HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: CSS_LeechProtocol
**Date**: February 8, 2023

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 CSS_LeechProtocol |
| **Approved By** | Evgeniy Bezuglyi \| SC Audits Department Head at Hacken OU |
| **Type** | Farm |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methodology** | Link |
| **Changelog** | 29.12.2022 - Initial Review<br>25.01.2023 - Second Review<br>08.02.2023 - Third Review |

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 CSS_LeechProtocol (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:

### Initial review scope

| | |
|---|---|
| **Repository** | https://github.com/Leech-Protocol/farm-contract/ |
| **Commit** | 921b7dda5e83e4505cb33e041b1582669fcef7b2 |
| **Technical Requirements** | Link |
| **Contracts** | File: ./contracts/Context.sol<br>SHA3:<br>b352b9c9ce4d7b9d16589b4b398e6cdadb8b7a82857597d5ef43262ee5131cef<br><br>File: ./contracts/Farm.sol<br>SHA3:<br>a1fbb1a1504d1125218878205fba0ea3abe3f7e6d9605482495e49d56df77ed2<br><br>File: ./contracts/FarmCore.sol<br>SHA3:<br>c0f848316b27534219e54cf515f0d33c14e423c577efe6a20144a7cdfea85549<br><br>File: ./contracts/IERC20.sol<br>SHA3:<br>83f3ba28dbcb0cda61852e0148b3473815ccf4e6e0879ab4fecfdf31f793604d<br><br>File: ./contracts/Ownable.sol<br>SHA3:<br>3355bf742a33a01b6e0a6dbce72f23b058b2137f88203514d33dec9845d593e2 |

### Second review scope

| | |
|---|---|
| **Repository** | https://github.com/Leech-Protocol/farm-contract/ |
| **Commit** | f68f292577ccf1919153e9a99cfd9a0c1761df7e |
| **Technical Requirements** | Link |
| **Contracts** | File: ./contracts/Context.sol<br>SHA3:<br>6ad58d09c85257dd94cc378aaff5b3862cbe3e640066155248f8a2c64e3c689b<br><br>File: ./contracts/Farm.sol<br>SHA3:<br>21d49279169d6782c135b91c2aed8c379c609ba3b38bb4b11629566cd5ee1bcf<br><br>File: ./contracts/FarmCore.sol |

| | |
|---|---|
| | SHA3:<br>33691f2c6d77b0d74eb119293c04466672547ed6d578ada10861795e4a46f91a<br><br>File: ./contracts/IERC20.sol<br>SHA3:<br>dcd808947ee6a949d53349c3e457860da91943fe98e17298ba620c98150358c2<br><br>File: ./contracts/Ownable.sol<br>SHA3:<br>03e961a326fabe733dc7a3894e62e5295f03bcf289b58cce3688c201931a9bbd |

## Third review scope

| | |
|---|---|
| **Repository** | https://github.com/Leech-Protocol/farm-contract/ |
| **Commit** | 3cc1690e3c1b6b21b15c37faab95accc4e7b025f |
| **Technical Requirements** | Link |
| **Contracts** | File: ./contracts/Farm.sol<br><br>SHA3:<br>3bec5492ba96755bca3a531f29d4ea80d46728186ef67e14acb34d3214529f5f<br><br>File: ./contracts/IFarm.sol<br><br>SHA3:<br>086a5375534dd9d41c9f14693bf444643d8fa68ebb3b6019c358a6ede66f90e6 |

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 the loss of user funds or contract state manipulation by external or internal actors. |
| High | High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors. |
| Medium | Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category. |
| Low | Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality |

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

## Executive Summary

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

### Documentation quality

The total Documentation Quality score is **10** out of **10**.

### Code quality

The total Code Quality score is **10** out of **10**.

### Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and user interactions are covered with tests.

### Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

### Summary

According to the assessment, the Customer's smart contract has the following score: **10.0**.

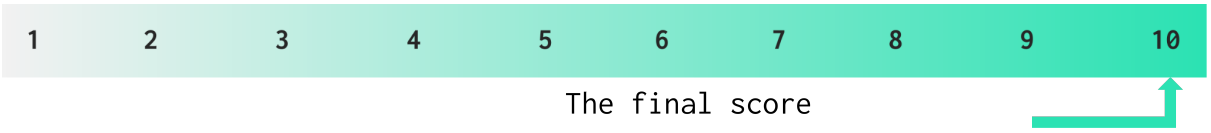


***Table. The distribution of issues during the audit***

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 29 December 2022 | 1 | 0 | 2 | 0 |
| 25 January 2023 | 0 | 0 | 0 | 0 |
| 7 February 2023 | 0 | 0 | 0 | 0 |

