Smart Contract Code Review and Security Analysis Report

Customer: SovereignWallet
Date: July 12, 18

Document:

Name:	Smart Contract Code Review and Security Analysis Report for SovereignWallet
Date:	12.07.2018

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by SovereignWallet (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between June 28th, 2018 - July 12th, 2018.

Scope

The scope of the project is TokenVesting smart contract, which can be found on github by link below:

https://github.com/SovereignWallet-Network/MUI-Smart-Contract/tree/master/contracts

Commit: f2e93de6ea38d754552949c0568116f6115356de

Full list of reviewed contracts is: ACB.sol, Airdrop.sol, BulkTransferable.sol, Claimable.sol, Depositable.sol, Destructible.sol, ERC20.sol, MuiToken.sol, Ownable.sol, Pausable.sol, PausableToken.sol, PermissionGroups.sol, PhaseBasedACB.sol, RBAC.sol, Roles.sol, SafeMath.sol, StandardToken.sol, Withdrawable.sol.

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

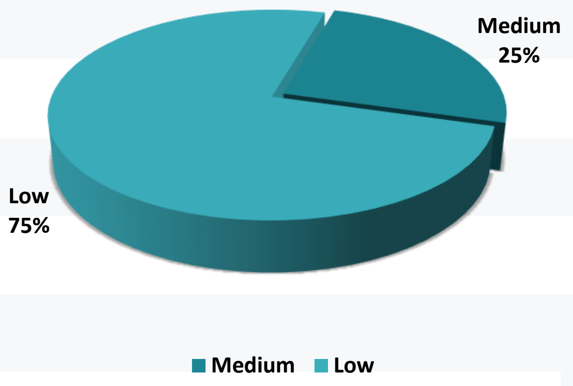
According to the assessment, smart contract is well secured and only a few minor fixes are required.

Automated tools analysis was done in previous project so this report doesn't contain automated tools findings.

Our team performed analysis of code functionality and manual audit. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 1 medium and 3 low severity vulnerabilities. They can't have major security or functional impact; however, it is recommended to fix them.

Graph 1. Vulnerabilities distribution



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Info	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

MuiToken contract overview

MuiToken contract is inherited from PausableToken and Claimable, these contracts (including Ownable, StandardToken, Pausable, ERC20, SafeMath) are copied from Zeppelin repository (sometimes with minor changes without any security impact) and they are commonly considered as secure. MuiToken contract parameters are:

- name is MUI Token
- symbol is MUI
- decimals is 6
- TOKEN_SUPPLY to 1000000000
- INITIAL_SUPPLY to TOKEN_SUPPLY * (10 ** uint256(decimals))

MuiToken contract constructor sets:

- totalSupply_ to INITIAL_SUPPLY
- balances[supplier] to INITIAL_SUPPLY

PermissionGroups contract overview

PermissionGroups contract is inherited from RBAC (uses Roles). RBAC and Roles are copied from Zeppelin repository (sometimes with minor changes without any security impact) and they are commonly considered as secure. PermissionGroups contract defines 3 roles:

- ROLE_ADMIN is admin (number of admins is limited to 5)
- ROLE_OPERATOR is operator (number of operators is limited to 50)
- ROLE_BLACK_LISTED is blacklisted

PermissionGroups contract defines next modifiers and functions:

- modifier onlyAdmin – checks whether msg.sender is admin
- modifier onlyOperator – checks whether msg.sender is operator
- modifier onlyWhiteListed – checks whether msg.sender is not blacklisted
- constructor function sets msg.sender to be admin
- function addAdmin – adds new admin
- function removeAdmin – removes admin
- function isAdmin – checks whether addr is admin
- function addOperator – adds new operator
- function removeOperator – removes operator
- function isOperator – checks whether addr is operator
- function addToBlackList – adds new address to blacklist
- function removeFromBlackList – removes address from blacklist
- function isBlackListed – checks whether addr is in blacklist

ACB, Airdrop, PhaseBasedACB contracts overview

ACB, Airdrop contracts are inherited from Withdrawable, Depositable, Destructible. Withdrawable contract is modified Zeppelin contract allowing to withdraw ether or tokens sent to the contract. Depositable contract is modified Zeppelin contract allowing to send ether or tokens to the contract. Destructible contract is modified Zeppelin contract allowing to destruct the contract and send remaining tokens to owner or recipient.

