Smart Contract Code Review and Security Analysis Report

Customer: Essentia ltd.
Date: June 8, 18

This document contains confidential information about IT systems and intellectual properties of the customer, as well as information about potential vulnerabilities and methods of their exploitation.
This confidential information is for internal use by the customer only and shall not be disclosed to third parties.

Document:

Name:	Smart Contract Code Review and Security Analysis Report for Essentia ltd.
Date:	08.06.2018

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Essentia ltd. (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 5th, 2018 - June 8th, 2018.

Scope

The scope of the project is Essentia smart contract, which can be found by link below: https://etherscan.io/address/0xfc05987bd2be489accf0f509e44b0145d68240f7#code

This smart contract consists of:

- contract Ownable
- contract ESSENTIA_ERC20
- contract ESSENTIA
- interface tokenRecipient
- library SafeMath

We have scanned these smart contracts 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

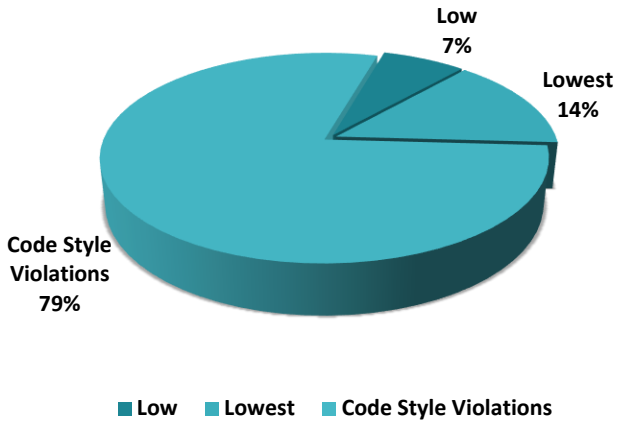
Executive Summary

According to the assessment, Customer`s smart contract is generally secure with some possibilities for improvement.

Our team performed code review, manual audit and automated checks with solc, Mythrill and remix IDE (see Appendix B pic 1-4). General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 3 security issues and 11 cases of violation of the code style guide, however, no critical, high or medium severity vulnerabilities were found.

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

Ownable contract describes basic access control with owner role which is an actual address what control the smart contract.

ESSENTIA_ERC20 contract describes custom ERC20 token with next parameters:

1. standard - ESSENTIA erc20 and Genesis
2. decimals – 18
3. name – ESSENTIA
4. symbol - ESS
5. Total supply of token is only declared

ESSENTIA contract is ESSENTIA_ERC20 contract, which defines 2 addresses A and B, that are hardcoded and hold all the tokens after constructor called. The balances of accounts are:

- address A – 614359681000000000000000000 tokens
- address B – 114095369200000000000000000 tokens

Total supply is 175531337300000000000000000 ESS tokens.

The functionality of ESSENTIA contract is grouped by access – functions with public access and function with only owner access (address owner can only run functions with onlyOwner modifier).

Public functions are:

- balanceOf – view that returns balance of the account
- transfer – allows to transfer token to some account
- transferFrom – allows to transfer token to some account from some other account if msg.sender has an allowance
- approve – changes the allowance for account to some value
- allowance – view that returns allowance for pair of owner and spender accounts
- increaseApproval – allows to increase the allowance for some account
- decreaseApproval – allows to decrease the allowance for some account
- approveAndCall – allows to approve and then communicate the approved contract

onlyOwner function is only transferOwnership to transfer ownership of the contract.

tokenRecipient interface is declaring the required functions and events to meet the ERC20 standard to be able to send tokens to the contract.

SafeMath library overwrites math operations with safety checks that throw on error.

Audit overview

Critical

No critical severity vulnerabilities were found.

High

No critical severity vulnerabilities were found.

Medium

No medium severity vulnerabilities were found.

Low

Overall

1. Compiler version is not locked. Consider locking the compiler version with latest one (see Appendix A pic 1 for evidence).

```
pragma solidity ^0.4.24; // bad: compiles w 0.4.24 and above
pragma solidity 0.4.24; // good: compiles w 0.4.24 only
```

Lowest / Code style / Info

Contract ESSENTIA

2. A and B addresses are hardcoded and their naming is unclear one (see Appendix A pic 2 for evidence). Consider adding functionality for management of these addresses – setting, changing etc. Consider also changing the name of accounts and adding a description about them in comments.
3. Contract contains some redundant code on lines 207 and 208 (see Appendix A pic 3 for evidence). Consider rewriting the code as described below.

```
balances[A]=balances[A].add(614359681*(uint256(10)**decimals));
balances[B]=balances[B].add(1140953692*(uint256(10)**decimals));
                                    TO
balances[A] = 614359681 * (uint256(10) ** decimals);
balances[B] = 1140953692 * (uint256(10) ** decimals);
```