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

## Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

| Item | Type | Description | Status |
|---|---|---|---|
| **Default Visibility** | SWC-100 SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | Passed |
| **Integer Overflow and Underflow** | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | Not Relevant |
| **Outdated Compiler Version** | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | Passed |
| **Floating Pragma** | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | Passed |
| **Unchecked Call Return Value** | SWC-104 | The return value of a message call should be checked. | Passed |
| **Access Control & Authorization** | CWE-284 | Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users. | Passed |
| **SELFDESTRUCT Instruction** | SWC-106 | The contract should not be self-destructible while it has funds belonging to users. | Not Relevant |
| **Check-Effect-Interaction** | SWC-107 | Check-Effect-Interaction pattern should be followed if the code performs ANY external call. | Passed |
| **Assert Violation** | SWC-110 | Properly functioning code should never reach a failing assert statement. | Passed |
| **Deprecated Solidity Functions** | SWC-111 | Deprecated built-in functions should never be used. | Passed |
| **Delegatecall to Untrusted Callee** | SWC-112 | Delegatecalls should only be allowed to trusted addresses. | Not Relevant |
| **DoS (Denial of Service)** | SWC-113 SWC-128 | Execution of the code should never be blocked by a specific contract state unless required. | Passed |
| **Race Conditions** | SWC-114 | Race Conditions and Transactions Order Dependency should not be possible. | Passed |

| | | | |
|---|---|---|---|
| **Authorization through tx.origin** | SWC-115 | tx.origin should not be used for authorization. | Not Relevant |
| **Block values as a proxy for time** | SWC-116 | Block numbers should not be used for time calculations. | Not Relevant |
| **Signature Unique Id** | SWC-117 SWC-121 SWC-122 EIP-155 EIP-712 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification. | Not Relevant |
| **Shadowing State Variable** | SWC-119 | State variables should not be shadowed. | Passed |
| **Weak Sources of Randomness** | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | Not Relevant |
| **Incorrect Inheritance Order** | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. | Passed |
| **Calls Only to Trusted Addresses** | EEA-Level-2 SWC-126 | All external calls should be performed only to trusted addresses. | Not Relevant |
| **Presence of Unused Variables** | SWC-131 | The code should not contain unused variables if this is not justified by design. | Passed |
| **EIP Standards Violation** | EIP | EIP standards should not be violated. | Not Relevant |
| **Assets Integrity** | **Custom** | Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract. | Passed |
| **User Balances Manipulation** | **Custom** | Contract owners or any other third party should not be able to access funds belonging to users. | Passed |
| **Data Consistency** | **Custom** | Smart contract data should be consistent all over the data flow. | Passed |
| **Flashloan Attack** | **Custom** | When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. | Not Relevant |

| | | | |
|---|---|---|---|
| **Token Supply Manipulation** | **Custom** | Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer. | Not Relevant |
| **Gas Limit and Loops** | **Custom** | Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit. | Not Relevant |
| **Style Guide Violation** | **Custom** | Style guides and best practices should be followed. | Passed |
| **Requirements Compliance** | **Custom** | The code should be compliant with the requirements provided by the Customer. | Passed |
| **Environment Consistency** | **Custom** | The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code. | Passed |
| **Secure Oracles Usage** | **Custom** | The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles. | Not Relevant |
| **Tests Coverage** | **Custom** | The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested. | Passed |
| **Stable Imports** | **Custom** | The code should not reference draft contracts, which may be changed in the future. | Passed |

## System Overview

*CSS_LeechProcol* is a mixed-purpose system with the following contracts:

- *Context.sol* - Provides information about the current execution context.
- *Farm.sol* - Allows users to deposit and withdraw funds.
- *FarmCore.sol* - Contains setter and getter functions for storage variables.
- *Ownable.sol* - Restricts and controls access to specific functions.
- *IERC20.sol* - An interface of ERC20 token.

### Privileged roles

- The admin of the Farm can block and unblock user from interacting with smart contract.
- The admin of the FarmCore can set service percent, disable and enable all functions of the smart contract, set MINTVL, CAPY, MINAPY, farm service, farm pool and farm pair.
- The owner of the FarmCore can set withdraw address, start and end the moving from farm to farm, add and remove admins.

### Risks

- In case of an admin keys leak, an attacker can lock access to funds that belong to users.

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

# Findings

## Critical

No critical severity issues were found.

## High

### H01. Highly Permissive Role Access

The owners of the project can set deposit and withdraw addresses, set service percent, start and stop farm, enable and disable services, remove and add admins, set farm pairs and pools, block and unblock users.

**Path:** ./contracts/*

**Recommendation**: Add highly permissive functionality to the *public* documentation.

**Status**: Fixed (documentation link)

### H02. Checks-Effects-Interactions Pattern Violation

The code violates the CEI pattern. This lowers code quality and can lead to reentrancy attacks.

**Path:** ./contracts/Farm.sol: deposit();

**Recommendation**: Refactor code to fit CEI pattern.

**Status**: Fixed (revised commit: f68f292)

## Medium

No medium severity issues were found.

## Low

### L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

**Path:** ./contracts/*

**Recommendation**: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

**Status**: Fixed (revised commit: f68f292)

# Disclaimers

## Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

## 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 Consultant cannot guarantee the explicit security of the audited smart contracts.