HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LunaFi_Technologies_Ltd
Date: September 9, 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 LunaFi_Technologies_Ltd
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Token pool
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.lunafi.io/
Timeline	05.07.2022 - 05.09.2022
Changelog	20.07.2022 - Initial Review 11.08.2022 - Second Review 09.09.2022 - 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 LunaFi_Technologies_Ltd (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/Luna-Fi/lunafi-smart-contracts-v2/tree/audit-branch
Commit:
8c76790967a9e3083876a26e4b84db0ffc092fa6
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/HousePool.sol
SHA3: ff7dd3b8bb361dcf6617995718b188b94fb07c9de2e2d7e1a133bd0a704489eb

Second review scope

Repository:
https://github.com/Luna-Fi/lunafi-smart-contracts-v2/tree/audit-branch
Commit:
ddf40ccf1aeb82e9f937744702b58a068db64710
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Type: Documentation
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/HousePool.sol
SHA3: 005c6552b6e2a7b65a29530763a00a990d4b575e79c902a0c1d95eafaea8a297

Third review scope

Repository:
https://github.com/Luna-Fi/lunafi-smart-contracts-v2/tree/audit-branch
Commit:
b27c8ee69ab8c47cac320730c6f19f9ce391d0b4
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Type: Documentation
Link

```
      Type: Technical Specification and Audit Responses
      Link

Integration and Unit Tests: Yes
Contracts:
      File: ./contracts/HousePool.sol
      SHA3: 82cb18d91abd6121fcb96131f893f077131703bdf5c75c70bde007778d509983
```

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 assets 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

Executive Summary

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

Documentation quality

The total Documentation Quality score is **8** out of **10**. Functional requirements are provided in a whitepaper. A technical description is not provided. Code is followed by NatSpec comments.

Code quality

The total CodeQuality score is **8** out of **10**. Code violates Style guide. Unit tests were provided. **Total test coverage is 78%**.

Architecture quality

The architecture quality score is **7** out of **10**. Single responsibility principle is violated.

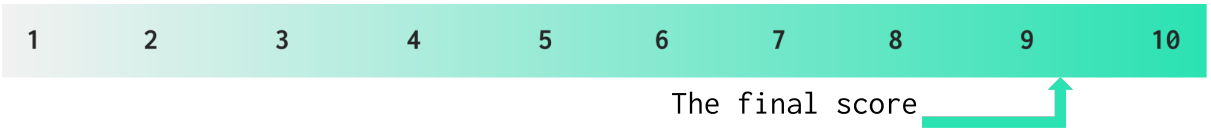
Security score

As a result of the audit, the code contains **2** low severity 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: **9.3**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Passed
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 it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Not Relevant

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Failed
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.	Passed
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.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	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	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
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.	Failed
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 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

HousePool is a token pool with the following contract:

- HousePool.sol - a contract with logic for working with a token pool. Contract performs calculations and implements external calls to other token contracts.

Privileged roles

- MANAGER_ROLE can get a sports book contract, set and get a cooldown active state.
- DATA_PROVIDER_ORACLE can update values of interest.
- STAKING_MANAGER can set an unstake window, cooldown seconds and reward per second.

Risks

- Contract performs external calls to token contracts that are not included in the audit scope. This report can evaluate security only for contracts included in the scope.

Findings

■■■■ Critical

1. Non-finalized code

The code contains commented code parts and TODO statements. Due to this, contract logic seems unfinished, and additional changes will be introduced in the future.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: storeBets (line 716), settleBets (lines 767, 775, 780), updateBets (lines 738, 739)

Recommendation: Finalize code logic and remove TODO statements.

Status: Fixed (revised commit: **ddf40cc**)

2. Requirements violation

The code violates the requirements provided by the Customer. In the documentation, *totalValueLocked* is counted by formula “Liquidity + Expected value of pending bets - Pending stakes - Pending commission”, while in code, Pending commission is absent.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: stake, updateAttributes, _updateTVL

Recommendation: Implement the code according to requirements.

Status: Fixed (revised commit: **ddf40cc**)

■■■ High

1. Highly permissive role access

Owner can change *unstakeWindowTime, cooldownSeconds, rewardPerSecond* and *cooldownActiveState*. Such permissions should be properly and in detail described in the documentation, so the users will be notified about such functionality.

This can lead to users' fund manipulations.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: stake, unstake, claimRewards, settleBets

Recommendation: Add highly permissive functionality to documentation.

Status: Fixed (revised commit: **b27c8ee**)

2. **Stack overflow possibility**

   The code performs a large number of multiplications and exponentiations of large numbers. Given that the contract often uses the int256 data type, there is a possibility that there will be a stack overflow and function execution revert.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: stake, unStake, getMaxWithdrawal

   Recommendation: If possible, use uint256 instead of int256 or store data in a few values.

   Status: Mitigated (with Customer's notice)

3. **Denial of service vulnerability**

   Loop in function *settleBets* performs external calls. Iterating over large structures and performing external calls in loops may lead to out-of-Gas exceptions.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Function: settleBets

   Recommendation: Implement size limitations and avoid external calls inside a loop.

