HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Tenderize
Date: April 22nd 2022

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 Tenderize
Approved by	Andrew Matiukhin | CTO Hacken OU Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Tenderize/tender-core
Commit	1fd606141625171fe792045ae9233890262d2d62 - Initial Audit 5b3b625d09e0bc02529946d9f2128af0c08c1a31 - Remediation Check
Deployed contract	No
Technical Documentation	Yes - https://docs.tenderize.me/contracts/liquidity/tenderfarm#itenderfarmaddrewardsuint256
JS tests	Yes
Website	https://www.tenderize.me/
Timeline	4 MARCH 2022 - 18 APRIL 2022
Changelog	11 MARCH 2022 - Initial Audit 22 APRIL 2022 - Remediation Check

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Tenderize (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 contracts.

Scope

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

Repository:
https://github.com/Tenderize/tender-core
Commit: 1fd606141625171fe792045ae9233890262d2d62 - Initial Audit
5b3b625d09e0bc02529946d9f2128af0c08c1a31 - Remediations Check
Technical Documentation: Yes
JS tests: Yes
Contracts:
TenderFarm.sol
ITenderFarm.sol
Graph.sol
IGraph.sol
IMatic.sol
Matic.sol
Livepeer.sol
ILivepeer.sol
TenderToken.sol
ITenderToken.sol
Tenderizer.sol
ITenderizer.sol
WithdrawalLocks.sol
WithdrawalPool.sol
TenderFarmFactory.sol
MathUtils.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

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided high-level documentation. This documentation contains detailed explanations of the desired workflow of the protocol and the functions used. The total Documentation Quality score is **10** out of **10**.

Code quality

The total Code Quality score is **8** out of **10**. The code follows almost all of the official guides and best practices. Detailed unit tests were provided.

Architecture quality

The Architecture quality score is **10** out of **10**. The code generally follows the desired best practice patterns.

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

Security score

As a result of the audit, security engineers found **3** high, **5** medium, and **8** low severity issues. The security score is **0** out of **10**.

As a result of the second review, security engineers found no new issues. The project contains **1** medium and **2** low issues. The security score is **10** out of **10**.

All found issues are displayed in the “Issues overview” section.

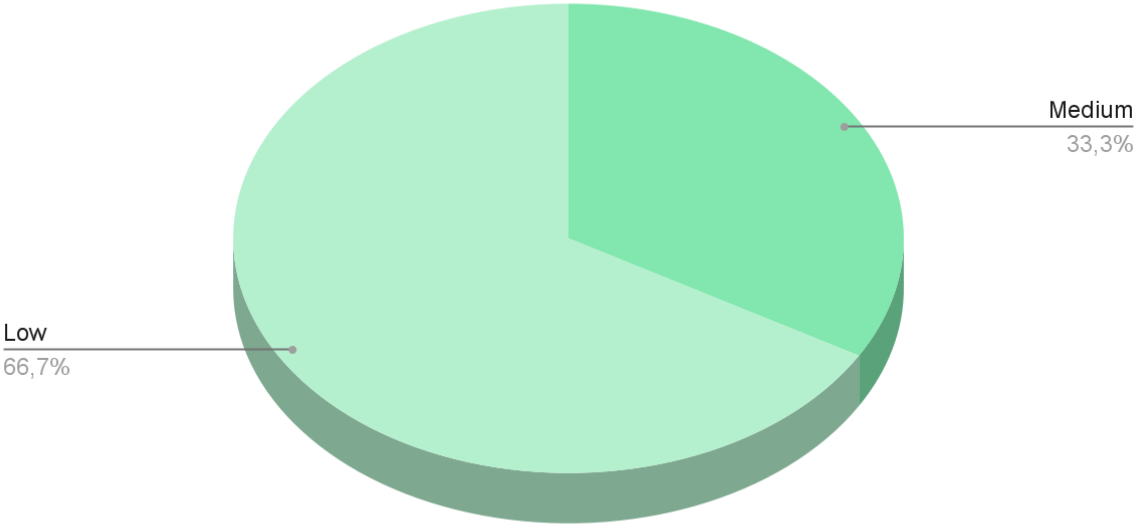
Summary

According to the assessment, the Customer's smart has the following score: **9.8**

NOTICE: Since the underlying system’s working process is not in the scope and clear, we assume that it is safe.

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

Graph 1. The distribution of vulnerabilities after the audit.



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

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 cannot lead to asset loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

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

Audit overview

Critical

No critical issues were found.

High

1. The contract allows Transfer to Arbitrary Addresses

   The users can transfer funds to arbitrary addresses.

   Contracts: Tenderizer.sol

   Function: execute(), batchExecute()

   Recommendation: Review and fix this logic.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

2. Unstake allows without burning tokens

   Unstake() function allows governors to unstake() without burning tokens. This can lead to contract depletion.

