HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Acta Finance/P2P Solutions LTD
Date: May 18th, 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 Acta Finance/P2P Solutions LTD.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	Staking with a referrals program
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	
Timeline	29.03.2022 - 18.05.2022
Changelog	04.04.2022 - Initial Review 14.04.2022 - Second review 29.04.2022 - Third review 18.05.2022 - Fourth 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 Acta Finance/P2P Solutions 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.

The second review was conducted on April 14th, 2022.

The third review was conducted on April 29th, 2022.

The fourth review was conducted on May 18th, 2022.

Scope

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

Repository:
https://github.com/ActaFi/actafi-avalanche-contracts/tree/master
Commit:
bb752219be4b8b90cd7be4a50d2d81d9618d9e43
Technical Documentation: Yes
JS tests: Yes
Contracts:
./contracts/utils/RewardStorage.sol
./contracts/Milestones.sol
./contracts/Referrals.sol
./contracts/StakingPool.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 ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency

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

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

Executive Summary

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

Documentation quality

The Customer provided a whitepaper with functional requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **7** out of **10**. The code is well commented and partly covered with unit tests.

Architecture quality

The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

Security score

As a result of the audit, security engineers found **1** critical, **1** high, **3** medium, and **3** low severity issues. The security score is **0** out of **10**. All found issues are displayed in the “Issues overview” section.

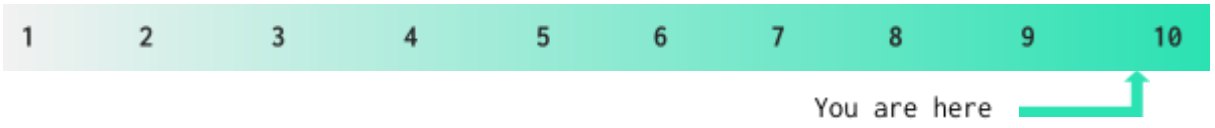
After the second review, the code contains **1** medium severity issue. The security score is **10** out of **10**.

After the third review, the code contains **1** high, **1** medium and **2** low severity issues. The security score is **5** out of **10**.

After the fourth review, the code contains **1** low severity issue. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.7**



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 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

Findings

Critical

1. **The production code has a function that was used for unit tests.**

   The staking contract has a function that allows to arbitrarily age the staking deposit. This may lead to the leakage of funds in the staking pool.

   Contracts: ./contracts/StakingPool.sol

   Function: _AGE_DEPOSIT_.

   Recommendation: Remove the function.

   Status: Fixed (Revised commit: e1a9169)

High

1. **The pool balance is potentially overestimated.**

   The Referrals.sol contract has the `_poolBalance` variable, which stores the amount of tokens in the pool. When somebody creates a referral by calling `createReferral` it takes the fee only when the user created the first link, but the internal `createNewLink` function adds the fee value after each link is created.

   After some time, the `_poolBalance` value may overestimate the actual contract balance.

   Contracts: ./contracts/Referrals.sol

   Function: createReferral, createNewLink

   Recommendation: Rework the function to add the fee amount to the balance variable only when the user pays the fee.

   Status: Fixed (Revised commit: e1a9169)

2. **Insufficient balance.**

   Rewards can be paid from staking balances.

   This will lead to issues with the contract management and funds accountability. If all users withdraw their stakes, some will not be able to do this due to balance insufficiency.

   Contracts: ./contracts/StakingPool.sol

   Recommendation: Separate staking and rewards balances

   Status: Fixed (Revised commit:cb76c7f)

Medium

1. **Mapping values overlapping.**

   The Referrals.sol contract has the code which allows the contract owner to rewrite values for keys in `hashLinkMap` mapping.

Rewriting `hashLinkMap` mapping may cause errors when calling `claimReferralByHashString` function.

Contracts: ./contracts/Referrals.sol

Function: createReferral

Recommendation: Rework statement condition not to pass owner.

Status: Fixed (Revised commit: e1a9169)

2. **Unexpected event emitting.**

   The Referrals.sol contract has the `claimReferralByHashString` function, which emits the `ClaimedReferralFailed` event whenever the function is called.

   Contracts: ./contracts/Referrals.sol

   Function: claimReferralByHashString

   Recommendation: Rework the function logic to emit `ClaimedReferralFailed` event only when there is no such hash key.

   Status: Fixed (Revised commit: e1a9169)

3. **Check effect interaction pattern violations.**

   The contracts interact with the token contract through the `IERC20` interface, but the token contract may have the correct functions interface and arbitrary internal functions implementation with a reentrancy call.

   The reentrancy attack is possible during the external call to an untrusted token contract.

   Contracts: ./contracts/StakingPool.sol

   Function: refund

   Recommendation: Interact only with trusted contracts, check external calls for reentrancy, and do the state variable modification before an external call execution.

   Status: Fixed (Revised commit: aa1694e)

Low

1. **Redundant use of SafeMath library.**

   SafeMath is not needed starting with Solidity 0.8. The compiler has built-in overflow checking.

   Contracts:
   ./contracts/Milestones.sol,
   ./contracts/Referrals.sol,
   ./contracts/StakingPool.sol

   Recommendation: Do not use SafeMath in the contract.

   Status: Fixed (Revised commit: e1a9169)

2. **Redundant counter variables.**

   The contracts have variables that count the number of items in the arrays, but arrays in solidity have a built-in `length` property.

   Contracts: ./contracts/Referrals.sol

   Recommendation: Remove `numOfAccounts`, `numOfLinks` variables. If needed create a getter function for the length property for `links` and `accounts` arrays.

   Status: Fixed (Revised commit: e1a9169)

3. **Redundant conditional statement.**

   The Referrals.sol contract has the `getUserLevel1Referals` function with the redundant conditional statement. For each `userAddress` the internal function loop iterates over `userHasLink[accounts[i].claimer]` values which may be the same for different `userAddress` input values.

   Contracts: ./contracts/Referrals.sol

   Function: getUserLevel1Referals

   Recommendation: To get the number of first-level referrals, it is enough to get the length of the `refMap[userAddress]` array.

   Status: Fixed (Revised commit: e1a9169)

4. **Unused event declaration.**

   The StakingPool.sol contract has the unused `FundStakingPool` event declaration.

   Contracts: ./contracts/StakingPool.sol

   Recommendation: Remove unused event or fire it in the proper function.

   Status: Fixed (Revised commit: 0402d5e)

5. **Redundant mapping declaration.**

   The Referrals.sol contract has a private mapping `userHasLink` that stores a boolean flag that detects if the user has created a link, but the mapping values are never used in the contract logic.

   The redundant mapping leads to unnecessary Gas spending.

   Contracts: ./contracts/Referrals.sol

   Recommendation: Remove redundant mapping.

   Status: Fixed (Revised commit: 75596af)

6. **Redundant import statements.**

   The RewardStorage.sol contract has the `ReentrancyGuard` import statement, which is not used in the contract.

Contracts: ./contracts/utils/RewardStorage.sol

Recommendation: Remove import statements for `ReentrancyGuard`.

Status: New

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.