   Status: Fixed (revised commit: **b27c8ee**)

Medium

1. **Unchecked token transfer**

   Contract does not check the return result of ERC20 token transfer. In case of this transfer failure, the function keeps running. ERC20 transfer functions return bool after transfers, and it is important to implement a return value check for this return value.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: stake, unstake, claimRewards, settleBets

   Recommendation: Implement a return value check for token transfers.

   Status: Fixed (revised commit: **b27c8ee**)

2. **Unoptimized loops usage**

   Contract reads and modifies state variables inside a loop. Loops are unoptimized, and their usage can lead to high Gas taxes.

   File: ./contracts/HousePool.sol

Contract: HousePool

Functions: storeBets, updateBets, settleBets

Recommendation: Cache arrays in a loop, save state variables to local memory, iterate the loop and save changes to the state after the loop finishes.

Status: Fixed (revised commit: **b27c8ee**)

3. **Reusable signatures**

Contract uses signatures to authenticate users. After a signature is used, anyone will be able to interact with the contract by repeating the function call with the same signature.

This can lead to users' fund manipulations.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: -

Recommendation: Add functionality to make every signature unique.

Status: Mitigated (with Customer’s notice)

4. **Code simplification possibility**

Contract declares some variables private, but there are external getter functions for these variables. If such variables are declared public, such functions will be declared automatically. This will save Gas and make code cleaner and easier to read.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: getRewardPerSecond, getCooldownSeconds, getUnstakeWindowTime, getPoolName, getPendingStakesValue, getTotalValueLocked, getLPTokenPrice, getLiquidityStatus

Recommendation: Declare variables as public and remove getter functions.

Status: Fixed (revised commit: **ddf40cc**)

5. **Redundant logic**

Logical checks that duplicate the logic of the previous statements or do not perform any validation can be removed to save Gas.

File: ./contracts/HousePool.sol

Contract: HousePool

Function: verify

Recommendation: Remove redundant logic.

Status: Fixed (revised commit: **b27c8ee**)

Low

1. **State variables' default visibility**

   Specifying state variables' visibility helps to catch incorrect assumptions about who can access the variable.

   This makes the contract`s code quality and readability higher.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Variables: MAX_PRECISION, bets, farmInfo

   Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

   Status: Fixed (revised commit: **ddf40cc**)

2. **Functions can be declared external**

   Public functions that are never called by the contract should be declared external to save Gas.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: getRewards, setCoolDownActiveState, getCoolDownActiveState, storeBets, updateBets, settleBets

   Recommendation: Declare mentioned functions as external.

   Status: Fixed (revised commit: **ddf40cc**)

3. **Missing zero address validation**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: initialize, setSportsBookContract, permitAndStake, stake, claimRewards

   Recommendation: Implement a zero address check.

   Status: Fixed (revised commit: **ddf40cc**)

4. **Boolean equality**

   Boolean constants can be used directly and do not need to be compared to true or false.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: stake, unStake, activateCooldown, getNextCooldownTimestamp, updateBets, _transfer

Recommendation: Remove boolean equality.

Status: Fixed (revised commit: **ddf40cc**)

5. **Redundant pragma statement**

   Pragma ABIEncoderV2 will be activated by default starting from Solidity 0.8.0.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: -

   Recommendation: Remove redundant statement.

   Status: Fixed (revised commit: **ddf40cc**)

6. **Redundant conversion**

   Some code parts implement data type conversion when converting uint256 to int256 even if the result value cannot drop under zero.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Functions: -

   Recommendation: Remove redundant conversions.

   Status: Mitigated (with Customer's notice)

7. **Invalid event emit values**

   Event UnStaked is emitted with information about the address that unstakes tokens, the address which receives the tokens, and the tokens amount. However, in event emitting, no matter what, both unstaker and receiver are *msg.sender*.

   File: ./contracts/HousePool.sol

   Contract: HousePool

   Function: unStake

   Recommendation: Change event emitting values.

   Status: Fixed (revised commit: **ddf40cc**)

8. **Code optimization possibility**

There is a constant variable in code equal to 10e18. It can be used instead of all mathematical operations with 10**18.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: -

Recommendation: Replace mathematical operations with declared variable.

Status: Fixed (revised commit: **b27c8ee**)

9. **Unused state variables**

Some state variables are declared, but never used.

File: ./contracts/HousePool.sol

Contract: HousePool

Variables: bettingStakes, totalPayouts, bets, stakerRewardsToClaim

Recommendation: Remove unused variables.

Status: Fixed (revised commit: **ddf40cc**)

10. **Boolean equality**

Boolean constants can be used directly and do not need to be compared to true or false.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: claimRewards, settleBet, stake, unStake

Recommendation: Remove boolean equality.

Status: New

11. **Redundant import**

Contract imports SafeERC20 contract without using it.

File: ./contracts/HousePool.sol

Contract: HousePool

Functions: -

Recommendation: Remove redundant import.

Status: New

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.