HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Zharta
Date: 29 September, 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 Zharta
Approved By	Paul Fomichov | Lead SC Auditor at Hacken OÜ
Tags	ERC20 and ERC721 token; NFT Renting
Platform	EVM
Language	Vyper
Methodology	Link
Website	https://www.zharta.io/
Changelog	08.09.2023 - Initial Review 29.09.2023 - 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 Zharta (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.

System Overview

Zharta is a protocol that allows to rent game assets(NFTs) by depositing them into vaults and securely managing upfront payments and rental durations with the following contracts:

- Renting – a contract that defines a protocol for managing NFT (game asset) rentals. It includes interface for interacting with vault. Users can create vaults and deposit game assets, set listing prices, start and close rentals, claim rewards, and withdraw assets. For each NFT id, a specific vault contract is created within the Renting contract. Exchanged NFTs also use the previously created Vault contract by resetting the variables.
- Vault – a contract to store one specific NFT for renting purposes. Vault's functions can be called only by the Renting contract.

Privileged roles

- The owner of Vault contract which is Renting contract can:
  - set a listing price
  - claim the rewards
  - withdraw the deposited NFT

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**.

- Functional requirements are partially missed:
  - Overall system requirements are provided.
- Technical description is inadequate:
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is provided but only in limited capacity.

Code quality

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

- The development environment is configured.
- Style guide violation and some best practice violations were detected.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and possible user interactions are covered with tests.

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

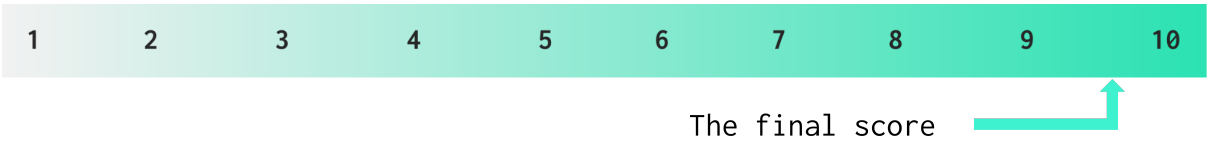

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
7 September 2023	6	3	2	1
29 September 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	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Vyper compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Vyper Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	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	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	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	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	
Flashloan Attack	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. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	

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

Token Supply Manipulation	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	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	Style guides and best practices should be followed.	Failed	I02
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	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	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	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	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

Critical

C01. Access Control Violation

Impact	High
Likelihood	High

The 'initialise' function within the Vault contract currently lacks proper access controls, allowing any user to invoke it. This vulnerability poses a significant risk, as when the first NFT renter, creator of corresponded Vault contract, withdraws their NFT from the Vault contract, the 'is_initialised' flag is set to 'False'. In an event that the Vault contract becomes empty, this design flaw creates an exploitable scenario. Malicious actors can exploit this vulnerability by invoking the 'initialise' function, effectively blocking any future rental or marketability of the associated non-fungible token.

Consequently, exchanged NFT owners will not be able to deposit their NFTs or utilize this rental market, thus undermining its intended functionality and violate reusable Vault contract feature.

Path: ./contracts/Vault.vy : initialise()

Recommendation: This function should only be callable by the NFT owner and exclusively from the 'Renting.vy' contract.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Fixed (Revised commit: efbdc3b)

High

No high severity issues were found.

Medium

M01. Unchecked Return Value/ Violation of Best Practices

Impact	High
Likelihood	Low

The functions do not use the SafeERC20 library for checking the result of ERC20 token transfers. Tokens may not follow the ERC20 standard and return false in case of transfer failure or not returning any value at all.

Path: ./contracts/Vault.vy: start_rental(), close_rental(), claim(), withdraw()

Recommendation: Check the result of the transfers.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Fixed (Revised commit: efbdc3b)

M02. Data Consistency

Impact	Medium
Likelihood	Medium

The Vault smart contract involves critical calculations related to token amounts. All values are multiplied and divided directly without loss of precision protection.

This leads to the fact that smart contract users may receive an inaccurate number of tokens, since the deviation as a result of such calculations is significant.

