HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: EthereumTowers
Date: July 04th, 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 Ethereum Towers
Approved By	Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	Staking
Platform	EVM
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://ethereumtowers.com
Timeline	16.06.2022 - 04.07.2022
Changelog	17.06.2022 - Initial Review 04.07.2022 - 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 Ethereum Towers (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/ethereumtowers/contracts
Commit:
94eb48031a02455bb3c48285ffe41fbbe3498079
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Type: Technical description
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/staking/EthereumWorldsNFTStaking.sol
SHA3: 762652eaa08dde6d058efede9819cf671c2e7a08150c0776c961ed16d4af5e10

Second review scope

Repository:
https://github.com/ethereumtowers/contracts
Commit:
2aec56b71c2be657ef1c117d85486b7bf49f0fb4
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Type: Technical description
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/staking/EthereumWorldsNFTStaking.sol
SHA3: b70e3bec3a80889cc459890548c25ad0cf8fbf5dc89f18fc9f3710a1fb393e69

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 methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional and technical requirements are provided. Whitepaper well describes the project.

Code quality

The total CodeQuality score is **9** out of **10**. Code well-covered with unit tests. Code style and naming conventions are respected. *should restrict calling stake for non-existing token* test is failing and should be fixed.

Architecture quality

The architecture quality score is **10** out of **10**. Contracts follow single responsibility principles, code is well-formatted.

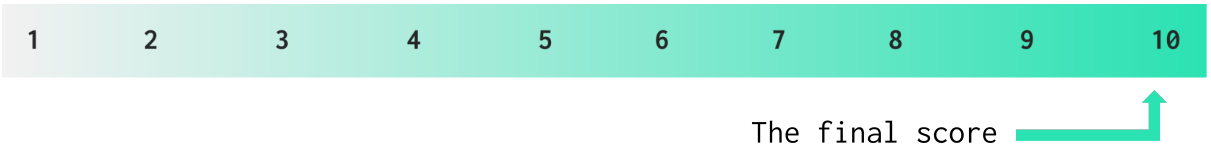
Security score

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: **9.9**.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@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.	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.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
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 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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	Passed
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	**Custom**	Transaction execution costs should not	Passed

Loops		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.	
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.	Passed
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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Ethereum Towers is a community-centric, vertical megastructure consisting of 4,388 resident-owned apartments and a variety of communal areas, set in the forthcoming Ethereum Worlds Metaverse with the following contracts:

- *EthereumWorldsNFTStaking* — a contract that rewards users for staking their NFTs. Rewards are calculated off-chain.

Privileged roles

- The owner of the *EthereumWorldsNFTStaking* contract can pause staking, unstacking, and claiming in case of emergency.
- The owner of the *EthereumWorldsNFTStaking* contract can update the service signer address.
- The owner of the *EthereumWorldsNFTStaking* contract can update the max amount of tokens in staking.
- The owner of the *EthereumWorldsNFTStaking* contract can withdraw ERC20 tokens sent to the contract.

Risks

- In case of an admin keys leak, an attacker can lock contract functionality.
- In case the contract will not be funded with world tokens - it would not be possible to claim rewards.

Findings

Critical

No critical severity issues were found.

High

Inconsistent contract state.

tokensInStake variable was calculated incorrectly, which could lead to staking more tokens to the contract than expected.

During *emergencyUnstake* contract decides which amount to unstake: requested by the user, or *maxTokensPerUnstake*, to prevent the out of Gas exception. Contract ignores which value was selected to unstake and always decreases *tokensInStake* value by *ids.length*

Contract: EthereumWorldsNFTStaking.sol

Function: emergencyUnstake

Recommendation: Use *unstakeAmount* instead of *ids.length* when decreases *tokensInStake*

Status: Fixed (2aec56b71c2be657ef1c117d85486b7bf49f0fb4)

Medium

Unused renting logic.

The contract contains unused for now renting logic. Considering that the contract is not upgradable - it is impossible to be sure that this logic would not be broken (for example, a user can unstake a token at any time, and it is unclear how it will affect renting logic).

Contract: EthereumWorldsNFTStaking.sol

Function: setRentable

Recommendation: Consider deleting this functionality or provide documentation on how it will be implemented.

Status: Fixed (2aec56b71c2be657ef1c117d85486b7bf49f0fb4)

Low

1. **Variable Shadowing.**

   Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B, which has a state variable x defined. This would result in two separate versions of x, accessed from contract A and the other

from contract B. In more complex contract systems, this condition could go unnoticed and subsequently lead to security issues.

Contracts: EthereumWorldsNFTStaking.sol

Functions:getTokensByOwner(address **owner**) -> Ownable.**owner**(), getClaimsInfo(address **owner**) -> Ownable.**owner**(), _deleteFromTokensArray(address **owner**) -> Ownable.**owner**(),

Recommendation: Consider renaming the function argument.

Status: Fixed (2aec56b71c2be657ef1c117d85486b7bf49f0fb4)

2. **Missing events arithmetic.**

   To simplify off-chain changes tracking, it is recommended to emit events when a crucial part of the contract changes.

   Contracts: EthereumWorldsNFTStaking.sol

   Functions: updateMaxTokensInStake, toggleShutdown

   Recommendation: Emit an event for critical parameter changes.

   Status: Fixed (2aec56b71c2be657ef1c117d85486b7bf49f0fb4)

3. **Zero address is allowed.**

   The new address for the service signer does not check if it is a zero address, which could be sent as a default value.

   Contracts: EthereumWorldsNFTStaking.sol

   Functions: updateServiceSigner

   Recommendation: Add check for zero address for _serviceSigner

   Status: Fixed (2aec56b71c2be657ef1c117d85486b7bf49f0fb4)

4. **Redundant SafeCast library.**

   The contract is using the SafeCast library to cast uint256 to uint32 and uint224. The usage of the library is redundant in this contract, as all variables could not be overflowed. To save Gas and simplify the code, it is recommended to remove the library.

   Contracts: EthereumWorldsNFTStaking.sol

   Recommendation: Remove SafeCast library.

   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 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.