HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: UFO Gaming
Date: July 13th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for UFO Gaming
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Staking, Time locked
Platform	EVM
Network	Polygon
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://ufogaming.io/
Timeline	27.06.2022 - 13.07.2022
Changelog	13.07.2022 - Initial Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by UFO Gaming (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/UFOGaming51/ufo-smartcontracts/
Commit:
    95a7dfc24bfeb084844add8768f7576b7f29214c
Technical Documentation:
    Type: Whitepaper (outdated partially)
    Link

    Type: Technical description
    Link

Integration and Unit Tests: Partial
Deployed Contracts Addresses: No

Contracts:
    File: ./contracts/ethereum/Staking.sol
    SHA3: d78755c3d7997345c4c86a7d77d5d49f9baee0e68a31f98d50d286b1fb42c701

    File: ./contracts/ethereum/StakingFactory.sol
    SHA3: a1179de61b5269d5f8681ab44e48e3a215a3deb93b55a02132d2698000c17687

Second review scope
Repository:
    https://github.com/UFOGaming51/ufo-smartcontracts/
Commit:
    19e1a8bc11c6c840560a03932951199791f21a25
Technical Documentation:
    Type: Whitepaper
    Link

    Type: Technical description
    Link

Integration and Unit Tests: Partial
Deployed Contracts Addresses: No

Contracts:
    File: ./contracts/ethereum/Staking.sol
    SHA3: cff8666c52d40125becf238f19ea51140c42f1b25b018d2e581e82d8bda0b2fb

    File: ./contracts/ethereum/StakingFactory.sol
    SHA3: 5e6a75270416a5e28035eff0c28c9f3111bd7e531c8a6c63cabb29e7a493aa12

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Low	Low-level vulnerabilities are mostly related to outdated, unused, etc.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.

www.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 **10** out of **10**. Well documented with clear technical requirements.

Code quality

The total CodeQuality score is **5** out of **10**. Deployment and basic user interactions are partially covered. Negative cases coverage is missed, and interactions by several users are not tested thoroughly.

Architecture quality

The architecture quality score is **10** out of **10**. The deployment of staking pools is well implemented.
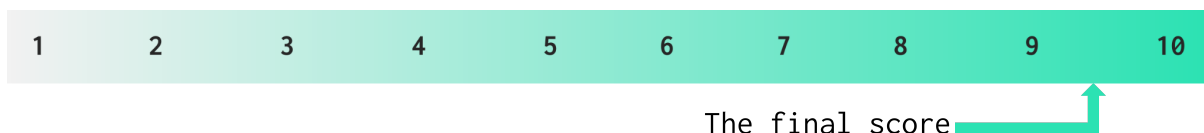
Security score

As a result of the audit, the code contains **1** low severity issue. 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: **9.5**.

1	2	3	4	5	6	7	8	9	10

The final score ➞

www.hacken.io

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.	Failed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Failed
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.	Passed
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.	Passed
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	Passed

www.hacken.io

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Failed
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	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.	Not Relevant
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.	Passed
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	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.	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.	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.	Not Relevant

System Overview

The UFO staking DAPP will allow users to earn Plasma tokens by staking UFO tokens and UFO/ETH pairs.

The files in the scope:
- **Staking.sol** - Staking contract with support for UFO or LP token staking to get Plasma rewards
- **StakingFactory.sol** - Aggregation and factory contract that creates new staking pools and tracks them

Privileged roles

- <u>Owner (StakingFactory.sol):</u> Can deploy new staking pools, has the ability to edit rewards for pools, emergency withdraw rewards
- <u>Pool (StakingFactory.sol):</u> Allows pools to update data about them in the factory registry
- <u>Owner (Staking.sol):</u> Can pause and unpause staking

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

No medium severity issues were found.

■ Low

1. State variables can be declared constant

Compared to regular state variables, the Gas costs of constant and immutable variables are much lower. Immutable variables are evaluated once at construction time, and their value is copied to all the places in the code where they are accessed.

Contract: StakingFactory

Recommendation: StakingFactory.totalBlockPerYear value is hardcoded. This variable can be declared constant.

Status: Fixed (14e06e33a70446aed49191b0de9867a1eb699051)

2. 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.

Contract: StakingFactory, Staking, Errors, IRootChainManager, IStaking, IStakingFactory

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

Status: Fixed (b9edd8ae10f909c99ca6e9a2e4ec49bb2ec45fcc)

3. Misleading variable names

Variables "_lockinBlocks", "totalBlocksPerYear", "lockinBlocks", refer to blocks when counting seconds.

Contract: Staking, StakingFactory

Recommendation: Rename the variables to represent accurate values.

Status: Reported

4. Duplicate variable "totalSecondsPerYear"

The local variable "totalSecondsPerYear" is used multiple times throughout the contracts instead of the global incorrectly named "totalBlocksPerYear" despite having the same value.

Contract: Staking, StakingFactory

Recommendation: Rename the variable "totalBlocksPerYear" and use it correctly.

Status: Fixed (3964ba74a36c7064dd968d5b9f070b8744b6fea8)

5. **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.

Paths: StakingFactory.constructor

Recommendation: Implement zero address checks on rewardToken.

Status: Fixed (19e1a8bc11c6c840560a03932951199791f21a25)

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.

UFO Gaming - Recommended Actions

Next Steps

1. Identify critical assets to scope. It is recommended to scope 4 assets:
   - https://ufogaming.io/ru

   - iOS app

   - Android app

   - Network

2. Pentest 4 assets:
   - https://ufogaming.io/ru

   - iOS app

   - Android app

   - Network

3. Set up crowdsourced security for all assets:
   - https://ufogaming.io/ru

   - iOS app

   - Android app

   - Network

www.hacken.io