Path: ./contracts/Vault.vy : _compute_rental_amount(), _is_within_duration_range();

Recommendation: Implement precision lost protection.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Mitigated (The Client stated that the current calculation is taken into account and precision is optimal for the system.) (Revised commit: efbdc3b)

Low

L01. CEI Pattern Violation

Impact	Low
Likelihood	Medium

In the *start_rental* function, a CEI pattern violation has been detected, although it doesn't immediately present a reentrancy risk. The contract's sequence involves conducting checks after making changes to the state or interacting with external entities, contrary to established best practices. To resolve this issue, it is advisable to refactor the affected code to conform to the CEI pattern, thereby enhancing code readability and alignment with recognized coding standards.

Path: ./contracts/Vault.vy : start_rental();

Recommendation: Follow the correct CEI pattern and always follow best practices.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Fixed (Revised commit: efbdc3b)

L02. Inconsistent Data

Impact	Low
Likelihood	Medium

The function set_listing_price() sets three state variables at the same time. This behavior is undesirable as it is not in line with best practices and may have security risks.

Path: ./contracts/Vault.vy : set_listing_price();

Recommendation: Separate the function so that a separate function is responsible for each logical part.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Mitigated (The client explained that the data structure should always maintain consistency, with parameters that are meant to be used together.) (Revised commit: efbdc3b)

Informational

I01. Redundant Declaration

The "__init__" function in the codebase currently lacks any meaningful functionality. As a result, declaring it in the contract is redundant and introduces unnecessary gas consumption. This issue not only affects the efficiency of the contract but also makes the code less concise.

Path: ./contracts/Vault.vy: __init__()

Recommendation: Remove the redundant function.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Fixed (Revised commit: efbdc3b)

I02. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Vyper programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

Order of functions should follow the instructions in the official guideline.

The suggested Maximum line length should be 100.

All code must conform to the PEP 8 style guide, especially indentation in Code Lay-out section.

Paths: ./contracts/Renting.vy
./contracts/Vault.vy

Recommendation: Follow the official Vyper style guide.
https://docs.vyperlang.org/en/stable/style-guide.html

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Reported (Revised commit: efbdc3b)

I03. Missing Event Indexes

Use indexed events to keep track of a smart contract's activity after it is deployed, which is helpful in reducing overall Gas.

Path:
./contracts/Renting.vy : VaultsCreated, NftsDeposited, ListingsCancelled, RentalStarted, RentalClosed, RewardsClaimed;

Recommendation: Rearrange the state variables for more effective utilization of smart contract storage.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Reported (Revised commit: efbdc3b)

I04. State Variables Can Be Declared Immutable

Variable`s *vault_impl_addr* value is set in the constructor. This variable can be declared immutable

Path:
./contracts/Renting.vy : vault_impl_addr

Recommendation: Declare mentioned variables as immutable.

Found in: ec4d00d6ac24128bdf9f42c650ae19e1815adc23

Status: Fixed (Revised commit: efbdc3b)

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.

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

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.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

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

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/Zharta/lotm-renting-protocol-v1
Commit	ec4d00d6ac24128bdf9f42c650ae19e1815adc23
Whitepaper	-
Requirements	Link
Technical Requirements	-
Contracts	File: contracts/Renting.vy SHA3: d361bf2ff68effe5bee0f05840a5dc395ce6043e30d2b4a5e1155ceed042042f File: contracts/Vault.vy SHA3: 345e6c0ac4293bad20548b25b5b97d54eb5bb43cf90b49195f0dd32c57385ee9 File: contracts/auxiliary/ERC20.vy SHA3: a2cd688c46f8d20a3585d4e9b3bb34580ce97f88e8ede1b86687dc7e1de7c53a File: contracts/auxiliary/ERC721.vy SHA3: f86c1a645374567f72a957a634399be5d3ace4cce7a75f9f0067656223b0e2d3

Second review scope

Repository	https://github.com/Zharta/lotm-renting-protocol-v1
Commit	efbdc3b7bb55657aeb1c0eb3d79f6fcb658f4d09
Whitepaper	-
Requirements	Link
Technical Requirements	https://github.com/Zharta/lotm-renting-protocol-v1/blob/main/README.md
Contracts	File: ./contracts/Renting.vy SHA3: ff66e961a04f33878e1555dd6d6191d92bbd15ea68ee4fd50499f5360a287e23 File: ./contracts/Vault.vy SHA3: f919c7a56724d57d31e3ce2ed8f6f2d8531d9bf9a928c686c3b3785c284be198