Code style issues

6. Library SafeMath is defined on lines 24-51 without any comments or descriptions. It was copied from OpenZeppelin SafeMath – library that is considered secure by community. Consider adding comments about the origin of the library or importing it from OpenZeppelin repository.
7. Contract Ownable is defined on lines 55-75 without any comments or descriptions. It was copied from older version of OpenZeppelin Ownable contract – library that is considered secure by community. Consider adding comments about the origin of the contract or importing it from OpenZeppelin repository.
8. Contract ESSENTIA_ERC20 is defined on lines 88-182 without any comments or descriptions for functions. All the functions were copied from OpenZeppelin contracts that are considered secure by community. Consider adding comments about the origin of the function and their functionality, for instance, like described below

```
/**
  * @dev Transfer tokens from one address to another
  * @param _from address The address which you want to send tokens from
  * @param _to address The address which you want to transfer to
  * @param _value uint256 the amount of tokens to be transferred
  */
 function transferFrom(
   address _from,
   address _to,
   uint256 _value
 )
   public
   returns (bool)
 {
   require(_to != address(0));
   require(_value <= balances[_from]);
   require(_value <= allowed[_from][msg.sender]);

   balances[_from] = balances[_from].sub(_value);
   balances[_to] = balances[_to].add(_value);
   allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
   emit Transfer(_from, _to, _value);
   return true;
 }
```

9. tokenRecipient interface doesn't have any description in comments. Consider adding description of interface functionality in comments.
10. ESSENTIA contract doesn't have any description in comments. Consider adding description of contract in comments.

11. Constant names are not in SNAKE_CASE (see lines 101, 102). Consider using SNAKE_CASE for constants.
12. Contract names are not in CamelCase (see lines 88, 186). Consider using CamelCase for constants.
13. No spaces near math expressions (see lines 207, 208, 210). Consider adding spaces near math operators like “=”, “+”, “**” etc.
14. 4 extra spaces on lines 195, 196. Consider removing extra spaces.
15. Semicolon has spaces before itself on line 187. Consider removing space before semicolon.
16. Definition is not surrounded with two blank line indents on lines 24, 55, 88, 186, 192.

Conclusion

During the audit all the contracts were manually reviewed and analyzed with static analysis tools. As-is description was described.

Audit report contains all found security vulnerabilities and code style guide violations in the reviewed code.

Overall quality of reviewed contracts is good and no global changes are needed. However, 3 low to lowest security issues were found that are needed to be fixed.

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. Compiler version is not locked:

Essentia.sol

```
1  pragma solidity ^0.4.24;
2
```

Pic 2. Unclear address naming and hardcoded addresses:

```
contract ESSENTIA is ESSENTIA_ERC20 {


        address public A;
        address public B;


    constructor (

        ) public {

        A = 0x564a1D21886ADF1F46FF9D867CE8827C5Ec1B388;
        B = 0xB33532656433f4Eca3782F6B20298d1424d1F2CF;


        balances[A]=balances[A].add(614359681*(uint256(10)**decimals));
        balances[B]=balances[B].add(1140953692*(uint256(10)**decimals));

        totalSupply=balances[A]+balances[B];


    }


}
```

Pic 3. Redundant code on lines 207-208:

```
207             balances[A]=balances[A].add(614359681*(uint256(10)**decimals));
208             balances[B]=balances[B].add(1140953692*(uint256(10)**decimals));
```

Appendix B. Automated reviews

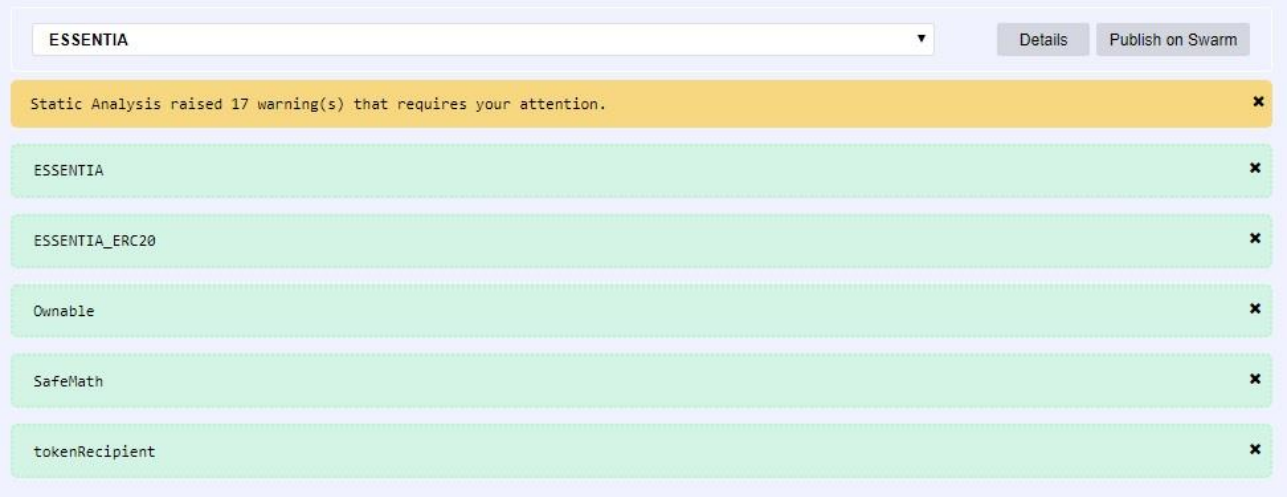
Pic 1. Solc automated report

```
pvl@ubuntu:~/solidity/projects/Essentia_new$ solc -o . --bin --abi --overwrite Essentia.sol
pvl@ubuntu:~/solidity/projects/Essentia_new$
```

Pic 2. Mythrill automated report

```
pvl@ubuntu:~/solidity/projects/Essentia_new$ myth -x Essentia.sol
The analysis was completed successfully. No issues were detected.
pvl@ubuntu:~/solidity/projects/Essentia_new$
```

Pic 3. Remix IDE automated report part 1



Pic 4. Remix IDE automated report part 2