HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer:Wixpool
Date:February 10, 2023

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 Wixpool
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	Wixpool Rate Stabilizer
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://wixpool.com/
Changelog	06.02.2022 - Initial Review 20.02.2022 - 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 Wixpool (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 review and security analysis of smart contracts in the repository:

Initial review scope

Repository	https://wixpool.com/
Commit	a3a02ae813d8a5fe1b1cf471d4eec7d5dd531e96
Contracts	File: ./contracts/WixpoolRateStabilizer.sol SHA3: 590e8c3155245224cf3a472f824d9cd1cc8743125ee34c2b612f1a05547e9613

Second review scope

Repository	https://wixpool.com/
Commit	7d09135df1106c31b49e24fd706cea2d58cf11e2
Contracts	File: ./contracts/WixpoolRateStabilizer.sol SHA3: 73e6ec6fe7003d3ebea106698cf219f9b28ee34b876996b016cd5bd721e36bd8

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 the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor gas optimization. These issues won't have a significant impact on code execution but affect code quality

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 **5** out of **10**.

- Functional requirements are provided in the Readme file. Basic requirements are provided.
- Technical description is not provided.

Code quality

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

- Code follows solidity style guidelines.
- Development environment is configured.

Test coverage

Code coverage of the project is **63.33%** (branch coverage).

- Part of the functionality is not covered.
- Multiple-user interactions are not covered.

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: **9.5**.

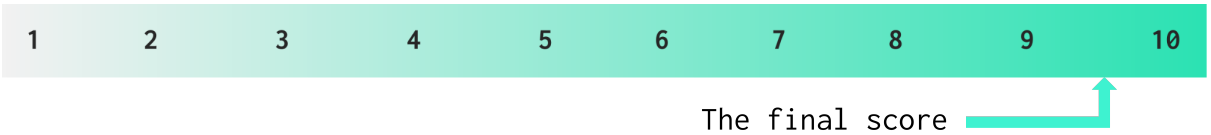

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
06 February 2023	3	2	0	0
10 February 2023	0	0	0	0

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

Checked Items

We have audited the Customers' smart contracts for commonly known and 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.	Not Relevant
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
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

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

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 EIP-712	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. EIP-712 should be followed during a signer verification.	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 or be locked on the contract.	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

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

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.	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
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 sufficient, 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, which may be changed in the future.	Passed

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

Wixpool Rate Stabilizer Overview

Wixpool Rate Stabilizer (WRS) is a multi-level yield stabilization algorithm that controls the composition of pools in plans depending on the number of users and the stability of the pool. WRS fixes the yield at a high level, preventing its decline even for "exotic" tokens with a high probability of yield decay.

The use of WRS allows to:

- use liquidity pools with a single token;
- achieve a completely stable average annual rate of return;
- fix a high yield period of more than 360 days;
- provide stable liquidity for more than 300 decentralized exchanges.

Wixpool Gate Defense Overview

To protect our users from manipulation of cryptocurrency prices and market collapse, we have introduced an advanced protection algorithm, Wixpool Gate Defense. Wixpool Gate Defense is a part of the Wixpool Rate Stabilizer root algorithm responsible for the security of the entire system.

How does Wixpool Gate Defense work

Experienced Wixpool users may have noticed that sometimes some plans are closed for investment. For example, a user cannot deposit tokens into the Ultimate plan. It's not a mistake, but the work of Wixpool Gate Defense, which protects users from the dangers of the market.

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

Findings

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

M01. Inconsistent data

_currentId value is used to assign a token id. However, 0 ID is skipped due to its increment before each minting.

Path: ./contracts/WixpoolRateStabilizer : _internalMint()

Recommendation: utilize 0 ID.

Found in: 8471531

Status: Mitigated (Initial ID 1 is a part of the requirements) **M02.**

Redundant values

The contract has a global variable called *_contractURI* and the corresponding getter for it: *contractURI()*. This value is not part of the ERC721 standard and could not be used by any of the known platforms that work with this standard.

Provided documentation also does not specify the meaning of this variable.

Path: ./contracts/WixpoolRateStabilizer.sol

Recommendation : remove the redundant value and ensure that *ERC721:tokenURI()* returns correct values that can be used to display NFT content.

Found in: 8471531

Status: Mitigated (The function can be used by OpenSea) **Low**

L01. Missing 0 value validation

The functions lack zero value validation of the *amount* parameter. **Path:**

./contracts/WixpoolRateStabilizer.sol : mintNFTs(),

./contracts/WixpoolRateStabilizer : mintNFTsOwner()

Recommendation: ensure that value is never 0. **Found**

in: 8471531

Status: Fixed (commit: f09135df) **L02.**

Best practices violation

The for loop in the function uses ids that start from 1. It goes against the commonly accepted language best practices.

Path: ./contracts/WixpoolRateStabilizer.sol : _internalMint

() **Recommendation**: utilize 0 ID.

Found in: 8471531

Status: Fixed (commit: f09135df) **L03.**

Floating Pragma

The contract uses a floating pragma. **Path:**

./contracts/WixpoolRateStabilizer.sol

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

Found in: 8471531

Status: Fixed (commit: f09135df)

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 based on best industry practices at the time of the writing 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 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, the Consultant cannot guarantee the explicit security of the audited smart contracts.