HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: PlayEstates
Date: November 8th, 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 PlayEstates
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Auction; GameFi; ERC721 token
Platform	EVM
Network	
Language	Solidity
Methodology	Link
Website	playestates.com
Changelog	11.10.2022 - Initial Review 26.10.2022 - Second Review 0.11.2022 - 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 PlayEstates (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/PlayEstate/contract-marketplace-v1
Commit:
49b54e7
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/engine/NFTEngineV1.sol
SHA3: d9029b3df75bfb060cfb451835af5a6bda4eb497b0d4aa9b94556dcc9994f777

Repository:
https://github.com/PlayEstate/contract-p2e-v1-core
Commit:
56c5773
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/GamePlayV2.sol
SHA3: eb3d17da283b676e954c12ded9e90638cd29134a47962471ceaa71bcb09dd18a

File: ./contracts/GamePlayV2Factory.sol
SHA3: a4317ad88efa7c17f81d29b427322f7001185b9dd444638d832329f3c8f0a288

File: ./contracts/GamePlayV2Storage.sol
SHA3: 15a642492ee66996647aae4289c07544805544d9071743b5067c34c2794ed237

Second review scope

Repository:
https://github.com/PlayEstate/contract-marketplace-v1
Commit:
9ce58c6
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/engine/NFTEngineV1.sol
SHA3: 30f0004c9700ac91664b162fa698e79f5dcbb7b126780d288a47d11a8f11afee

Repository:
https://github.com/PlayEstate/contract-p2e-v1-core
Commit:
69a12b4
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/GamePlayV2.sol
SHA3: 23db05955dd3df3d255f19810e25671eaa7b7f59bf4a67e56dc0c1d74cd95411

File: ./contracts/GamePlayV2Factory.sol
SHA3: f44da19016395f89119a1220a37828b39cd56e1377d8df6ada7ff733f688b085

File: ./contracts/GamePlayV2Storage.sol
SHA3: 7dbbb4f818c7bc67466f6ca123205a840ac6eef934fa1b553271f50e11a3b0f7

Third review scope

Repository:
https://github.com/PlayEstate/contract-marketplace-v1
Commit:
afbbf50
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/engine/NFTEngineV1.sol
SHA3: 01aaa3286f7cb5390721cf095ba43032ff66eedcea1374ecc0a01044289f319c

Repository:
https://github.com/PlayEstate/contract-p2e-v1-core
Commit:
622848a
Documentation:
Lightpaper

Readme

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/GamePlayV2.sol
SHA3: 56e242ba084eb1d99ee73c21de9ed9baace7f8e65003becc730bff9b73c7643e

File: ./contracts/GamePlayV2Factory.sol
SHA3: f44da19016395f89119a1220a37828b39cd56e1377d8df6ada7ff733f688b085

File: ./contracts/GamePlayV2Storage.sol

SHA3: b074618e90d482cd7d146ef158dca314a00180d4fc81f5de4498366ccc76147b

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

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 **8** out of **10**.

- Technical requirements are provided.
- Code is followed by NatSpec comments.
- Functional requirements are partially missed.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- Code violates Style guide.

Test coverage

Test coverage of the project is **96.5%**.

- Deployment and basic user interactions are covered with tests.
- Revert case of *_updateHighestBid* is not tested.

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: **9.4**.

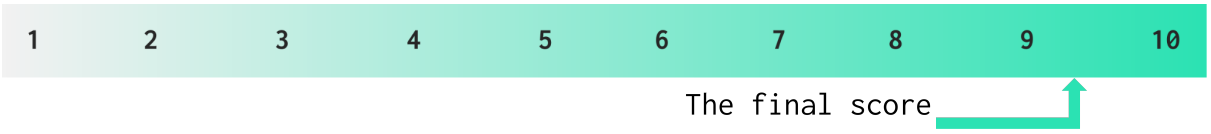

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
11 October 2022	3	2	1	1
26 October 2022	0	1	1	0
07 November 2022	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more 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.	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 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.	Passed
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 identifiers should always be used. All parameters from the signature should be used in signer recovery	Passed
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.	Passed