Airdrop contract defines airdrop strategy for MUI token. Airdrop contract constructor sets tokenAddress to token parameter. Airdrop fallback function reverts everything. Airdrop contract defines next functions:

- function setIncentives – sets incentives hash root; has onlyAdmin modifier
- function isClaimed – checks whether the incentive with index is claimed
- function claim – claims incentive and transfer claimed amount to claimer
- function checkMerkleProof – checks whether given inputs check merkle tree
- function markClaimed – marks the incentive with the given index claimed

ACB contract defines “Algorithmic Central Bank” with next variables and constants:

- FEE_RATE_DENOMINATOR is 1000000
- buyPriceACB is not set by default
- sellPriceACB is not set by default
- buySupplyACB is set to 0 by default
- sellSupplyACB is set to 0 by default
- feeRateACB is set to 0 by default
- minSellAmountACB is set to 0 by default
- maxBuyAmountACB is set to ~uint256(0) by default

ACB contract constructor sets tokenAddress to token parameter and calls setPrices with initialBuyPrice and initialSellPrice parameters. ACB fallback function reverts everything. ACB contract defines next functions:

- function setPrices – sets prices in ACB; has onlyAdmin modifier
- function setFeeRate – sets feeRateACB; has onlyAdmin modifier
- function setAvailableSupplies – calls setSupplies with special conditions; has onlyAdmin modifier
- function setBuySellCaps – sets maxBuyAmountACB and minSellAmountACB; has onlyAdmin modifier
- function buyFromACB – allows to buy tokens for client
- function sellToACB – allows to sell specified number of tokens for client
- function feeCollector – sends value without fee to specified address
- function setSupplies – sets sellSupplyACB and buySupplyACB
- function calculateCost – calculates costs for given number of tokens

PhaseBasedACB contract defines ACB with phases special conditions and next variables and constants:

- phaseStartTime is set to 0 by default
- phaseEndTime is set to 0 by default
- phaseIndex is set to 0 by default
- initialBuySupplyACB is set to 0 by default
- initialSellSupplyACB is set to 0 by default

PhaseBasedACB contract constructor sets tokenAddress to token parameter, calls setPrices with initialBuyPrice and initialSellPrice parameters and calls ACB constructor. PhaseBasedACB contract defines next functions and modifiers:

- whenPhaseActive modifier – checks whether phase is active
- whenPhaseDeactive modifier – checks whether phase is not active
- function isPhaseActive – checks whether phase active or not
- function setSalePhase – calls setPhasePeriod with startTime, endTime parameters; calls super.setPrices with buyPriceACB, sellPriceACB; calls super.setAvailableSupplies with buySupplyACB, sellSupplyACB parameters; sets initialBuySupplyACB and initialSellSupplyACB; increments phaseIndex; has onlyAdmin and whenPhaseDeactive modifiers
- function setSalePhase – sets phaseStartTime and phaseEndTime; has onlyAdmin modifier
- buyFromACB – calls super.buyFromACB; has whenPhaseActive and onlyWhiteListed modifiers
- sellToACB– calls super. sellToACB; has whenPhaseActive and onlyWhiteListed modifiers
- buyBack – calls super. buyFromACB; has onlyAdmin modifier

Audit overview

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

1. markClaimed function defines new redeemed mapping when called and it marks incentive claimed in the redeemed mapping. However, vRedeemed[version] is not changed so isClaimed won't show incentive as claimed. Consider overwriting values in vRedeemed instead of redeemed (see Appendix A pic 1 for evidence).