   Contracts: Tenderizer.sol

   Function: unstake()

   Recommendation: Review and fix this logic.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

3. _claimSecondaryRewards() function ignores errors and return values

   In Livepeer.sol contract, _claimSecondaryRewards() function ignores errors and returns values during the reward conversion to Uniswap in the try/catch. This makes contract impossible to work with WETH.

   Contracts: Livepeersol.sol

   Function: _claimSecondaryRewards()

   Recommendation: Review and fix this logic.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

Medium

1. Missing Allowance Configurations

   In the Tenderizer.sol contract, the deposit function transfers tokens via transferFrom function. There should be allowance control and configuration for the related parties before that process.

   Contracts: Tenderizer.sol

   Function: -

   Recommendation: Implement control mechanisms.

   Status: Mitigated. The active version of the *transferFrom* function does not require allowance configurations.

2. Unchecked Return

   In integration contracts, _stake functions use approve() methods to create necessary allowance amounts, but their returns are never checked.

   In Livepeer.sol, _claimSecondaryRewards function uses the approve() function to transfer WETH, but their returns are never checked.

   Contracts: Livepeer.sol, Matic.sol, Graph.sol

   Function: _stake, _claimSecondaryRewards

   Recommendation: Implement control mechanisms.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

3. Unchecked Transfer

   In integration contracts, _withdraw functions call transfer and does not check its return value.

   Contracts: Livepeer.sol, Matic.sol, Graph.sol

   Function: _withdraw

   Recommendation: Implement control mechanisms.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

4. Function Call Order Assumption

   In Livepeer.sol, WETH address is set by the setUniwapRouter() function. However, this function is never called in the constructor. It is assumed that it will be called when it is being deployed.

   Contracts: Livepeer.sol

   Function: -

Recommendation: Set WETH address via setUniwapRouter() in the constructor.

Status: Reported

5. Useless Event Description

In the Tenderizer.sol contract, the GovernanceUpdate event is ambiguous.

Contracts: Tenderizer.sol

Function: -

Recommendation: Include the address in the event description.

Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

Low

1. Missing Zero Address Validation

In the TenderToken.sol contract, initialize never checks TotakStakedReader address for potential 0x0. The shares of the function never check its parameter for potential 0x0.

In the Livepeer.sol contract, initialize never checks livepeer address for potential 0x0.

The Matic.sol contract does not check if addresses matic and matic manager are 0x0.

In the Graph.sol contract, initialize never checks graph address for potential 0x0.

In the TenderFarm.sol contract, initialize never checks its parameters for potential 0x0.

In the WithdrawPools.sol contract, unlock never checks the account for potential 0x0.

Contracts: Livepeer.sol, Matic.sol, Graph.sol, WithdrawPool.sol, TenderFram.sol, TenderToken.sol

Function: initialize(), unlock ()

Recommendation: Implement a missing zero address check.

Status: Reported

2. Redundant Return

In the TenderToken.sol contract, _mint and _mintShare functions return unnecessary uint values.

Contracts: TenderToken.sol

Function: _mint(), _mintShare

Recommendation: Remove returns.

Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

3. Comment - Implementation Mismatch

   The Matic.sol contract's _stake function, *if no amount is specified, stakes all available tokens, but the logic is not implemented that way.*

   Contracts: Matic.sol

   Function: _stake()

   Recommendation: Review and fix this logic.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

4. Convert If statement to Require

   In the Matic.sol contract, the _stake function uses an if statement to check if _node is zero, and returns if it is.

   Contracts: Matic.sol

   Function: _stake()

   Recommendation: Use a require statement instead.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

5. Redundant Variable Control

   In the Matic.sol contract, the _stake function uses an if statement to check if _node is zero and if it is not checks it again and modifies its value

   Contracts: Matic.sol

   Function:_stake()

   Recommendation: Remove redundant check.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

6. Redundant Variable Control

   In the Matic.sol contract, the _withdraw function checks if balAfter is bigger than the balance difference, and if they are equal, a require statement below reverts the function.

   Contracts: Matic.sol

   Function:_withdraw()

Recommendation: Use revert in the first check.

Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

7. Ignored Return Value

   In the Graph.sol contract, during the staking, delegate method is used, but its return value is ignored.

   Contracts: Graph.sol

   Function: _stake()

   Recommendation: Implement checks.

   Status: Fixed (5b3b625d09e0bc02529946d9f2128af0c08c1a31)

8. Functions that can be Declared as *external*

   In order to save Gas, public functions that are never called in the contract should be declared as *external*.

   Contracts: Tenderizer.sol, TenderFram.sol, TenderToken.sol

   Function: initialize(), selfPermit(), farm(), farmWithPermit(), farmFor(), unfarm(), harvest(), addRewards(), availableRewards(), stakeOf(), deposit(), claimRewards() , calcDepositOut()

   Recommendation: Implement checks.

   Status: Reported

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

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with 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 a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. 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 cannot guarantee the explicit security of the audited smart contracts.