HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Optimus Ventures
Date: May 17, 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 for Optimus Ventures
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Staking; ERC721 token
Platform	EVM
Language	Solidity
Methodology	Link
Website	-
Changelog	24.03.2023 - Initial Review 19.04.2023 - Second Review 03.05.2023 - Third Review 17.05.2023 - Fourth 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 Optimus Ventures (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 includes review and security analysis of the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/OptimusVentures/HODLAudit
Commit	4b0ebda94a37dd43530d9b76b90ae4639b1d8d19
Whitepaper	-
Functional Requirements	-
Technical Requirements	-
Contracts	File: ./contracts/HODLStaking.sol SHA3: 369f3cd7224804e810fae54e7b3a416a0bbcd91bce4e8e4bd3b79c26e10b2478 File: ./contracts/HODLStakingFactory.sol SHA3: 9489dc53113ec8a2a7b103276e583c44472321ff665f2e522f78c71bfdcaf7fc File: ./contracts/HODLBase.sol SHA3: 9eebc1232a52df4ea503be9f200576145cd9bdcdfad447f1556104761428d2e2 File: ./contracts/Libraries/TransferHelper.sol SHA3: 8d3d077f06013d67e26f9c598fc2928dd78efad79e9a6ecddb313e868fd07057 File: ./contracts/Interfaces/IHODLNFTCore.sol SHA3: 3cb617ee65d586224b35e9490bbf8130f5031cea53895afc7ee6b40859bb610f File: ./contracts/Interfaces/IPair.sol SHA3: b2b7be717c8a9945fc656f6e8a41462e4b85cf2f0d6c39f7c59d173efa0881b3

Second review scope

Repository	https://github.com/OptimusVentures/HODLAudit
Commit	dfd62180e5c384192fa90423c98028528629942d
Contracts	File: contracts/HODLBase.sol SHA3: ebae5e7b72374f4bf9c1c19cb7b4e2472b16c6953d89b92179fe3c310b22a298 File: contracts/HODLStaking.sol

	SHA3: 22d8985020a40492216077b264e6c740d935c3c27a1f82984fbf209a8a95c4ba File: contracts/HODLStakingFactory.sol SHA3: 525c8094e8cbb1c8a82ebc588a442f64450817d126627eb94b8a752137bc82fa File: contracts/Interfaces/IHODLNFTCore.sol SHA3: 03fba7656b6afc18ed153d2442c90d907fe0a374ee7be01c7245d4414723b364 File: contracts/Interfaces/IHODLStaking.sol SHA3: 0847a1dbe3f4aeb00c129ce50fc66450b0a95f5668df4ddb9cfd36aabb264936 File: contracts/Interfaces/IPair.sol SHA3: 6c326a087e726ed39c19f90aba1ebaa18b6f329608b80d04ffaf62e7c92b1de5 File: contracts/Libraries/TransferHelper.sol SHA3: 8d3d077f06013d67e26f9c598fc2928dd78efad79e9a6ecddb313e868fd07057

Third review scope

Repository	https://github.com/OptimusVentures/HODLAudit
Commit	e1e35bbb162dd4d75d3377b5390b3aa76ce89f5d
Contracts	File: contracts/HODLBase.sol SHA3: ebae5e7b72374f4bf9c1c19cb7b4e2472b16c6953d89b92179fe3c310b22a298 File: contracts/HODLCoreNFT.sol SHA3: 389121b9505acf5314b461383c9b40d0ec07a1be1c7a49dd6aec39f071e071a5 File: contracts/HODLStaking.sol SHA3: 22d8985020a40492216077b264e6c740d935c3c27a1f82984fbf209a8a95c4ba File: contracts/HODLStakingFactory.sol SHA3: a28342b8309bcd9fdd4498936de28ae498c1c5197f519e217501e9533797eee4 File: contracts/Interfaces/IHODLNFTCore.sol SHA3: 03fba7656b6afc18ed153d2442c90d907fe0a374ee7be01c7245d4414723b364 File: contracts/Interfaces/IHODLStaking.sol SHA3: 0847a1dbe3f4aeb00c129ce50fc66450b0a95f5668df4ddb9cfd36aabb264936 File: contracts/Libraries/TransferHelper.sol SHA3: 8d3d077f06013d67e26f9c598fc2928dd78efad79e9a6ecddb313e868fd07057