Low

2. Compiler version is not locked. Consider locking the compiler version with latest one (see Appendix A pic 2 for evidence).

```
pragma solidity ^0.4.23; // bad: compiles w 0.4.23 and above
pragma solidity 0.4.23; // good: compiles w 0.4.23 only
```

3. Function buyFromACB calculates token amount accordingly to next formula:

```
uint256 tokenAmount = sellPriceACB.mul(msg.value).div(1 ether);
```

   However, integer div is used, what means if sellPriceACB = 1 and user sends 1.99 ether to buyFromACB, user receives 1 token for 1 ether and 0.99 ether is send to contract for nothing (see Appendix A pic 3 for evidence). Consider using additional require statement, for example, require(sellPriceACB.mul(msg.value) % (1 ether) < someValue).
4. checkMerkleProof and markClaimed descriptions are poor – they are copied from claim and isClaimed functions descriptions. This can lead to functions misuse. Consider changing the descriptions to correct ones.

Lowest / Code style / Info

No lowest severity vulnerabilities, code style issues and informational statements were found.

Conclusion

During the audit the contracts were manually reviewed. As-is description was described.

Audit team have not found any significant security issues during the audit.

Overall quality of reviewed contract is high and only some minor fixes are required. Contracts contain only medium to low issues, which can’t have serious impact on contracts security.

Disclaimers

Disclaimer

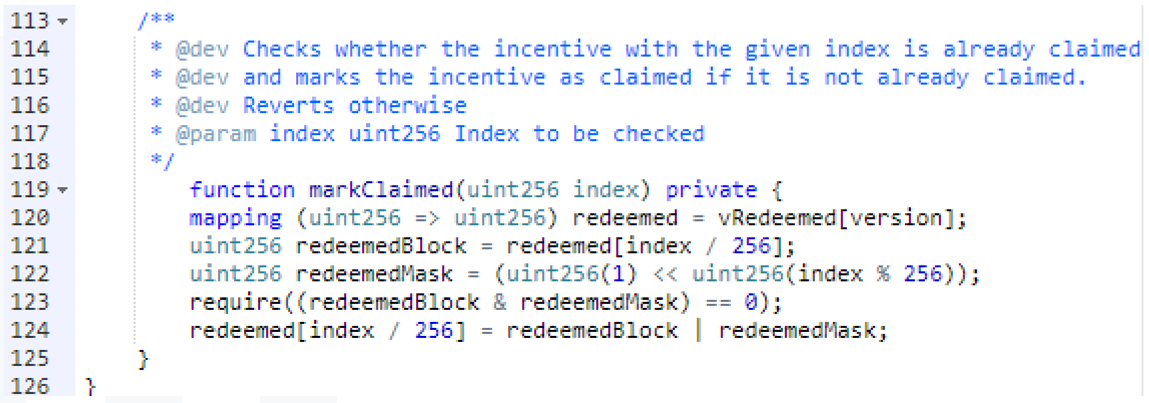
The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts.

Technical Disclaimer

Smart contract build on the top of Ethereum blockchain means that a lot of features could be covered by tests, but Turing completeness of Solidity programming language realization leaves some space for unexpected runtime exceptions.

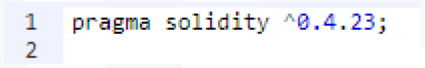
Appendix A. Evidences

Pic 1. markClaimed doesn't change vRedeemed:



```
    /**
     * @dev Checks whether the incentive with the given index is already claimed or not
     * @dev and marks the incentive as claimed if it is not already claimed.
     * @dev Reverts otherwise
     * @param index uint256 Index to be checked
     */
        function markClaimed(uint256 index) private {
        mapping (uint256 => uint256) redeemed = vRedeemed[version];
        uint256 redeemedBlock = redeemed[index / 256];
        uint256 redeemedMask = (uint256(1) << uint256(index % 256));
        require((redeemedBlock & redeemedMask) == 0);
        redeemed[index / 256] = redeemedBlock | redeemedMask;
    }
}
```

