SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Trava Finance
Date: September 06th, 2021

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Trava Finance.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Transfer controller
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/TravaFinance/Lending-Protocol
Commit	7d3aedab448c99691fb0eb6e7f4ed210b1f03336
Technical Documentation	NO
JS tests	NO
Timeline	09 AUGUST 2021 - 06 SEPTEMBER 2021
Changelog	17 AUGUST 2021 - INITIAL AUDIT 26 AUGUST 2021 - SECOND REVIEW 06 SEPTEMBER 2021 - THIRD REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Trava Finance (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between August 9th, 2021 - August 17th, 2021. The second code review conducted on August 26th, 2021. The third code review conducted on September 6th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/TravaFinance/Lending-Protocol
Commit:
7d3aedab448c99691fb0eb6e7f4ed210b1f03336
Technical Documentation: No
JS tests: No
Contracts:
contracts/protocol/lendingpool/ReserveInterestRateStrategy.sol
contracts/protocol/lendingpool/LendingPoolConfigurator.sol
contracts/protocol/lendingpool/LendingPool.sol
contracts/protocol/lendingpool/LendingPoolStorage.sol
contracts/protocol/lendingpool/LendingPoolCollateralManager.sol

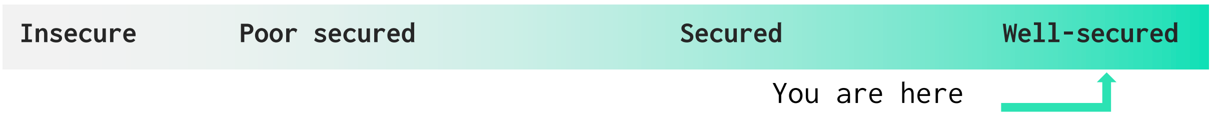
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.



Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **4** high, **4** medium and **3** low severity issues.

After the second review security engineers found **1** medium severity issue.

After the third review security engineers found **no issues** in the current scope.

Notice:

This is the repository part audit. Most of the contracts in the repository were out of the current audit scope.

Notice 2:

Possible Flash Loan is fixed in the TToken contract (contracts/protocol/tokenization/TToken.sol) which is out of the scope of the current audit.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a 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 assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

1. Unimplemented function

   Function **LendingPool.finalizeTransfer** has commented-out unimplemented logic.

   Recommendation: Please implement the logic or comment-out entire function.

   Fixed before the second review.

2. Uninitialized state variable

   State variable **LendingPoolStorage._addressesProvider** is not initialized for the contract `LendingPoolCollateralManager` but it's being called on lines *109* and *309* of the LendingPoolCollateralManager.sol

   Recommendation: Please consider initializing the state.

   Status: Not an issue.

   Cusromer's comment: *We consider the contract LendingPoolCollateralManager as a Logic Contract where we execute our code by using delegate call from LendingPool Contract. That explains why we don't initialize the state for that variable.*

3. Uninitialized state variable

   State variable **LendingPoolStorage._providerId** is not initialized for the contract `LendingPoolCollateralManager` but it's being called on lines *109* and *309* of the LendingPoolCollateralManager.sol

   Recommendation: Please consider initializing the state.

   Status: Not an issue.

   Cusromer's comment: *We consider the contract LendingPoolCollateralManager as a Logic Contract where we execute our code by using delegate call from LendingPool Contract. That explains why we don't initialize the state for that variable.*

4. Uninitialized state variable

   State variable **LendingPoolStorage.reservesCount** is not initialized for the contract `LendingPoolCollateralManager` but it's being called on line 108 of the LendingPoolCollateralManager.sol

   Recommendation: Please consider initializing the state.

   Status: Not an issue.

Cusromer's comment: *We consider contract LendingPoolCollateralManager as a Logic Contract where we execute our code by using delegate call from LendingPool Contract. That explains why we don't initialize the state for that variable.*

🟩 🟩 Medium

1. Unused import

   Contract **LendingPoolConfigurator** imports *hardhat/console.sol* library which is not used in the code.

   Recommendation: Please consider removing or commenting-out the import.

   Fixed before the second review.

2. Possible FlashLoan attack in the **LendingPool** contract

   IF both of **TToken** and it's **underlyingAsset** are traded on multiple DEXes and have different rates there could be a FlashLoan attack.

   Attacker could FlashLoan a **TToken**, and process **withdraw**, which will give then the exact same amount of the **underlyingAsset**, and then calling **deposit**, they will receive the same amount of **TToken.**

   Having the 1-to-1 "*exchange*" on the LendingPool and different rates on DEXes, attacker could "*play*" with the market many times in one transaction until receive enough rewards.

   The same would work to borrow method. So anyone could borrow with not having assets but FlashLoaning them.

   Recommendation: Please consider some restrictions like minting/returning asset in the next block only.

   Fixed before the third review by adding token transfer restrictions for transferring in the same block where minting occurred.

3. Not checked contract address

   Calling **LendingPool.initReserve** with providing incorrect address for **asset** or **tTokenAddress** would make this "*pair*" unworkable because neither **LendingPool.initReserve** nor **ReserveLogic.init** check the contracts to implement the given interfaces (**IBEP20** and **ITToken** respectively).

   Recommendation: Please consider using interfaces as argument to **initReserve** function and also do a zero-address check to make sure there not 0x0 provided.

   Fixed before the second review.

4. No event on lendingPoolConfigurator change

   While **lendingPoolConfigurator** has much powers it would be good to have an event emited when it's changed for easily off-chain tracking.

Recommendation: Please emit an event in the **LendingPool.setConfiguration** function.

Fixed before the second review.

■ Low

1. Unused state variable

   Recommendation: Please remove or comment-out unused state variables: **LendingPoolStorage._flashLoanPremiumTotal** and **LendingPoolStorage._maxStableRateBorrowSizePercent**

   Fixed before the second review.

2. Public function should be external

   public functions that are never called by the contract should be declared **external** to save gas.

   Recommendation: Use the **external** attribute for functions never called from the contract.

   Fixed before the second review.

3. Unused function argument

   Recommendation: Please remove or comment-out unused function arguments: **address** reserve

   Fixed before the second review.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

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

As a result of the audit, security engineers found **4** high, **4** medium and **3** low severity issues.

After the second review security engineers found **1** medium severity issue.

After the third review security engineers found **no issues** in the current scope.

Notice:

This is the repository part audit. Most of the contracts in the repository were out of the current audit scope.

Notice 2:

Possible Flash Loan is fixed in the TToken contract (contracts/protocol/tokenization/TToken.sol) which is out of the scope of the current audit.

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 the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free 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 the security of smart contracts.

Technical Disclaimer

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