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.	Passed
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.	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 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

PlayEstates is a mixed-purpose system with the following contracts:

- *NFTEngine.sol* - Used to create sales & auctions and manage them effectively for sellers, buyers, and bidders.
- *GamePlayV2Factory.sol* - Factory contract to deploy and initialize a game engine contracts.
- *GamePlayV2Storage.sol* - Abstract contract to manage state variables, events, structs used in GamePlayV2 engine contract, and some base logics such as signature verification feature.
- *GamePlayV2.sol* - Game engine logic contract to manage rounds, scores, rankings, rewards.

Privileged roles

- The owner of NFTEngineV1 can change treasury.
- The owner and admin of GamePlayV2Factory can deploy GamePlayV2.
- The owner of GamePlayV2 can create round, update signer, emergencyClaim, recover wrong tokens, distribute rankings and rewards, update ranking rewards, update score, round, lock round, set up rewards rate according to rankings.

Risks

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

Findings

■■■■ Critical

1. Requirements Violation

The code should not violate the requirements provided by the Customer. A function commentary explains a supposed execution flow of this function (initializing the game engine) which is not implemented in the code.

Path: ./contracts/GamePlayV2Storage.sol:initialize()

Recommendation: Implement missing code.

Status: Fixed (Revised commit: 9ce58c6)

■■■ High

1. Unfinalized Code

Commented code parts and unfinished functions are left in code. The production code should not contain any functions or variables that are being used solely in the test environment or during the development process.

Path: ./contracts/engine/NFTEngineV1.sol:mintNFT(); ./contracts/GamePlayV2StorageV2: _matchAddresSigner.sol();

Recommendation: Finalize contracts or remove commented code parts.

Status: Fixed (Revised commit: 9ce58c6)

2. Unchecked Call Return Value

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally, the funds will not be transferred to the treasury.

Path: ./contracts/engine/NFTEngineV1.sol:buyNFT();

Recommendation: Check call return value.

Status: Fixed (Revised commit: b455f09)

■■ Medium

1. Unoptimized Loops Usage

Contracts use loops without optimization. Optimizing loops will lower Gas taxes.

Paths: ./contracts/engine/NFTEngineV1.sol: checkFeeRatesLessThanMaximum() ./contracts/GamePlayV2.sol: _calcRankingRewards();

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: b455f09)

2. Using SafeMath in Solidity ^0.8.0

Starting with Solidity ^0.8.0, SafeMath functions are built-in. Due to this, using this library is redundant.

Path: ./contracts/GamePlayV2Storage.sol; ./contracts/GamePlayV2.sol

Recommendation: Remove redundant functionality.

Status: Fixed (Revised commit: 69a12b4)

Low

1. 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.

Paths: ./contracts/engine/NFTEngineV1.sol; ./contracts/GamePlayV2Factory.sol; ./contracts/GamePlayV2Storage.sol; ./contracts/GamePlayV2.sol;

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

Status: Fixed (Revised commits: 9ce58c6, 69a12b4)

2. No Error Messages in Require Statements

Some *require* statements are missing error messages.

This makes code harder to test and debug.

Path: ./contracts/engine/NFTEngineV1.sol;

Recommendation: Add error messages to require conditions.

Status: Fixed (Revised commit: 9ce58c6)

3. Public Functions Can Be Declared External

Public functions that are never called by contract should be declared external.

This will lower Gas taxes.

Paths: ./contracts/GamePlayV2.sol: initialize(); ./contracts/GamePlayV2.sol: updateRound(); ./contracts/GamePlayV2.sol: setRewardRateMapForRanking();

Recommendation: Declare mentioned functions as external

Status: Fixed (Revised commit: 69a12b4)

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.