is cryptographically hashed through an hashing function so anyone who knows the secret can easily reproduce the hash but not the other way round (pre-image resistance). A hashlocked transaction include the hash of the preimage in the output script and the output of the transaction may be unlocked only if the preimage of the hash is presented in the unlockchain script.

Having introduced the functionalities of the blockchain that will be exploited by Bob to avoid Alice's cheating, let's show in detail what happens.

Let's start with the situation in which:

- Alice have 13 satoshi
- Bob have 7 satoshi

(a) Alice sends a payment of 2 satoshis to Bob

(b) the channel status is updated by assigning 11 satoshis to Alice and 9 to Bob. Each party before generating the new commitment transaction:

- send the pre-image used in the previous commitment transaction to the other party, so reveals the previous secret;
- generate a new secret, that is a new pre-image;
- send the hash of the new secret to the other party;
- generates a new commitment transaction with the new balances (Alice: 11 satoshi, Bob: 9 satoshi) and the new hash (received from the other party) and sends it to the other party.

Taking into consideration the commitment transaction sent by Bob to Alice and signed by Bob; we say that:

- the output that transfers bitcoin to Bob is a simple locking script that can be instantly redeemed by Bob, if Alice decides to sign and publish the transaction on the blockchain;
- the output that transfers bitcoin to Alice includes anti-cheat mechanisms as she can redeem that output only after some time (time lock) for example 1000 blocks. During this time Bob can redirect the transfer to himself if he detects that Alice is cheating, but he needs the pre-image to do it (hash lock).
- a symmetric transaction in which only the output locking scripts differ, is sent from Alice to Bob.

(c) This is the moment in which Alice decides to cheat and she does so by registering on the blockchain not the last transaction (Alice: 11, Bob: 9) but the previous, therefore, the one in which Alice has 13 satoshi and Bob has 7 satoshi. But Bob now knows the pre-image used by Alice in the previous transaction (it was revealed to him by herself in the previous step).

(d) Bob signs the new transaction and redeems his satoshis immediately.

(e) Alice instead, has to wait 1000 blocks before getting her satoshi and in this time interval Bob detect that Alice has cheated by going to register the previous transaction. So he uses the pre-image of the previous transaction to redeem all the satoshi of the channel and thus punishing Alice (this type of check must be done regularly to reveal any cheating behavior by the other party and can also be carried out by a third party).

So thanks to the Time lock mechanism Alice could not carry out the cheat in a short time that did not allow Bob to notice the deception and thanks to the hash lock mechanism Bob was able to punish Alice by redeem the entire contents of the channel.