SMART CONTRACT AUDIT REPORT

For

5sol (Order #FO61266436F05)

Prepared By: Kishan patel **Prepared For**: Stormyset

Prepared on: 19/10/2019

Table of Content

• 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.

• Overview of the audit

The project has 1 file 5soul.txt. It contains approx 861 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

• 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.

- ## Over and underflows

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, but all the functions have strong validations, which prevented this attack.

- ## 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.

- ## 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.

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.

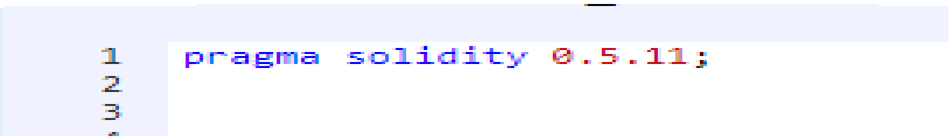
Use of “require” function in this smart contract mitigated this vulnerability.

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.

Good things in smart contract

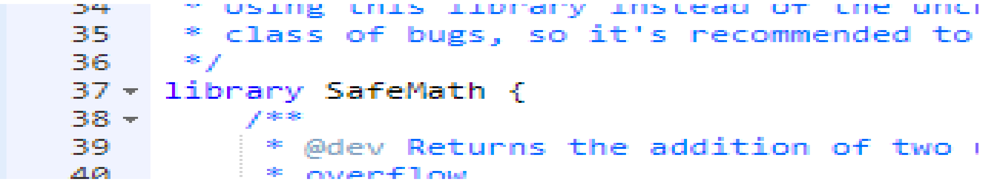
Contract version is stable:-



```
pragma solidity 0.5.11;
```

- Here you are using a stable version of solidity smart contract.But, It is a old version. Please try to move on to the latest(0.5.12) version so you can get good security benefits.

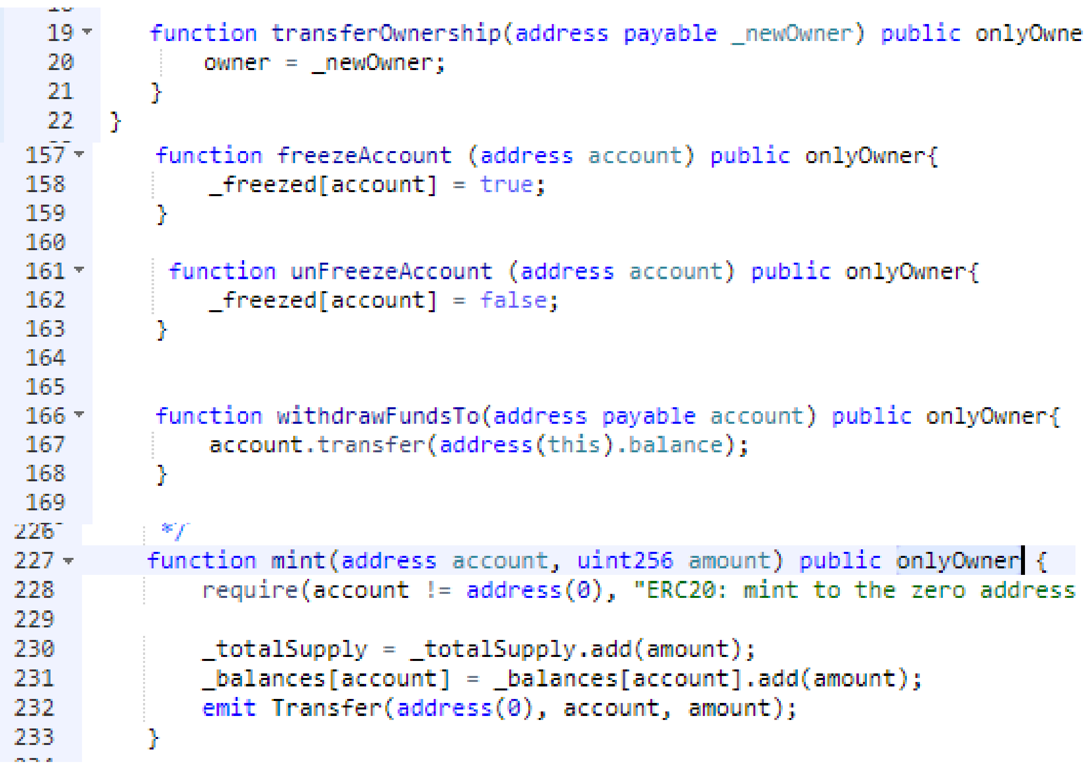
SafeMath library:-



```
 * using this library instead of the unc
 * class of bugs, so it's recommended to
 */
library SafeMath {
    /**
     * @dev Returns the addition of two
     * overflow
```

- You are using SafeMath library it is good thing. This will protect you from underflow and overflow attacks.

