HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Transient Networks Limited
Date: April 27th, 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 Transient Network Limited.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token; ERC677 token; Betting
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://transientnetwork.io
Timeline	06.04.2022 - 27.04.2021
Changelog	18.04.2022 - Initial Review 27.04.2022 - Second 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 Transient Network Limited (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/dev-team-transient/eSports-Smart-Contracts/tree/audit
Commit:
2730fcbe66db4c911e8cd0de2a4ba3d2bd800079 - Initial Audit
db530c9220a6cbfdd6c635c061699ae956f5c780 - Second Review
Documentation: Yes - https://docs.tnetwork.io
JS tests: No
Contracts:
CompetitionPool.sol
GuaranteedCompetitionContract.sol
P2PCompetitionContract.sol
RegularCompetitionContract.sol
CompetitionFactory.sol
GameScoreKeeperOracle.sol
SportManager.sol
String.sol
Metadata.sol
CompetitionContract.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:

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

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
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

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

Executive Summary

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

Documentation quality

The Customer provided functional requirements unrelated to this project and no technical requirements. The total Documentation Quality score is **0** out of **10**.

Code quality

The total CodeQuality score is **5** out of **10**. Code follows official language style guides. No unit tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**. Smart contracts of the project follow the best practices, and the project has a clear architecture.

Security score

As a result of the audit, security engineers found **1** critical, **1** high, **4** medium, and **9** low severity issues. The security score is **0** out of **10**.

As a result of the second review, security engineers found no new issues. **1** critical, **1** high, **1** medium and **8** low severity issues from the previous revision were fixed. As a result, the code contains **2** medium and **1** low severity issues. The security score is **10** out of **10**.

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

Summary

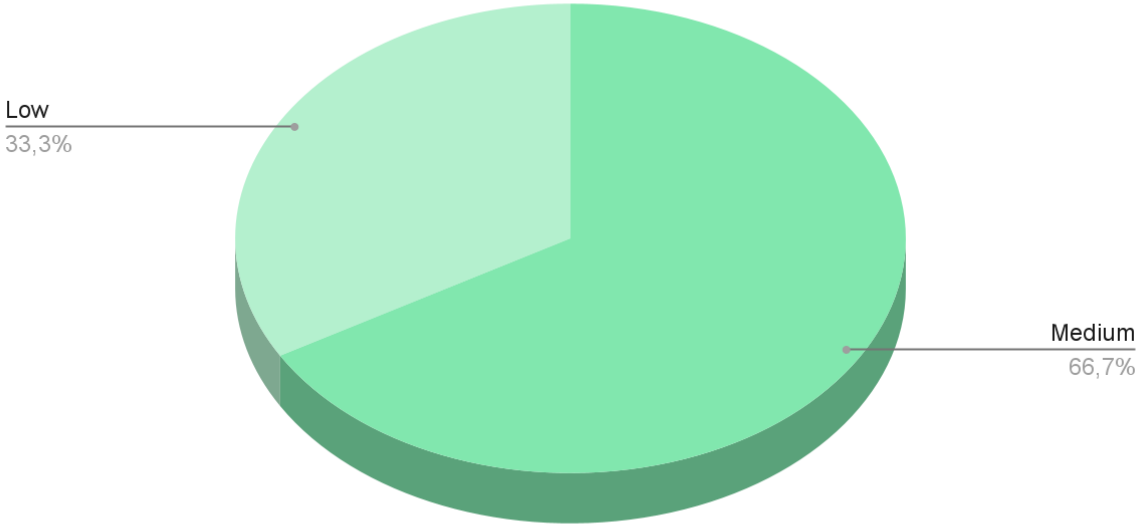
According to the assessment, the Customer's smart contract has the following score: **8.5**

Notice

1. ERC20 tokens being used in the competitions should adopt standard ERC20 implementations. Any implementation with a fallback logic can lead to unexpected behavior during token transfers, such as reverts.

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

Graph 1. The distribution of vulnerabilities after the second 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 asset 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

Findings

Critical

1. **Unauthorized Access.**

   The oracle address, the LINK token address, the job Id, and the payment value can be changed by anyone. A malicious oracle implementation can be appointed as the oracle that provides the results for the competitions, or the LINK token can be changed at any time by anyone with any (potentially malicious) token.

   This can lead to betting result manipulations and potential DoS vulnerabilities.

   Contracts: GameScoreKeeperOracle.sol

   Function: setOracle(), setToken(), setPayment(), setJobId()

   Recommendation: Implement role-based access for these functions.

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

High

1. **Unexpected function access.**

   The P2P betting contract has an external function that allows everyone to vote from any account and not pay a fee. The `vote` function adds `fee` to the `totalFee`, but it does not check if the tokens were transferred. The `distributeReward` function calculates the `ownerReward` based on the `totalFee` value and attempts to transfer `ownerReward` value, but the `safeTransfer` call will fail if the contract has not had enough tokens.

   The reward may not be sent if the `totalFee` does represent the wrong value.

   Contracts: P2PCompetitionContract.sol

   Function: vote()

   Recommendation: Allow to call vote function only from CompetitionPool contract.

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

Medium

1. **Missing Allowance Check**

   The safeTransferFrom() function, of ERC20 (of version 4.2.0), is being called in other functions. It does not have a built-in control mechanism for checking the allowance between the sender and the receiver, and the code never checks if there is enough allowance prior to calling it.

