SMART CONTRACT AUDIT REPORT

For

EZOOW (Order # OCT142018A)

Prepared By: Yogesh Padsala

Prepared on: 14/10/2018

audit@etherauthority.io

Prepared For: EZOOW TOKEN

https://www.ezoow.com

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:

- EZOOW.sol

It contains approx **335** 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 detail of the whitepaper located at: https://www.ezoow.com/Ezoow-Whitepaper.pdf

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 In-built ICO functionality

It is always better that ERC20 token have some sort of mechanism which enables users to receive tokens immediately by sending ether to the contract address. And having fully fledged ICO facility is even better.

4.2 Ability to mint new tokens

This is foreseen features as there might be chances where admin need to generate new tokens. This could be due to any un-expected events such as lost of tokens in error, etc.

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 start/stop and ICO 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

- transferFrom function

```
146     function transferFrom(address _from, address _to, uint256 _value) publi
147         require(_value <= allowance[_from][msg.sender]);     // Check allow
148         allowance[_from][msg.sender] = allowance[_from][msg.sender].sub(_va
149         _transfer(_from, _to, _value);
150         return true;
151     }
```

This function checks the amount of sender as well as allowance before doing the transfer.

- _transfer function (internal)

```
241     function _transfer(address _from, address _to, uint _value) internal {
242         require (_to != 0x0);                               // Prevent tra
243         require (balanceOf[_from] >= _value);               // Check if th
244         require (balanceOf[_to].add(_value) >= balanceOf[_to]); // Check f
245         require(!frozenAccount[_from]);                     // Check if se
246         require(!frozenAccount[_to]);                       // Check if re
```

This functions does all the required checking of the inputs before doing any transfer. This Checks-Effects-Interactions pattern is very good thing.

- Fallback function (Payable)

```
253     function () payable public {
254         require(endTime > now);
255         require(startTime < now);
256         uint ethervalueWEI=msg.value;
257         // calculate token amount to be sent
258         uint256 token = ethervalueWEI.mul(exchangeRate); //weiamount * pri
259         tokensSold = tokensSold.add(token);
260         _transfer(this, msg.sender, token);              // makes the tra
261         forwardEherToOwner();
```

This contract accepts incoming ether only if the ICO is running, which is determined by its start and end dates. Then it calculates the token price according to the set exchange rate. And does all the process first and transfers the tokens at the last, which is really good practice.

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 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. And we confirm that, this does not raise any underflow or overflow, but it is good practice to use SafeMath library at all the mathematical calculations. Following lines does not have safemath used.

#74, #75, #76, #77

Please implement Safemath at those places.

7.2: Compiler version need to be updated

Please use latest solidity version to compile the code, which currently is: **0.4.25**

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

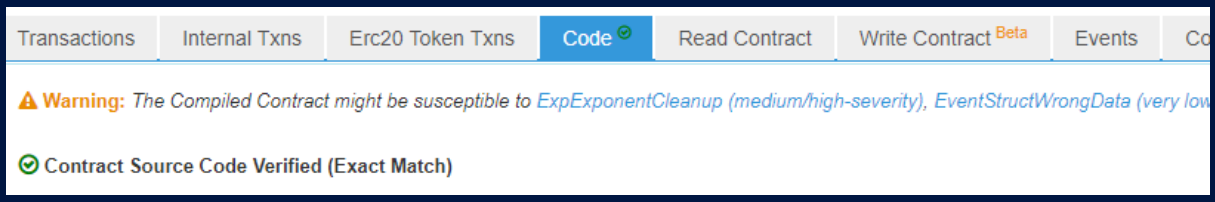
This contract depends on the timestamp as place like line number #254, #255, #293. There is nothing wrong in that but please be aware that the timestamp of the block can be slightly manipulated by the miner.

8.4 Use of self-destruct function

It many times happens, where contract owner would need to upgrade the contract or to add any important feature in the contract.

So, the only way that can be possible by creating brand new contract and destroying the old one. And that time, self-destruct comes to help.

8.5 Solidity compiler bugs

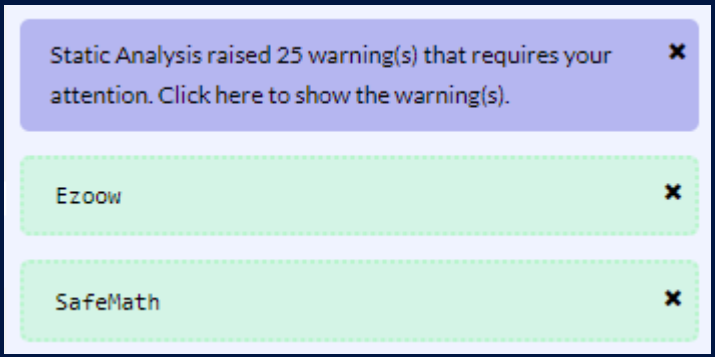


Etherscan.io is displaying warning message for the compiler bugs, which were existed in compiler version 0.4.24. Now, we carefully checked that those warnings do not have any impact on the use case of this contract code.

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 whitepaper at the company website: https://www.ezoow.com

The compiler also displayed 25 warnings:



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.