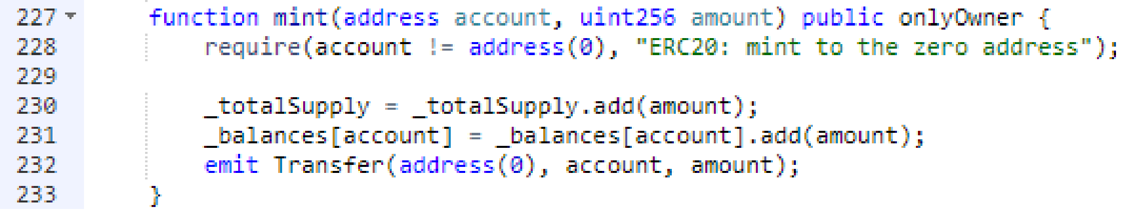
- ## Owner can call Functions:-



```
    function transferOwnership(address payable _newOwner) public onlyOwner {
        owner = _newOwner;
    }
}
     function freezeAccount (address account) public onlyOwner{
         _freezed[account] = true;
     }

      function unFreezeAccount (address account) public onlyOwner{
         _freezed[account] = false;
     }


     function withdrawFundsTo(address payable account) public onlyOwner{
         account.transfer(address(this).balance);
     }

     */
    function mint(address account, uint256 amount) public onlyOwner {
        require(account != address(0), "ERC20: mint to the zero address

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

- These five functions (transferOwnership, freezeAccount, unFreezeAccount, withdrawFundsTo, mint) are only called by the owner's contract.

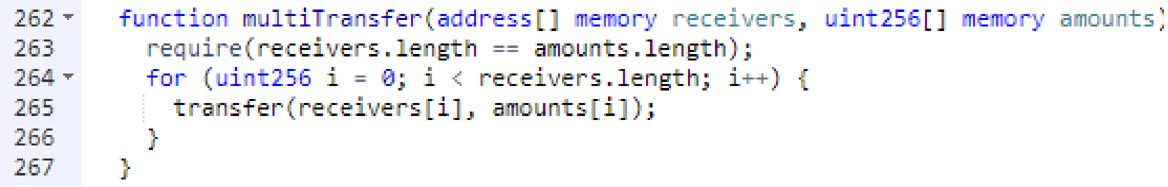
- ## mint Function:-



```
    function mint(address account, uint256 amount) public onlyOwner {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }
```

- Here you are checking "account" address that is a good thing.

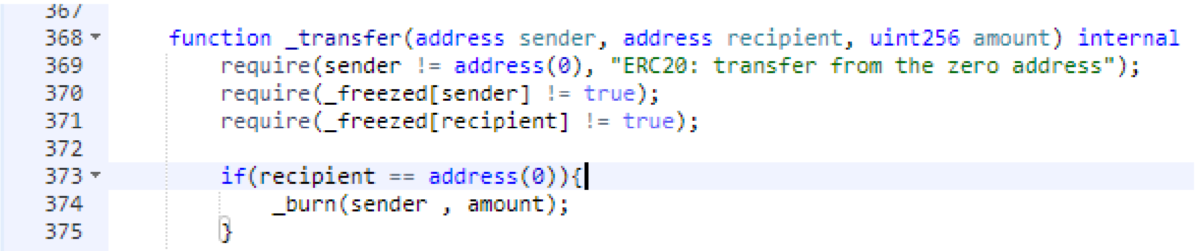
- ## multiTransfer Function:-