   This can lead to unexpected behaviors in the function execution.

   Contracts: CompetitionPool.sol

Function: createNewRegularCompetitionContract(), createNewGuaranteedCompetitionContract(), createNewP2PCompetitionContract(), voteP2P(), acceptP2P(), _placeBet()

Recommendation: Use the latest implementation of the ERC20 standard.

Status: Fixed (2730fcbe66db4c911e8cd0de2a4ba3d2bd800079)

2. **Checks-Effects-Interactions pattern violation.**

   The state variables are updated after competition result data has been gathered from the oracle.

   The state variables are updated after competition creation and configuration have been made.

   This can lead to unexpected behaviors in the function execution.

   Contracts: GameScoreKeeperOracle.sol, CompetitionPool.sol

   Function: requestData(), createNewRegularCompetitionContract(), createNewGuaranteedCompetitionContract(), createNewP2PCompetitionContract()

   Recommendation: Update state variables before making external calls.

   Status: Reported

3. **Potential Out-of-Gas exception.**

   Iterating over arrays (such as winners and bet slip lists) and making external calls (safeTransfer) may lead to enormous Gas consumption due to the arrays' size.

   Contracts: CompetitionPool.sol, Competition.sol

   Function: betSlip(), _sendRewardToWinner()

   Recommendation: Implement array size limitations.

   Status: Reported

Low

1. **Unused Return.**

   There are calls (setBasic()) made to handle basic configuration during the RegularCompetition and GuaranteedCompetition creations. However, the return results of these calls are never checked.

   This can lead to unexpected behaviors in the function execution.

   Contracts: CompetitionFactory.sol

   Function: createRegularCompetitionContract(), createGuaranteedCompetitionContract()

   Recommendation: Implement control mechanisms or remove the return statement from the function.

Status: Reported

2. **Unused local variable.**

   The variable `creatorReward` is created but never assigned a value. This variable is only used when delivering the prizes to the relevant addresses. However, this operation is unnecessary since no operation is performed with this variable.

   Contracts: P2PCompetitionContract.sol

   Function: distributeReward()

   Recommendation: Remove unused variable.

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

3. **Unused State variable.**

   The Regular Competition contract has declared a variable `notSupportAtribute` but never assigned a value and never used.

   Contracts: RegularCompetitionContract.sol

   Recommendation: Remove unused variable

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

4. **Redundant Code Block.**

   The local variable `creatorReward` is created but never assigned a value or used anywhere in the code. This means its value is always the default value for the type uint256, which is zero.Therefore, it is unnecessary to check if this variable is greater than zero and if it is, transferring it is unnecessary.

   Contracts: P2PCompetitionContract.sol

   Function: distributeReward()

   Recommendation: Remove this unused code block.

   Status: Fixed (2730fcbe664c911e8cd0de2a4ba3d2bd800079)

5. **Use of hard-coded values.**

   Hard-coded values are used in computations.

   Contracts: GameScoreKeeperOracle.sol

   Function: requestData(), bytesToUint256Array()

   Recommendation: Convert these variables into constants.

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

6. **Missing event emitting.**

   Events for critical state changes (e.g. owner and other critical parameters) should be emitted for tracking things off-chain.

   Contracts: CompetitionFactory.sol

Function: setCompetitionFactory(), setTypeToAddress(), setOracle()

Recommendation: Create and emit related events.

Status: Fixed (2730fcbe664c911e8cd0de2a4ba3d2bd800079)

7. **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.

   Contracts: GameScoreKeeperOracle.sol, Competition.sol, CompetitionFactory.sol, CompetitionPool.sol, P2PCompetition.sol

   Function: constructor(), setTypeToAddress(), setSportManager(), setOracle(),

   Recommendation: Implement zero address checks.

   Status: Fixed (2730fcbe66db4c911e8cd0de2a4ba3d2bd800079)

8. **Floating pragma.**

   The project uses floating pragma ^0.8.0.

   Contracts: GameScoreKeeperOracle.sol, Competition.sol, CompetitionFactory.sol, CompetitionPool.sol, P2PCompetition.sol, RegularCompetition.sol, GuaranteedCompetition.sol, Metadata.sol, SportManager.sol

   Function: -

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

   Status: Fixed (2730fcbe66db4c911e8cd0de2a4ba3d2bd800079)

9. **Functions That Can be Declared as *external*.**

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

   Contracts: GameScoreKeeperOracle.sol, Competition.sol, CompetitionFactory.sol, CompetitionPool.sol, P2PCompetition.sol, RegularCompetition.sol, GuaranteedCompetition.sol, Metadata.sol, SportManager.sol

   Function: setJobId(), setOracle(), setPayment(), getToken() , setToken(), requestData(), fulfill(), withdrawLink(), setFactory(), setTokenAddress(), withdrawToken(), betSlip(), acceptP2P(), voteP2P(), createNewRegularCompetition(), createNewGuaranteedCompetition(), createNewP2PCompetition(), renounceOwnership() , getTotalToken(), getAttributeById()

   Recommendation: Aforementioned should be declared as *external*.

   Status: Fixed (3e8d2c39c1f17f8cbedc5a1f5c97308aec156c5c)

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

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.