Ether Authority

SMART CONTRACT AUDIT REPORT

For

PEERPRIME ICO (Order # OCT042018A)

Prepared By: Yogesh Padsala

Prepared on: 04/10/2018

audit@etherauthority.io

Prepared For: PRIME NETWORK

https://www.peerprime.io

Table of Content

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2. Overview of the audit

The project has following file:

- PEERPRIME-ICO.sol

It contains approx **610** lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, which increases the readability.

The audit was performed by Yogesh Padsala, from Ether Authority Limited. Yogesh has extensive work experience of developing and auditing the smart contracts.

The audit was based on the solidity compiler 0.4.25+commit.59dbf8f1 with optimization enabled compiler in remix.ethereum.org

This audit was also performed the verification of the details exists in the main website: https://www.peerprime.io

3. Attacks tested on the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

3.1: Over and under flows

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, and all the functions have strong validations, which prevented this attack.

3.2: Short address attack

Although this contract **is not vulnerable** to this attack, it is highly recommended to call functions after checking validity of the address from the outside client.

3.3: Visibility & Delegatecall

No such issues found in this smart contract and visibility also properly addressed.

3.4: Reentrancy / TheDAO hack

Use of “require” function and Checks-Effects-Interactions pattern in this smart contract mitigated this vulnerability.

3.5: Forcing ether to a contract

Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability

3.6: Denial of Service (DoS)

There is no process consuming loops in the contracts which can be used for DoS attacks. Also, there is no progressing state based on external calls, and thus this contract is not prone to DoS.

4. Good things in the smart contract

4.1 Two levels of Administration

From the first sight, it is impressive to have two levels of administrations with varying authority level. This is really helpful especially when the organization will have many administrative parties.

4.2 Declaring the variable as constant

If the state variables are not supposed to be changed, then it is good practice to declare them as constant. It saves less gas compared to the variables which are not declared as constant.

4.3 Minimum data stored in the contract

This contract stores very minimum amount of data in the smart contract, which is really good thing as that minimize the gas cost to users of the contract down the road.

4.4 Good validations

This contract processes loop with good validations as well as functions are having good require conditions.

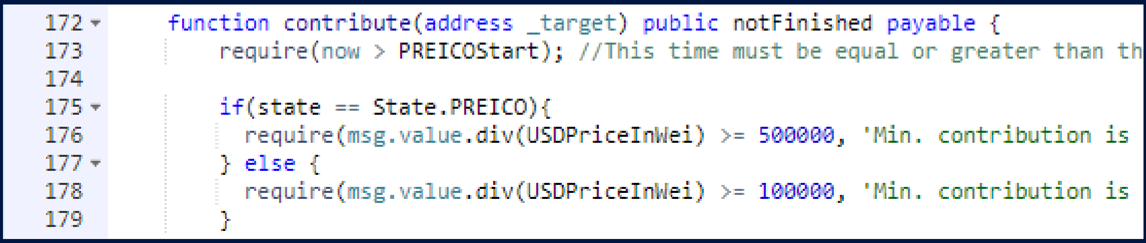
4.5 Control over crowdsale elements

The ability for admin to change softcap and set token price is good thing, as there might be various cases where those things need to be changed down the road.

Also, the ability for admin to transfer the tokens which might exist in the contract after ICO is over, is foreseen feature as that might be very useful in the future.

4.4 Good things in the code

- contribute function

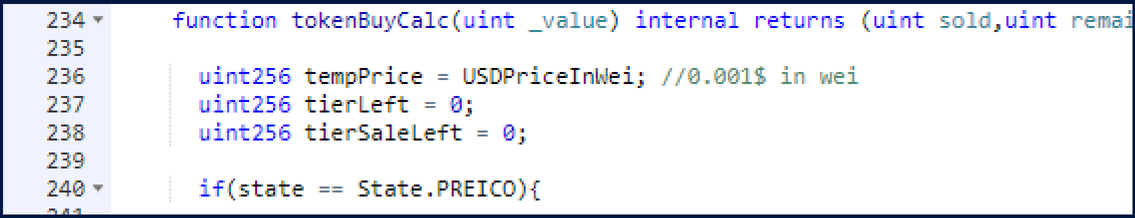


```
    function contribute(address _target) public notFinished payable {
        require(now > PREICOStart); //This time must be equal or greater than th

        if(state == State.PREICO){
          require(msg.value.div(USDPriceInWei) >= 500000, 'Min. contribution is
        } else {
          require(msg.value.div(USDPriceInWei) >= 100000, 'Min. contribution is
        }
```

This is the most important function for the ICO, which checks for the time frame, token amounts, whitelisting, etc. Most importantly, it transfers the token at last after doing all other processing, which is a very good thing.

- tokenBuyCalc function



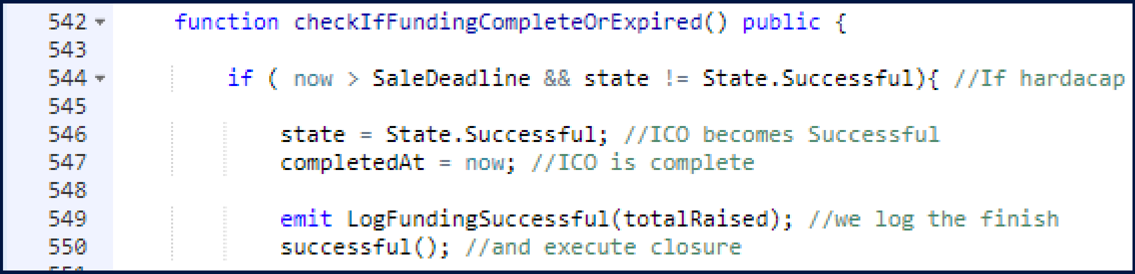
```
    function tokenBuyCalc(uint _value) internal returns (uint sold,uint remai

      uint256 tempPrice = USDPriceInWei; //0.001$ in wei
      uint256 tierLeft = 0;
      uint256 tierSaleLeft = 0;

      if(state == State.PREICO){
```

This function calculates token amount to send to contributor, according to various stages or crowdsale.

