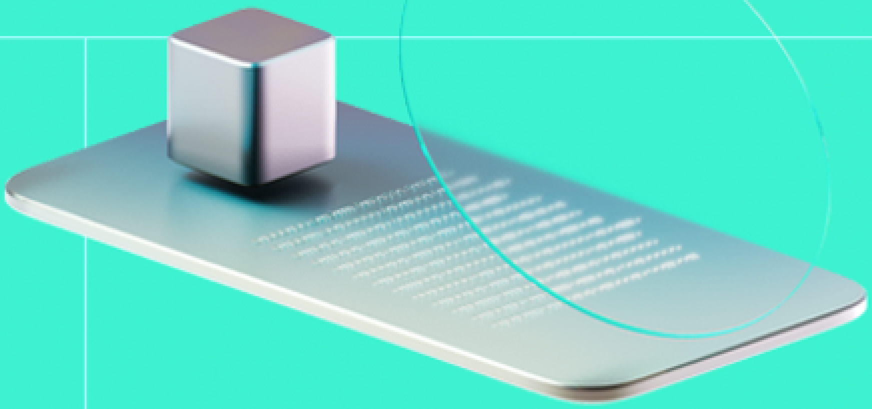
Smart Contract Code Review And Security Analysis Report



Customer: Zharta

Date: 18/03/2024

We express our gratitude to the Zharta team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Zharta revolutionizes the NFT landscape by offering instant NFT loans and staking backed by NFTs, ensuring seamless liquidity without the hassle of traditional lending processes, while providing unparalleled security and benefits such as fixed APR.

Platform: EVM

Language: Vyper

Tags: NFT Lending

Timeline: 27/02/2024 - 04/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Zharta/lotm-renting-protocol-v1
Commit	c746cea

Audit Summary

10/10	10/10	99%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

0	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	0

Document

Name	Smart Contract Code Review and Security Analysis Report for Zharta
Audited By	Turgay Arda Usman, Seher Saylik
Approved By	Grzegorz Trawinski
Website	https://www.zharta.io
Changelog - Preliminary Report	06/04/2024
Changelog - Final Report	18/04/2024

Table of Contents

System Overview

Zharta is an NFT renting and staking platform with the following contracts:

RentingERC721V3 — a simple non-fungible token contract that enables the conversion of deposited NFTs into new ERC721 tokens. It facilitates the minting and burning of tokens corresponding to deposited NFTs by the renting contract. Mint and burn functions can only be called by the renting contract address.

RentingV3 — a contract that facilitates the delegation and management of NFT rentals, offering functions to delegate NFTs to wallets, start, extend, and close rentals, and claim rewards. Users can deposit NFTs into vaults, revoke listings, and stake for their NFTs, while also being able to withdraw NFTs only if it is not in an active rental and claim rewards accrued from rentals.

- Once rental context data, such as price and maximum duration, is signed by both the owner and protocol admin, users can rent the NFT. They can do so by providing the required signatures and specifying the rental duration.
- Renters are also permitted to cancel the rental at any time, albeit with the requirement to pay for at least the minimum duration of the rented NFT. This ensures that there is a fair compensation for the time the NFT was intended to be rented, even if the rental is terminated prematurely.
- The rental price, or APR (Annual Percentage Rate), for an NFT is established at the beginning of the rental period and remains fixed throughout the rental duration. This means that renters are aware of the price they will pay upfront and can plan accordingly, without the risk of unexpected price fluctuations during the rental period.

VaultV3 — In the system, for each deposited NFT, a VaultV3 contract is created where the NFT is stored, and staking management for the specific NFT is handled within the Vault contract. This setup ensures that each NFT has its own dedicated storage and staking functionalities, maintaining a clear separation of concerns and facilitating efficient management of assets and associated staking activities.

Privileged roles

- The authorities of the RentingV3 contract"s protocol admin:
  - Claiming the accrued protocol fees
  - Setting the protocol fee
  - Changing the protocol wallet address
  - Pausing/unpausing the contract
  - Changing the admin role

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

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

- Functional requirements are provided
- Technical description is provided.

Code quality

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

- The development environment is configured.
- The code follows best practices.

Test coverage

Code coverage of the project is **99%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Tests are well written and the interactions by several users are tested thoroughly.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **0** low severity issues, leading to a security score of **10** out of **10**. Following remediation, all identified issues have been resolved.

All identified issues are detailed in the "Findings" section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- If a renter desires to cancel the rental agreement before the minimum expiration date, they are obligated to pay the rental fee until the minimum expiration date specified in the contract, rather than until the current blockchain block timestamp.
- The `RentingV3.vy` enables the owner to set platform fee via the `set_protocol_fee()` function. This means that the owner can submit a transaction to alter this value concurrently. By allocating a higher gas limit to their transaction, the owner can ensure it is processed faster, resulting in the user receiving a different amount of tokens than anticipated. This situation allows the owner to engage in front-running, exploiting their ability to preempt user transactions and alter exchange outcomes to their advantage
- This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with out-of-scope contracts are presumed to be correct and are not examined in this audit. We want to highlight that Interactions with contracts outside the specified scope, such as:
  - 3rd-party/ApeCoinStaking.sol
  - 3rd-party/HotWallet.sol
  - 3rd-party/HotWallet2.sol **have not been verified or assessed as part of this report.**

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.
Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with external contracts. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.