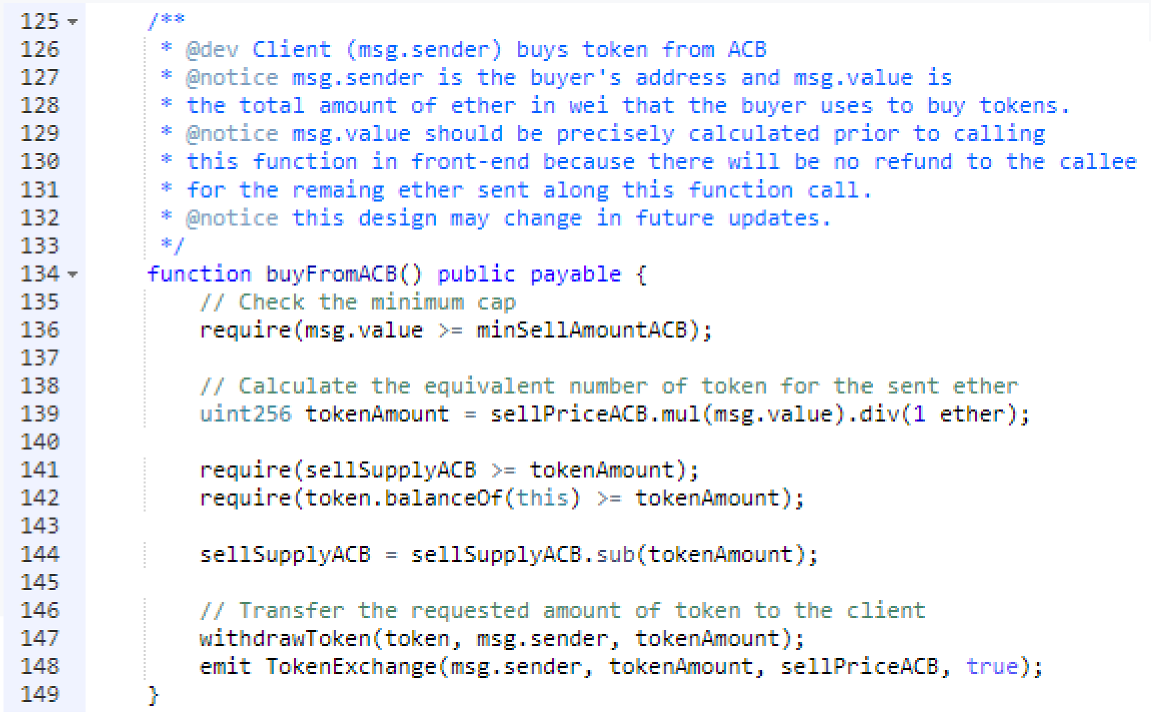
Pic 2. Compiler version not locked:



```
pragma solidity ^0.4.23;

```

Pic 3. Extra funds can be sent to buyFromACB:



```
    /**
     * @dev Client (msg.sender) buys token from ACB
     * @notice msg.sender is the buyer's address and msg.value is
     * the total amount of ether in wei that the buyer uses to buy tokens.
     * @notice msg.value should be precisely calculated prior to calling
     * this function in front-end because there will be no refund to the callee
     * for the remaing ether sent along this function call.
     * @notice this design may change in future updates.
     */
    function buyFromACB() public payable {
        // Check the minimum cap
        require(msg.value >= minSellAmountACB);

        // Calculate the equivalent number of token for the sent ether
        uint256 tokenAmount = sellPriceACB.mul(msg.value).div(1 ether);

        require(sellSupplyACB >= tokenAmount);
        require(token.balanceOf(this) >= tokenAmount);

        sellSupplyACB = sellSupplyACB.sub(tokenAmount);

        // Transfer the requested amount of token to the client
        withdrawToken(token, msg.sender, tokenAmount);
        emit TokenExchange(msg.sender, tokenAmount, sellPriceACB, true);
    }
```