- checkIfFundingCompleteOrExpired function



```
    function checkIfFundingCompleteOrExpired() public {

        if ( now > SaleDeadline && state != State.Successful){ //If hardacap

            state = State.Successful; //ICO becomes Successful
            completedAt = now; //ICO is complete

            emit LogFundingSuccessful(totalRaised); //we log the finish
            successful(); //and execute closure
```

It checks whether ICO is successful or hardcap has reached or not. And changes the state state accordingly.

5. Critical vulnerabilities found in the contract

Critical issues that could damage heavily the integrity of the contract. Some bug that would allow attackers to steal ether is a critical issue.

=> No critical vulnerabilities found

6. Medium vulnerabilities found in the contract

Those vulnerabilities that could damage the contract but with some kind of limitations. Like a bug allowing people to modify a random variable.

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

Those do not damage the contract, but it's better to resolve them and make the contract more efficient, optimized and clean.

7.1: Revert-function in the body of the conditional operator if

At line #532, the construction if (condition) {revert();} is used instead of require(condition);

Please note, this does not raise any vulnerability, as both are equivalent. But it just increases the good readability.

7.2: Implicit visibility level

Again, this is not a big issue in the solidity. Because if you do not put any visibility, then it will automatically take "public". But it is good practice to specify visibility at every variables and functions.

Line numbers: #121 and #131

Please put "public" visibility at the variables at above lines.

7.3 Costly loop possibility

The function contribute() implements ‘while’ loop. In ideal condition that does not raise any problem as there will not be many iterations. But still there is possibility where loop can go out of control and which max out the block’s gas limit making the contract stuck. Please try to put logic to restrict more potential iterations.

7.4 Unchecked Math

Safemath library is included, which is good thing. But at some place, it is not used.

This is not a big issue, as validations are done well. But it is good practice to use it at all the mathematical calculations. Following lines does not have safemath used.

#366, #392, #425, #337, #275, #306, #118, #242, #244, #304, #517, #119, #454, #273, #394, #368, #485, #335, #519, #423, #487, #456

Please implement Safemath at those places.

8. Discussions and improvements

8.1 Putting higher degree of control

It is good idea to put ability for owner to put safeguard in the code. So, let’s say for example, there would be any un-intended event occurred in the future, then owner can put a safeguard and which prevents all the process from happening until the issue is resolved.

This can be easily achieved by declaring a variable for that, which can be used in all the functions. Admin can make this variable true or false. Another way is to create modifier for that and use it in every function.

8.2 Declaring custom function for require () validation

It is good idea to create a custom function for the require condition, and make an Event to fire in case of failed “required” validation. It helps client to better understand why any possible error occurred.

```
bool internal dorequireRevert; // <=== IMPORTANT DEBUG/REVERT SWITCH
                                        // false => keep going but emit RequireFailed
                                        // true  => do the revert
function dorequire(bool testresult, string message) internal {
        if (!testresult) {
                emit RequireFailed(message);
                if (dorequireRevert) {
                        require(false, message);
                }
        }
}
dorequire (1 != 0, "one is not equal zero!");  //use it everywhere as like this!
```

8.3 Timestamp dependence awareness

This contract depends on the timestamp as place like line number #547. There is nothing wrong in that but please be aware that the timestamp of the block can be slightly manipulated by the miner.

8.4 Token price variation

Pricing of the token is calculated as per USD. Now considering Ether’s value in USD keeps fluctuating. Hence, please be aware that there will be difference in the amount of tokens being sent for certain Ether contribution.

However, admin can keep updating this pricing. But still there will be the variation to be aware of. If your business logic allow, then token pricing in ETH would a lot easier.

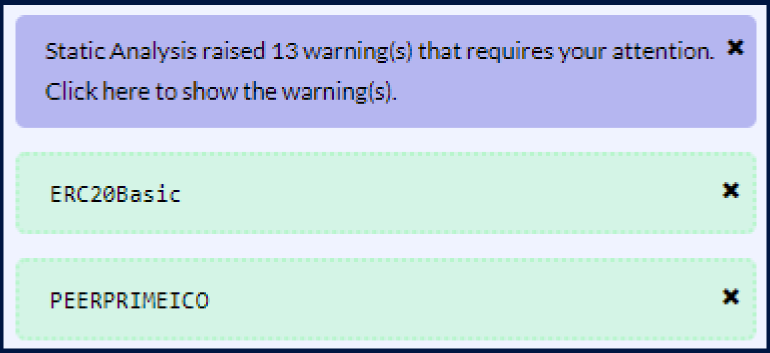
```
1 ETH = 1000 Token          // is more easier

1 Token = $0.001            // prone to variation
```

9. Summary of the Audit

Overall the code performs good data validations as well as meets the calculations according to the information presented in the website: https://www.peerprime.io

The compiler also displayed 13 warnings:



Static Analysis raised 13 warning(s) that requires your attention. ✖
Click here to show the warning(s).

ERC20Basic ✖

PEERPRIMEICO ✖

Now, we checked those warnings are due to their static analysis, which includes like gas errors and all. So, it is important to supply correct gas values while calling various functions.

Those warnings can be safely ignored as should be taken care while calling the smart contract functions.

Please try to check the address and value of token externally before sending to the solidity code.

It is also encouraged to run bug bounty program and let community help to further polish the code to the perfection.