```
  function multiTransfer(address[] memory receivers, uint256[] memory amounts)
    require(receivers.length == amounts.length);
    for (uint256 i = 0; i < receivers.length; i++) {
      transfer(receivers[i], amounts[i]);
    }
  }
```

- Here you are seeing the number of receiver address and number of amounts is same. if it is not the same then this function will not run.

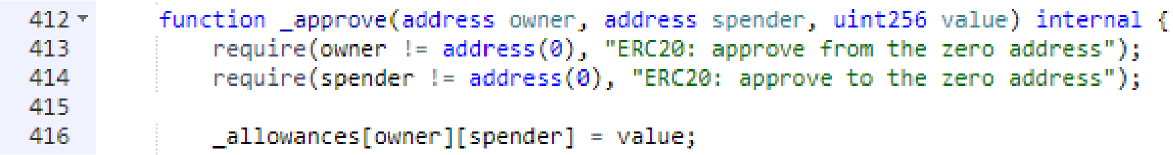
_transfer Function:-



```
function _transfer(address sender, address recipient, uint256 amount) internal
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(_freezed[sender] != true);
    require(_freezed[recipient] != true);

    if(recipient == address(0)){
        _burn(sender , amount);
    }
```

- Here you are checking the sender address is secured. Also all the senders and recipients are not frozen. If the recipient addresses is not secured then the token will be burn.

_approve Function:-

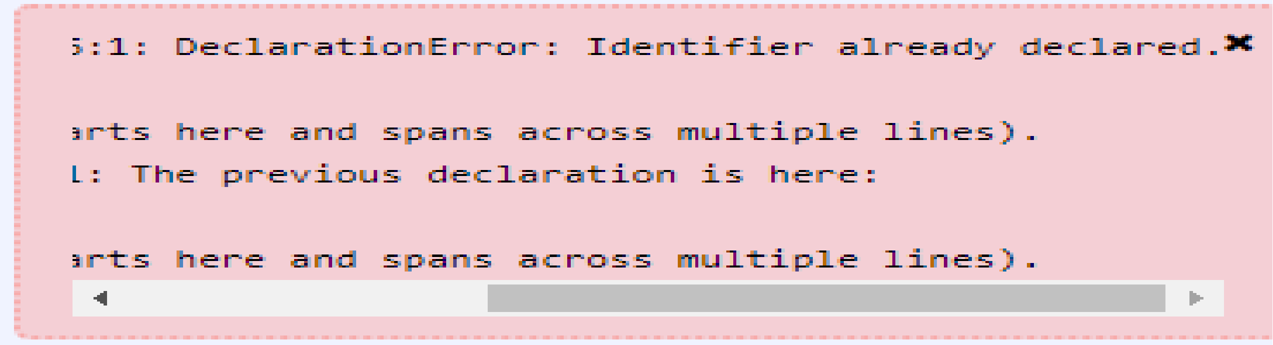


```
function _approve(address owner, address spender, uint256 value) internal {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = value;
```

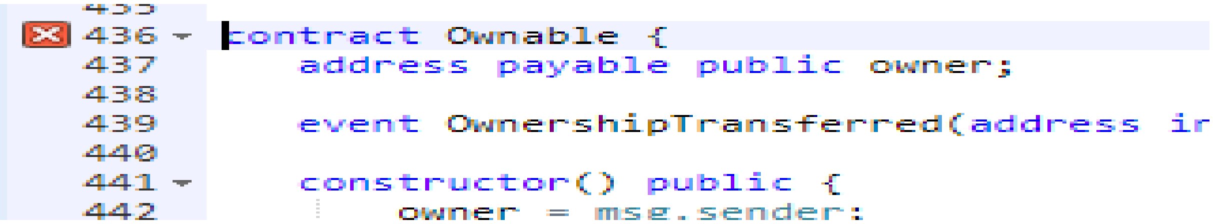
- Here you are checking the owner and spender addresses are well or not this is good thing.

Critical vulnerabilities found in the contract

Identifier already declared :-



```
5:1: DeclarationError: Identifier already declared.

arts here and spans across multiple lines).
1: The previous declaration is here:

arts here and spans across multiple lines).
```



```
436  contract Ownable {
437      address payable public owner;
438
439      event OwnershipTransferred(address in
440
441      constructor() public {
442          owner = msg.sender;
```

- I found that the real code of the contract is from 1 to 431 lines. After the line number 432 to 861 the code is repeated. If you have any strong reason for that then let me know. Otherwise, this will not allow you to publish code on ethereum.

- **Solution:-**
- Remove the all coding from line number 432 to line number 861.

• Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

• Low severity vulnerabilities found

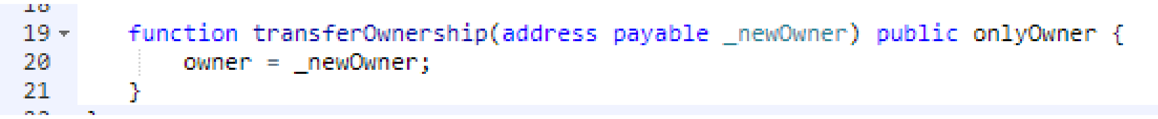
• 7.1: Short address attack

=> This is not a big issue in the solidity, because nowadays security is increased in a new solidity version. But it is good practice to check for the short addresses.

=> After Updating the version of solidity it's not mandatory.

=> In some functions you are not checking the value of address parameter

- **function:- transferOwnership('_newOwner')**

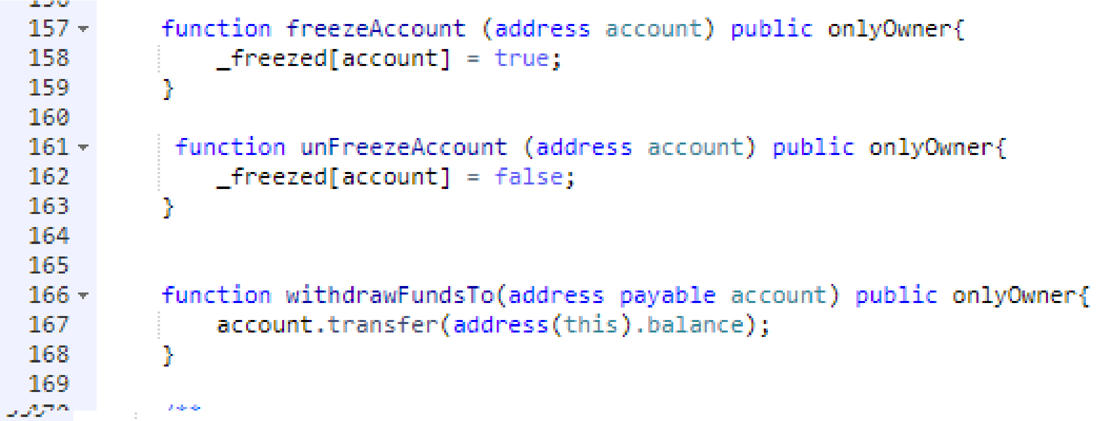


```
    function transferOwnership(address payable _newOwner) public onlyOwner {
        owner = _newOwner;
    }
```

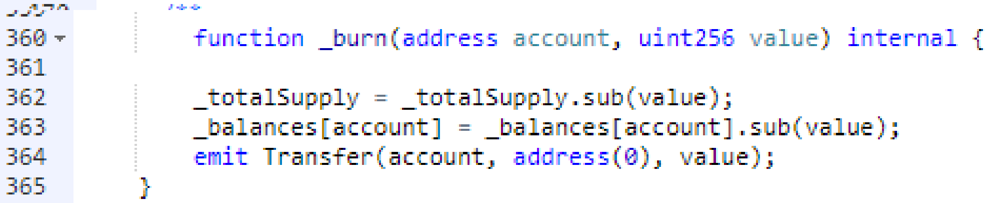
- it's necessary to check the address value of "_newOwner". Because here you are passing whatever variable comes in "_newOwner" address from outside. Also, I don't find any strong reason to use the "address payable public owner;".

- **Solution:-**
- require(_newOwner!=address(0));

- **function:- freezeAccount, unFreezeAccount, withdrawFundsTo,_burn('account')**



```
    function freezeAccount (address account) public onlyOwner{
        _freezed[account] = true;
    }

     function unFreezeAccount (address account) public onlyOwner{
        _freezed[account] = false;
    }


    function withdrawFundsTo(address payable account) public onlyOwner{
        account.transfer(address(this).balance);
    }

```



