

SMART CONTRACT AUDIT REPORT

For

Eye Network token (Order #FO711C5)

Prepared By: Yogesh Padsala

Prepared For: Eye Network token

Prepared on: 07/09/2018

Revised on: 12/9/20018

Table of Content

1. Disclaimer
2. Overview of the audit
3. Attacks made to the contract
4. Good things in smart contract
5. Critical vulnerabilities found in the contract
6. Medium vulnerabilities found in the contract
7. Low severity vulnerabilities found in the contract
8. Summary of the audit

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 4 file EnbManageContract.sol, ERC20.sol, EyeNetworkToken.sol and SafeMath.sol. It contains approx 325 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation.

3. Attacks made to 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

An overflow happens when the limit of the type variable uint256, 2^{256} , is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract $0 - 1$ the result will be $= 2^{256}$ instead of -1. This is quite dangerous.

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

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

3.3: Visibility & Delegatecall

It is also known as, The Parity Hack, which occurs while misuse of Delegatecall.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume “Public” visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

3.4: Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed.

Use of “require” function in this smart contract mitigated this vulnerability.

3.5: Forcing ether to a contract

While implementing “selfdestruct” in smart contract, it sends all the ether to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the “Required” conditions. Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability.

4. Good things in smart contract

4.1 EnbManageContract.sol

4.1.1 buyTokens Function:-

```
64
65     function buyTokens(address beneficiary, uint256 weiAmount) internal {
66         require(beneficiary != 0x0);
67
68         // calculate token amount to be sent
69         uint256 tokens = weiAmount.mul(rate);
70
71         if(remain_investor.sub(tokens) <= 0){
72             uint256 real = remain_investor;
73             remain_investor = 0;
```

- Here you are checking address parameter of beneficiary.

4.1.2 transferForVote Function:-

```
174
175     function transferForVote(address beneficiary, uint256 tokenamount) public {
176         require(beneficiary != 0x0);
177         if(remain_vote.sub(tokenamount * 1 ether) <= 0){
178             uint256 real_vote = remain_vote;
179             remain_vote = 0;
180             transferToken(beneficiary, real_vote);
181         }else{
182             remain_vote = remain_vote.sub(tokenamount * 1 ether);
183             transferToken(beneficiary, tokenamount * 1 ether);
184         }
185     }
```

- Here first check the address of beneficiary then process goes ahead.

4.2 ERC20.sol

=> It is Interface for ERC20 Standard

4.3 EyeNetworkToken.sol

• 4.3.1 transferForVote Function:-

```
50     function transfer(address dst, uint256 wad) public returns (bool) {
51         assert(_balances[msg.sender] >= wad);
52
53         _balances[msg.sender] = _balances[msg.sender].sub(wad);
54         _balances[dst] = _balances[dst].add(wad);
55
56         emit Transfer(msg.sender, dst, wad);
57
58         return true;
59     }
```

- First code is checking user has sufficient balance, then it allows transfer.

- **4.3.1 transferFrom Function:-**

```
function transferFrom(address src, address dst, uint256 wad) public returns (bool) {  
    assert(_balances[src] >= wad);  
    assert(_approvals[src][msg.sender] >= wad);  
  
    _approvals[src][msg.sender] = _approvals[src][msg.sender].sub(wad);  
    _balances[src] = _balances[src].sub(wad);  
    _balances[dst] = _balances[dst].add(wad);  
  
    emit Transfer(src, dst, wad);  
  
    return true;  
}
```

- In this function, it is first checked the balance of sender as well as the allowance of the sender, then this script allow user to transfer tokens.

4.4 SafeMath.sol

=> This is standard SafeMath library.

4.5 Ownable.sol

```
38 function transferOwnership(address newOwner) onlyOwner public {  
39     require(newOwner != address(0));  
40     emit OwnershipTransferred(owner, newOwner);  
41     owner = newOwner;
```

- In this function you are checking address parameter of newOwner if it's correct then you allow to assign in owner.