This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

Findings

Vulnerability Details

Observation Details

F-2024-1195 - Unused Event In RentingV3 - Info

Description: The `RentalChanged` event is declared within the contract's codebase, however, it is not utilized anywhere in the contract logic, rendering it redundant and potentially misleading readers.
Resolving this issue necessitates either integrating the RentalChanged event into pertinent contract functionality or removing it entirely to optimize gas usage and enhance code clarity.

Assets:

- RentingV3.vy [https://github.com/Zharta/lotm-renting-protocol-v1/tree/draft/v3-design]

Status: Fixed

Recommendations

Recommendation: Remove the unused event from the contract.

Remediation (Revised commit: 8b3218f): The Zharta team removed the unused event from the contract.

F-2024-1196 - Missing Event Logging - Info

Description: The functions `claim_token_ownership()` and `set_paused()` fail to log important state updates, hindering off-chain tracking of important contract events. Proper event logging is crucial for off-chain analysis and auditing, ensuring comprehensive oversight and accountability in contract operations.

Assets:

- RentingV3.vy [https://github.com/Zharta/lotm-renting-protocol-v1/tree/draft/v3-design]

Status: Fixed

Recommendations

Recommendation: Log the required events for critical state updates.

Remediation (Revised commit: 0261f02): The Zharta team introduced the missing events for the given functions.

F-2024-1311 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

Function `init()` is lack of missing zero address validation.

Assets:

- VaultV3.vy [https://github.com/Zharta/lotm-renting-protocol-v1/tree/draft/v3-design]

Status: Fixed

Recommendations

Recommendation:

Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation (Revised commit: cf84a21): The Zharta team introduced zero address checks for the `init()` function.

F-2024-1312 - Checks-Effects-Interactions Pattern Violation - Info

Description:

State variables are updated after the external calls to the vault contract that sends a call to the msg.sender to check if the msg.sender(contract) can accept NFTs.

The current implementation do not allow `onERC721Received()` function in possible malicious contract to trigger any external calls or to modify any state variables since its type is declared as “view” in the `RentingERC721` contract interface. However, it’s always recommended to use a reentrancy guard for possible future risks.

It is always best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

```
@external
def withdraw(token_contexts: DynArray[TokenContext, 32]):


"""
@notice Withdraw multiple NFTs and claim rewards
@dev Iterates over token contexts to withdraw NFTs from their vaults
and claim any unclaimed rewards, while also burning the matching ERC
721 renting token.
@param token_contexts An array of token contexts, each containing th
e vault state for an NFT.
"""



withdrawal_log: DynArray[WithdrawalLog, 32] = empty(DynArray[Withdra
walLog, 32])
tokens: DynArray[TokenAndWallet, 32] = empty(DynArray[TokenAndWallet
, 32])
total_rewards: uint256 = 0


for token_context in token_contexts:
assert self._is_context_valid(token_context), "invalid context"
assert not self._is_rental_active(token_context.active_rental), "act
ive rental"
token_owner: address = renting_erc721.owner_of(token_context.token_i
d)
if token_owner != empty(address):
assert msg.sender == token_owner, "not owner"
else:
assert msg.sender == token_context.nft_owner, "not owner"


vault: IVault = self._get_vault(token_context.token_id)


self._consolidate_claims(token_context.token_id, token_context.nft_o
wner, token_context.active_rental, False)


self._clear_token_state(token_context.token_id)


tokens.append(TokenAndWallet({
token_id: token_context.token_id,
wallet: token_context.nft_owner
}))


vault.withdraw(token_context.token_id, msg.sender)
self.listing_revocations[token_context.token_id] = block.timestamp
```

```
withdrawal_log.append(WithdrawalLog({
vault: vault.address,
token_id: token_context.token_id,
}))


renting_erc721.burn(tokens)


rewards_to_claim: uint256 = self.unclaimed_rewards[msg.sender]


# transfer reward to nft owner
if rewards_to_claim > 0:
self._transfer_payment_token(msg.sender, rewards_to_claim)
self.unclaimed_rewards[msg.sender] = 0


log NftsWithdrawn(
msg.sender,
nft_contract_addr,
rewards_to_claim,
withdrawal_log
)
```

Assets:

- RentingV3.vy [https://github.com/Zharta/lotm-renting-protocol-v1/tree/draft/v3-design]

Status: Fixed

Recommendations

Recommendation: Follow the checks-effects-interactions pattern when interacting with external contracts and implement reentrancy lock to safeguard the function.

Remediation(Revised commit: e15a6be): The Zharta team changed the order of operations, which involves making the payment transfer and updating the claimed rewards

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.

Risk Statement

This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with out-of-scope contracts are presumed to be correct and are not examined in this audit. We want to highlight that Interactions with contracts outside the specified scope, such as:

- 3rd-party/ApeCoinStaking.sol
- 3rd-party/HotWallet.sol
- 3rd-party/HotWallet2.sol **have not been verified or assessed as part of this report.**

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.
Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with external contracts. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.
This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
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 will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

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

Scope Details

Repository	https://github.com/Zharta/lotm-renting-proto
Commit	e15a6be05d88b651fed6d49793405c5e3a45d0a7
Whitepaper	https://zharta.gitbook.io/zharta-welcome-kit/overview/introducing-zharta
Requirements	https://github.com/Zharta/lotm-renting-protocol-v1/pull/69
Technical Requirements	https://zharta.gitbook.io/zharta-welcome-kit/overview/introducing-zharta

Contracts in Scope

contracts/RentingERC721V3.vy

contracts/RentingV3.vy

contracts/VaultV3.vy