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

Fourth review scope

Repository	https://github.com/OptimusVentures/HODLAudit
Commit	c6cd38fbea2aafbe346de04010f6dbc77aa301a0
Contracts	File: contracts/HODLBase.sol SHA3: ebae5e7b72374f4bf9c1c19cb7b4e2472b16c6953d89b92179fe3c310b22a298 File: contracts/HODLCoreNFT.sol SHA3: 203d76d196470829ab7c02cce29121c470c18c08a124765aa91c47c56ea4082d File: contracts/HODLStaking.sol SHA3: 0f5cc81e01c0e9f9c81bfe3ee77dbe4d60857de0e2d6636ca8a7a2bcd4dff26f File: contracts/HODLStakingFactory.sol SHA3: a28342b8309bcd9fdd4498936de28ae498c1c5197f519e217501e9533797eee4 File: contracts/Interfaces/IHODLNFTCore.sol SHA3: 03fba7656b6afc18ed153d2442c90d907fe0a374ee7be01c7245d4414723b364 File: contracts/Interfaces/IHODLStaking.sol SHA3: 0847a1dbe3f4aeb00c129ce50fc66450b0a95f5668df4ddb9cfd36aabb264936 File: contracts/Interfaces/IPair.sol SHA3: 6c326a087e726ed39c19f90aba1ebaa18b6f329608b80d04ffaf62e7c92b1de5

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 **7** out of **10**.

- Functional requirements are partially provided.
    - Use cases and business logic are missing.
- Technical description is provided.

Code quality

The total Code Quality score is **8** out of **10**.

- Template code patterns were found (TransferHelper).
- The development environment is configured.

Test coverage

Code coverage of the project is **94.80%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Deployment instructions are not provided.

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: **9.1**.The system users should acknowledge all the risks summed up in the risks section of the report.

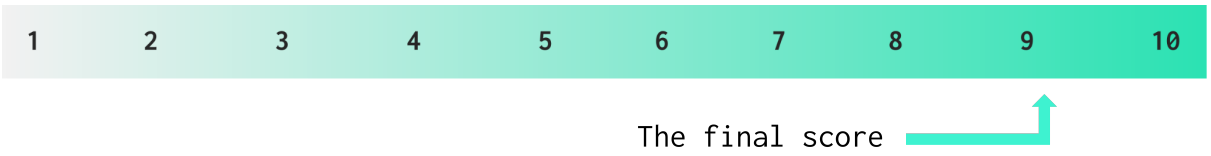

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
24 March 2023	3	6	2	0
19 April 2023	0	1	1	0

03 May 2023	0	0	1	0
17 May 2023	0	0	0	0

Risks

- There are complex fee and reward formulas used in the project. However, since no documentation is provided, only the mathematical correctness of these formulas can be checked. It is not possible to check if the formulas actually return the intended values.

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

System Overview

Optimus Ventures is a staking project that takes and rewards ERC20, which generates an ERC721 token for its users as a pinkslip. The project contains the following contracts:

- *HODLStakingFactory* – HODL ERC20 staking pools, farms and manages ownership and control over it.
- *HODLStaking* – HODL staking pool contract, to stake ERC20 tokens with a dynamic APY.
- *HODLBase* – contains structs and global variables.
- *HODLCoreNFT* - The HODLCoreNFT contract is a ERC721 token (NFT) collection representing HODL staking positions. It also provides royalty payments to a specified address.

Privileged roles

- The owner of the HODLStakingFactory contract has the authority to pause/unpause the contract. They may also set the coreNFT, pool master contract, pool creation payment token address, project fee, minimum staking period, pool creation price, and pool fee ranges.
- The onlyProjectWallet role of the HODLStaking contract is responsible for pausing/unpausing the contract, initiating the staking period after the deposit time has elapsed, and in the event of an emergency, terminating the contract and withdrawing any unearned rewards.
- The onlyOwner role of the HODLStaking contract is tasked with halting user fund deposits via the setDepositable function.
- The owner role of the HODLCoreNFT contract has the authority to set URI to any token ID.
- The minter role within the HODLCoreNFT contract is capable of minting new tokens and adding new minter addresses to the system.

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.	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

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

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
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.	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 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
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.	Not Relevant

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

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

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

Findings

Critical

No critical issues were found.

High

H01. Unverifiable Logic

The *HODLStaking.sol* and *HODLStakingFactory.sol* contracts highly depend on the *HODLCoreNFT.sol* contract, which is out of the audit scope and has some highly permissive owner privileges.

Path: ./contracts/HODLCoreNFT.sol

Recommendation: Audit the entire system and ensure that no EOA can be added to *allowedMinters* during system deployment.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: e1e35bbb162dd4d75d3377b5390b3aa76ce89f5d)

H02. Highly Permissive Role Access

The sweep function aims to drain the funds stuck in the contract without touching the staking tokens or rewards. In order to ensure that it contains the following require check; "*require(token != address(_stakingDetails.stakingToken) || token != address(_stakingDetails.rewardToken), 'Error: you can sweep the staking token');*". This check allows the owner to access the staking tokens and related rewards.