5. Critical vulnerabilities found in the contract

5.1 EnbManageContract.sol

=> No critical vulnerabilities found

5.2 ERC20.sol

=> No critical vulnerabilities found

5.3 EyeNetworkToken.sol

=> No critical vulnerabilities found

5.4 SafeMath.sol

=> No critical vulnerabilities found

6. Medium vulnerabilities found in the contract

6.1 EnbManageContract.sol

=> No medium vulnerabilities found

6.2 ERC20.sol

=> No medium vulnerabilities found

6.3 EyeNetworkToken.sol

=> No medium vulnerabilities found

6.4 SafeMath.sol

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

7.1 EnbManageContract.sol

=> No Low vulnerabilities found


7.2 ERC20.sol

=> No Low vulnerabilities found

7.3 EyeNetworkToken.sol

7.3.1 Unnecessary condition in transfer function


```
43     function transfer(address dst, uint256 wad) public returns (bool) {  
44         require(dst != 0x0);  
45         require(wad > 0);  
46         require(_balances[msg.sender] >= wad);  
47         _balances[msg.sender] = _balances[msg.sender].sub(wad);  
48         _balances[dst] = _balances[dst].add(wad);  
49         emit Transfer(msg.sender, dst, wad);
```



=>Here you are checking the wad value is above the 0. but in solidity that is not possible to track negative value so remove this line. You are checking the balance of user that's sufficient validations.

7.3.2 unnecessary condition in transferFrom function


```
53     function transferFrom(address src, address dst, uint256 wad) public returns (bool) {  
54         require(src != 0x0);  
55         require(dst != 0x0);  
56         require(wad > 0);  
57         require(_balances[src] >= wad);  
58         require(_approvals[src][msg.sender] >= wad);  
59         approvals[src][msg.sender] = approvals[src][msg.sender].sub(wad);
```



=>Here you are checking the wad value is above the 0.but in solidity that is not possible to track negative value so remove this line. You are checking the balance of user that's sufficient validations.

7.3.3 Unnecessary condition in approve function

```
66     function approve(address guy, uint256 wad) public returns (bool) {  
67         require(guy != 0x0);  
68         require(wad > 0);  
69         require(wad <= _balances[msg.sender]);  
70         _approvals[msg.sender][guy] = wad;  
71         emit Approval(msg.sender, guy, wad);  
72         return true;
```



=>Here you are checking the wad value is above the 0.but in solidity that is not possible to track negative value so remove this line. You are checking the balance of user that's sufficient validations.

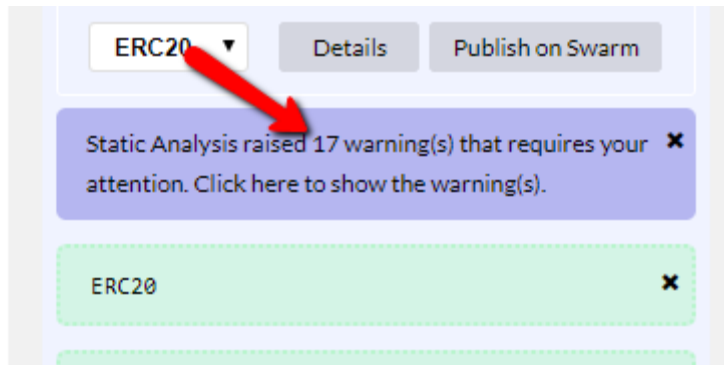
7.4 SafeMath.sol

=> No Low vulnerabilities found

Summary of the Audit

Overall the code is well commented, and performs good data validations.

The compiler also displayed 17 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.

The warnings in the orange division should be fixed as it is good practice to keep the code clean.

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

- **Note:** You are using constant for display value but it's synonym of view. So it's better to use view instead of constant.

<https://ethereum.stackexchange.com/questions/25200/solidity-what-is-the-difference-between-view-and-constant>