```
    /**
      function _burn(address account, uint256 value) internal {

      _totalSupply = _totalSupply.sub(value);
      _balances[account] = _balances[account].sub(value);
      emit Transfer(account, address(0), value);
    }
```

- Here you forgot to check the address value of "account". Because here you are passing whatever variable comes in "account" address from outside.

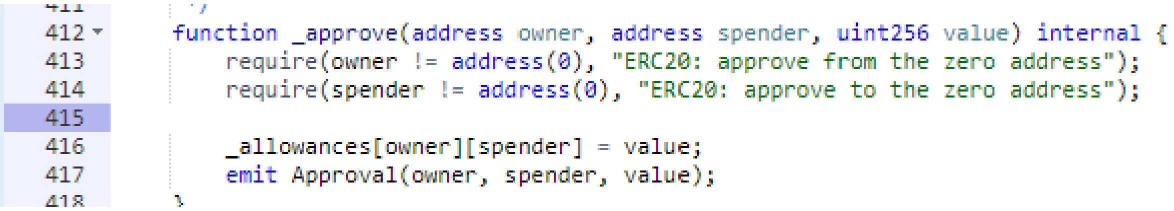
- **Solution:-**
- require(account !=address(0));

- **7.2: vulnerabilities in _approve function.**

=> This is not a big issue and your contract is not susceptible to any risks because you are checking balance and allowance in every function.

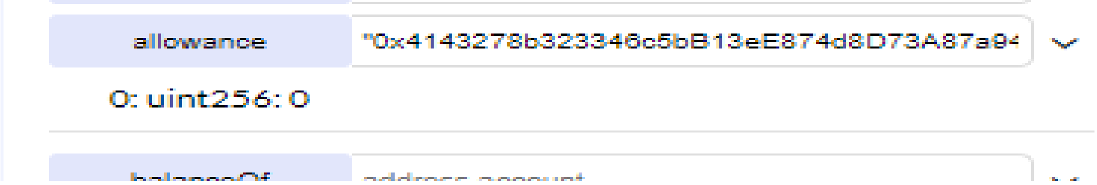
=>But it is good practice to check the balance in approve function.

=>So here, negative value also gets accepted in this function for allowance. So the allowance of any user goes wrong.



```
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

- **Underflow and overflow attack**
    - Allowance before approve

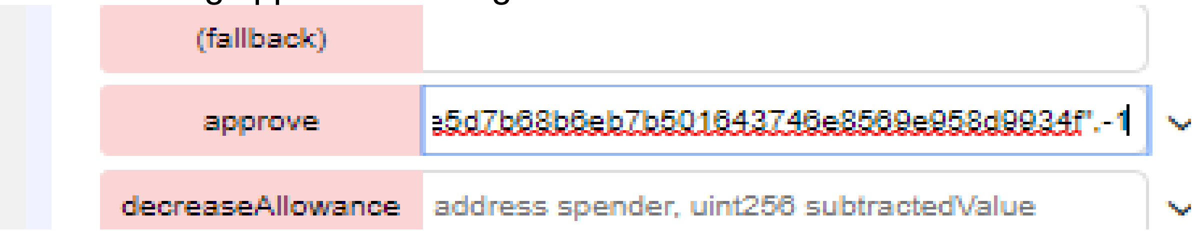


"0x4143278b323346c5bB13eE874d8D73A87a94

0: uint256: 0

address account

- Calling approve with negative value.

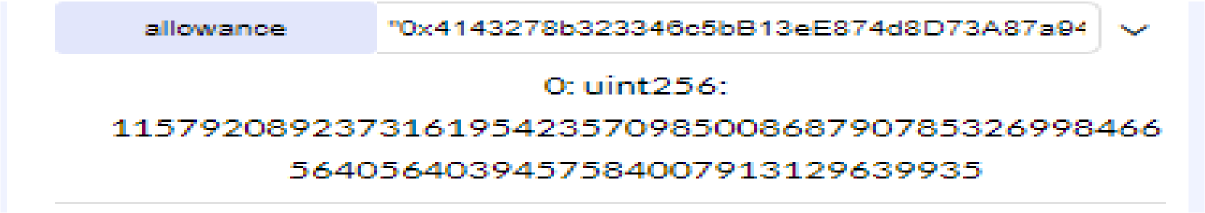


a5d7b68b6eb7b501643746e8569e958d9934f",-1

address spender, uint256 subtractedValue

- https://ropsten.etherscan.io/tx/0xdc7e994a01ad075c470ec48e6bcbb902c61151fdf4f1d1a82c5006352ab2c3fb

