HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Statera
Date: June 18, 2020
Platform: Ethereum
Language: Solidity

Document

Name	Smart Contract Code Review and Security Analysis Report for Statera
Platform	Ethereum / Solidity
Repository	https://github.com/StateraProject/statera-token/tree/master/contracts
Commit	611d2b5c2675d824c233400f732d172d3cc14738
Branch	master
Date	18.06.2020

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Statera (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 17th, 2020 - June 18th, 2020.

Scope

The scope of the project are smart contracts within the repository:

Repository - https://github.com/StateraProject/statera-token/tree/master/contracts

Commit - 611d2b5c2675d824c233400f732d172d3cc14738

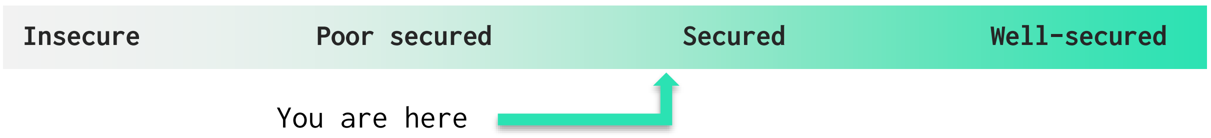
Branch - master

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
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

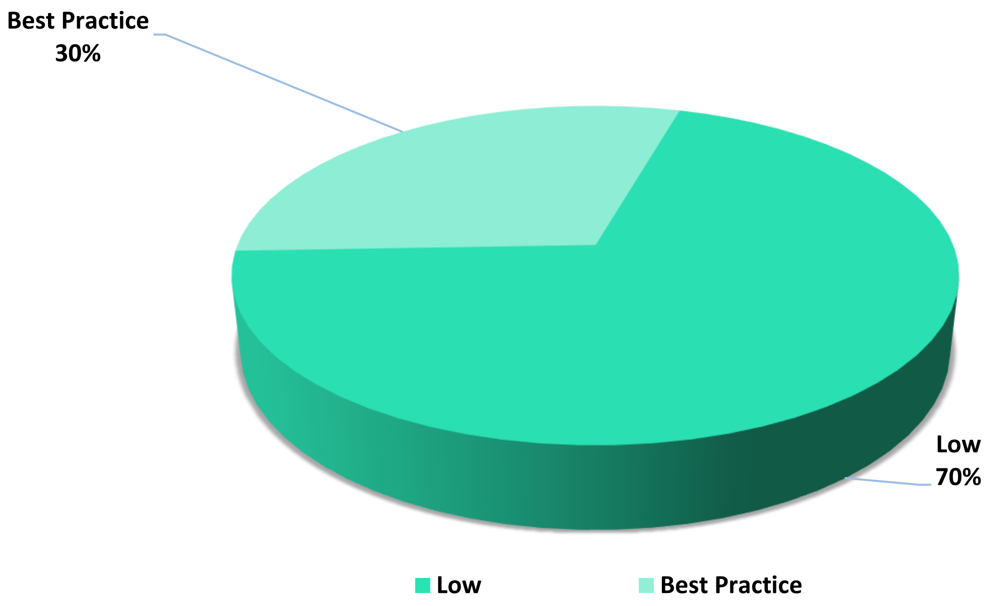
According to the assessment, Customer`s smart contracts are secured.



Our team performed analysis of code functionality, manual audit and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We found 7 low and 3 best practice issues in smart contract code.

Graph 1. The distribution of vulnerabilities.



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 / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

IERC20 is the standard ERC20 interface.

SafeMath library is standard library for math operations that prevents integer overflows and underflows.

ERC20Detailed is standard **IERC20** implementation with extra token parameters.

Contract **Statera** is **ERC20Detailed**.

Contract **Statera** defines following public parameters:

- private mapping (**address** => **uint256**) **_balances**
- private mapping (**address** => mapping (**address** => **uint256**)) **_allowed**
- string **tokenName** is set to "Statera"

- string **tokenSymbol** is set to "STA"
- uint8 **tokenDecimals** is set to 18
- uint256 **_totalSupply** is set to 101000000000000000000000000
- uint256 **basePercent** is set to 100

Contract **Statera** has 14 functions:

- **constructor** - calls **ERC20Detailed** constructor and issues total supply to caller
- **totalSupply** - public view function that returns token total supply
- **balanceOf** - public view function that returns the balance of the account
- **allowance** - public view function that returns the allowance of the spender account for specified address
- **cut** - public view function that returns the commission to be burned for the value
- **transfer** - public function that transfers tokens from caller to receiver
- **approve** - public function that approves transfer of token for specified spender
- **transferFrom** - public function that transfers from address to specified receiver by approved caller

- **upAllowance** - public function that increases allowance for specified account
- **downAllowance** - public function that transfers from address to specified receiver by approved caller
- **_issue** - internal function that mints tokens to account
- **destroy** - external function that burns tokens
- **_destroy** - internal function that burns tokens
- **destroyFrom** - external function that burns tokens from specified address

Audit overview

Critical

No critical issues were found.

High

No high issues were found.

Medium

No medium issues were found.

Low

1. No visibility specified for tokenName, tokenSymbol, tokenDecimals and _totalSupply, thus, all of these parameters will be public. However, parameters starting with underscore(_totalSupply) should be private.

2. No solidity version is specified in the pragma. It's recommended lock the pragma with latest stable version of the solidity.

3. Constructor is payable, however, if deployer send any Ether to the contract - it'll be forever locked. It's recommended to remove payable from constructor.

4. When 0.000000000000000001 token is transferred - it's not being transferred but being burned - receiver gets 0.

5. Poor naming for following functions:

    a. upAllowance - should be increaseAllowance

    b. downAllowance - should be decreaseAllowance

c. _issue - should be _mint

d. destroy - should be burn

e. _destroy - should be _burn

f. destroyFrom - should be burnFrom

g. _allowed - should be _allowances

The issue may lead to integration issues, for example, with other smart contracts that expect upAllowance function to be called increaseAllowance.

6. The code is not fully covered with documentation. The absence of documentation may cause differences between implementation and expected behavior of smart contracts. Documentation should be presented for all smart contract’s code.

7. Code is not covered with any unit tests. It increases the risk of the unexpected flaws in the smart contracts. It is recommended to have 100% code coverage with automated tests.

Lowest / Code style / Best Practice

8. Outdated version of SafeMath is used. assert is outdated and not recommended to use within the code. It’s recommended to update SafeMath to newer version.

9. It’s no needed to store in storage tokenName, tokenSymbol and tokenDecimals that are used only once in code - in constructor.

10. Following checks can be omitted:

    a. Line 36

    b. Line 61

    c. Line 107

Conclusion

Smart contracts within the scope was manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good. Security engineers found several low issues that don’t have serious security impact.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.