Audit Report

NOIZD – Ethereum contracts

October 13, 2021

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of this Report

Oak Security has been engaged by NOIZD to perform a security audit of the NOIZD smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/noizd-dev/backend/tree/dev/hardhat

Commit hash: `6c328e2ac537788331238163efb0eafaaf896e8b`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The smart contracts implement an ERC-721-compatible NFT and an admin controlled staking and minting contract, that allows users to stake ETH (treated as bids) and has an admin role which can trigger token minting for stakers. ETH can be withdrawn with a timelock.

The admin role has the privileges to do the following:

- Mint tokens.
- Complete a user bid operation by minting and converting the stake.
- Pause the contract.
- Refund users' funds explicitly.
- Transfer ownership of the token contract.

The admin role cannot steal the users' funds or block them, meaning that a compromised key cannot lead to users losing staked funds.

How to read this Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged** or **Resolved**. Informational notes do not have a status, since we consider them optional recommendations.

Note, that audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note, that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Summary of Findings

No	Description	Severity	Status
1	NOIZDStaking.sol: Instant withdrawal signatures can be replayed to bypass freeze time in some cases	Critical	Resolved
2	NOIZDStaking.sol: Refund could be blocked by a malicious or faulty receiver	Minor	Acknowledged
3	NOIZDNFT.sol: Unreachable conditional statement	Informational	Acknowledged
4	Outdated OpenZepplin release used	Informational	Resolved

Code Quality Criteria

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Detailed Findings

1. `NOIZDStaking.sol`: Instant withdrawal signatures can be replayed to bypass freeze time in some cases

Severity: Critical

Nonces are used as a type of withdrawal id to make instant withdrawal permissions unique, which is verified in function `_withdrawInstant`. However, nonces are kept on a per-user basis, meaning that a user could re-use an admin generated signature intended for another user currently on the same nonce.

Recommendation

Consider using a globally unique nonce or encode the staker's address in the signature as well.

Status: Resolved

2. `NOIZDStaking.sol`: `Refund` could be blocked by a malicious or faulty receiver

Severity: Minor

The transaction reverts if a single Ether transfer fails (line 388), allowing a malicious or faulty receiver to block the refund operation. This is mitigated by the fact that the built-in pagination allows the admin caller to avoid such malicious or faulty receivers.

Recommendation

Consider using pull over push refunds only, i.e. marking stakes as refundable and allowing users to claim them. Alternatively, not requiring the success of the transfer operation is an option to mitigate the issue.

Status: Acknowledged

The team mitigated the issue by not requiring success. However, the refund could still be grieved by a single receiver spending all the gas. There is no real incentive for such an attack and the paging of the iteration allows recovering from such a case. The team is aware of this potential shortcoming but has made the trade-off of using push-based refunds to create a better user experience.

3. `NOIZDNFT.sol`: Unreachable conditional statement

Severity: Informational

The following code in function `tokenURI` in line 63 will always result in the execution of the first branch:

```
return bytes(baseURI).length > 0 ? string(abi.encodePacked(baseURI, _tokenURIs[tokenId])) : "";
```

The reason for this is that `baseURI` is read from the hardcoded value `"ipfs://"`.

Recommendation

Consider removing the conditional statement and returning the non-empty string.

Status: Acknowledged

The team decided to leave this statement as is, in order to deviate from the original OpenZeppelin implementation as little as possible.

4. Outdated OpenZeppelin release used

Severity: Informational

The codebase imports a relatively recent version of the OpenZeppelin smart contract library. However, there have been recent security releases that fix issues, some of which are related to ECSDA signing. These issues seem not to apply in the present use case of the contracts. However, we recommend using the latest security release, particularly, since the draft EIP-712 implementation used is updated frequently.

Recommendation

Consider updating the OpenZeppelin release used.

Status: Resolved