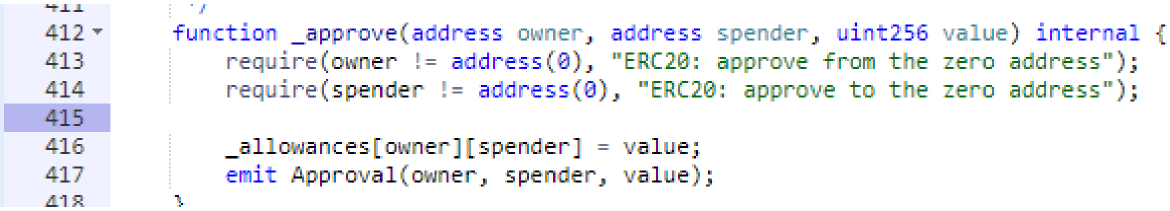
- Allowance after approve



"0x4143278b323346c5bB13eE874d8D73A87a94

0: uint256: 115792089237316195423570985008687907853269984665640564039457584007913129639935

- **Solution:-**



```
  */
function _approve(address owner, address spender, uint256 value) internal {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = value;
    emit Approval(owner, spender, value);
}
```

- In approve function you have to put one condition.

  require(_value <= _balances[owner]);

- By this way, user only approves the amount which he has in the balance.

=> This problem is also effect on decreaseAllowance, increaseAllowance and approve functions.

=> This error also available in decreaseAllowance, increaseAllowance and approve.

7.3: Not necessary function in SafeMath library:-

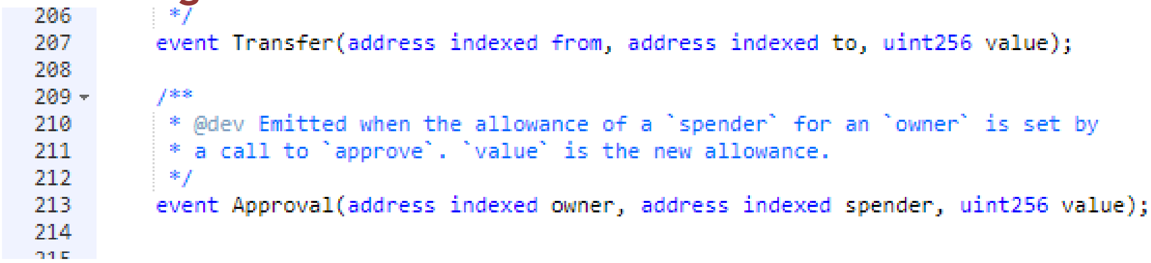
=> As, I showed there is some functions which is not used from the safeMath library.

=>I showed that mul, div and mod function is not used in the contract. So, I am giving suggestions to remove those functions from the library for good code.

=>This is not a huge problem. If your owner is allowed to put a full library then it is okay.

=> **Solution :-**
=> Remove the mul, div and mod functions from the SafeMath library.

7.4: Not good structured code:-



```
 */
event Transfer(address indexed from, address indexed to, uint256 value);

/**
 * @dev Emitted when the allowance of a `spender` for an `owner` is set by
 * a call to `approve`. `value` is the new allowance.
 */
event Approval(address indexed owner, address indexed spender, uint256 value);
```

=> I found that the code is not well structured.

=>The events should be on the top but in your code, I found in the middle of the code.

=>_transfer function has no comments.

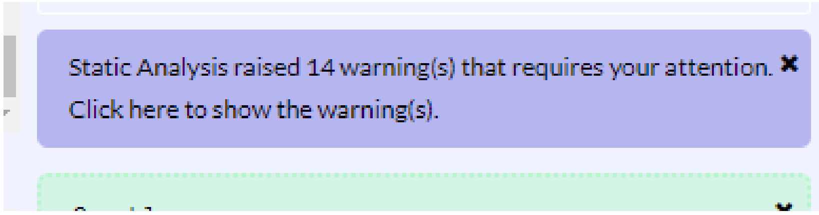
=> Also I found too many comments which are not required. But if the owner allowing you then this is okay.

=> **Solution :-**
=> This events should be come after or before the constructor function.

Summary of the Audit

Overall the code is not well. The compiler also displayed 14

warnings and 3 errors.



Static Analysis raised 14 warning(s) that requires your attention. ✖
Click here to show the warning(s).

Now, we checked those warnings are due to their static analysis, which includes 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 of while calling the smart contract functions.

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

The errors I mentioned in the critical vulnerabilities (**Identifier already declared**).

- **Note:** Please focus on approve function, version of contract because in the new version you will get new features, address value checking and structure your code.