SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Team-Finance
Date: August 13th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for TeamFinance.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Transfer controller
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/trustswap/team-finance-contracts
Commit	622b66ddde1bf6840960524067b13722cf7e2c6f
Technical Documentation	NO
JS tests	NO
Timeline	11 AUGUST 2021 - 13 AUGUST 2021
Changelog	13 AUGUST 2021 - INITIAL AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Team 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 11th, 2021 - August 13th, 2021.

Scope

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

```
Repository:
      https://github.com/trustswap/team-finance-contracts
Commit:
      622b66ddde1bf6840960524067b13722cf7e2c6f
Technical Documentation: No
JS tests: No
Contracts:
      LockToken.sol
      IPriceEstimator.sol
      IERC20Extended.sol
```

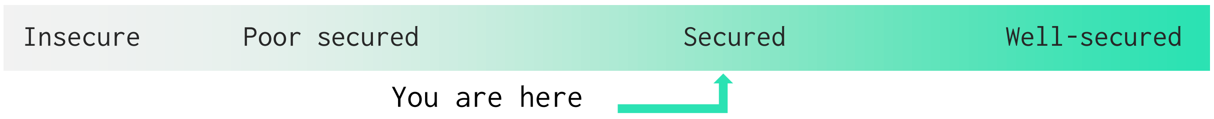
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

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.



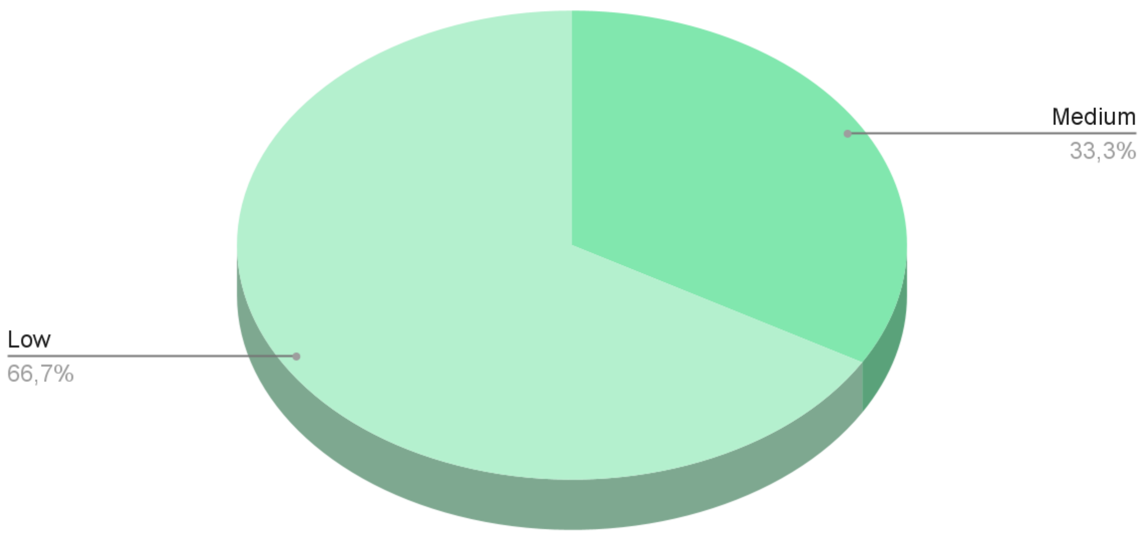
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview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

As a result of the audit, security engineers found 1 medium and 2 low severity issues.

Notice:

The LockToken contract contains no implemented IPriceEstimator functionality. We may not guarantee the secureness or correctness of the calculations after this interface will be implemented.

Graph 1. The distribution of vulnerabilities after the 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

No high severity issues were found.

🟩 🟩 Medium

1. Unknown interface implementation

   The contract has IPriceEstimator interface contract which could be changed anytime by the owner. We can neither look at nor check the functionality of the implementation.

   Recommendation: Please make it more clear what's the IPriceEstimator contract is and, maybe, add an additional event or event argument, containing an updated IPriceEstimator address.

🟦 Low

1. Style guide violation: Code Layout

   A solidity declares a style guide and a code layout guide as a part of it. It is very important to follow them because it makes a code more readable and easy to audit

   Recommendation: Please follow the code layout guide.

2. A public function that could be declared 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.

Lines: LockToken.sol#72-82

```
function lockTokens(
   address _tokenAddress,
   address _withdrawalAddress,
   uint256 _amount,
   uint256 _unlockTime
)
public
payable
whenNotPaused
returns (uint256 _id)
{
```

Lines: LockToken.sol#128-138

```
function createMultipleLocks(
   address _tokenAddress,
   address _withdrawalAddress,
    uint256[] memory _amounts,
    uint256[] memory _unlockTimes
)
public
payable
whenNotPaused
returns (uint256 _id)
{
```

Lines: LockToken.sol#193-198

```
function extendLockDuration(
    uint256 _id,
    uint256 _unlockTime
)
public
{
```

Lines: LockToken.sol#211-216

```
function transferLocks(
    uint256 _id,
    address _receiverAddress
)
public
{
```

Lines: LockToken.sol#245-249

```
function withdrawTokens(
    uint256 _id
)
public
{
```

Lines: LockToken.sol#353

```
function getTotalTokenBalance(address _tokenAddress) view public returns
(uint256)
```

Lines: LockToken.sol#359

```
function getTokenBalanceByAddress(address _tokenAddress, address
_walletAddress) view public returns (uint256)
```

Lines: LockToken.sol#365

```
function getAllDepositIds() view public returns (uint256[] memory)
```

Lines: LockToken.sol#371

```
function getDepositDetails(uint256 _id) view public returns (address
_tokenAddress, address _withdrawalAddress, uint256 _tokenAmount, uint256
_unlockTime, bool _withdrawn)
```

Lines: LockToken.sol#378

```
function getDepositsByWithdrawalAddress(address _withdrawalAddress) view
public returns (uint256[] memory)
```

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 1 medium and 2 low severity issues.

Notice:

The LockToken contract contains no implemented IPriceEstimator functionality. We may not guarantee the secureness or correctness of the calculations after this interface will be implemented.

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.