This can lead an owner to wipe out the staking rewards or staking funds.

Path: ./contracts/HODLStaking.sol: sweep()

Recommendation: Replace the || (or condition) with an && (and condition).

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

H03. Highly Permissive Role Access

The *_isApprovedOrOwner()* function has been overridden. This alteration permits any address with the minter role to transfer tokens from any other address. As a result, those with the minter role will be able to transfer tokens from user wallets.

Path: ./contracts/HODLCoreNFT.sol : _isApprovedOrOwner()

Recommendation: Minters should not have access to funds that belong to users. Implement limitations on the privileges associated with the minter role.

Found in: e1e35bbb162dd4d75d3377b5390b3aa76ce89f5d

Status: Fixed (Revised commit: c6cd38fbea2aafbe346de04010f6dbc77aa301a0)

Medium

M01. Inconsistent data - Variable is not limited

The owner of the contract can set any value for the *_feeRanges* variable as there is no lower or upper limit for it.

This may result in the unexpected deduction of high fees from users.

Path: ./contracts/HODLStakingFactory.sol : setFeeRanges()

Recommendation: Provide limitations for stored configuration values.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

M02. Denial of Service

The getUserTokens function iterates through all the tokens that belong to a user even if those tokens are related to other stakings deployed by *HODLStakingFactory*.

If the number of tokens is big enough, the user won't be able to deposit.

Path: ./contracts/HODLStaking.sol : deposit(), getUserDetails(), getUserTokens()

Recommendation: Implement a for loop limitation.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

M03. Contradiction - Missing validation

The *checkParams* modifier contains an *if* control statement that depends on the user-supplied parameters. As the *isFarming* variable is not utilized within any portion of the contracts, passing a value of

false for this variable would not have any consequences. As a result, a user could potentially bypass the *require* checks of the *checkParams* modifier by passing a value of *false* for *params.isFarming*

Path: ./contracts/HODLStakingFactory.sol : setFeeRanges()

Recommendation: If the *require* statement is dependent on user-supplied parameters, then there should be consequences for passing different parameters. Re-implement the logic of the modifier.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

M04. Tautology

The modifier includes a *totalSupply() >= 0* require statement. As *totalSupply* returns a *uint* variable, it is equal to or greater than zero in Solidity. Therefore, the use of the statement is redundant in this case.

Path: ./contracts/HODLStakingFactory.sol : checkParams()

Recommendation: Remove the redundant "*=*" operator.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: e1e35bbb162dd4d75d3377b5390b3aa76ce89f5d)

M05. Contradiction - Function Name - Functionality Mismacth

The *projectSend* function starts a new staking period after the deposit time ends. However the function's name does not reflect this functionality.

Path: ./contracts/HODLStaking.sol : projectSend()

Recommendation: Update function name.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

M06. Requirements Violations

The project takes the admin fee from the staking creator but this is not mentioned in the documentation.

The project takes the staking pool creation fee as *_optcmPrice* from the staking creator but this is not mentioned in the documentation.

The Project applies a rewards fee for the admin, but it is not mentioned in the documentation.

The Project takes a withdrawal fee from the user for the admin but is not mentioned in the documentation.

The code should not contain undocumented functionality.

Path: ./contracts/HODLStakingFactory.sol : createStaking()

Recommendation: Provide a detailed explanation of the functionality for users in the public documentation.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

Low

L01. Redundant Variable

The *StakingUser* structure utilizes the *StakingNFT* structure, which includes an *isOwner* variable. While stakers also possess the *isOwner* variable, it remains consistently *false* for them. The inclusion of the *isOwner* variable within the *StakingUser* structure is redundant, and its redundant storage could result in unnecessary gas consumption.

Path: ./contracts/HODLBase.sol

Recommendation: Remove the redundant variable from the *StakingUser* structure.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

L02. Missing Zero Address Validation

Address parameters are used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path: ./contracts/HODLStakingFactory.sol: setMasterHodl()

Recommendation: Implement zero address checks.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

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

L03. Missing Zero Check Before Transfer

The *refundAmount* variable can be equal to zero in which case the function will perform zero token transfers.

This can lead to unnecessary gas consumption

Path: ./contracts/HODLStaking.sol: _endContractAfter()

Recommendation: Implement a check to prevent zero token transfers.

Found in: 4b0ebda94a37dd43530d9b76b90ae4639b1d8d19

Status: Fixed (Revised commit: dfd62180e5c384192fa90423c98028528629942d)

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.