Security Audit Report

ICN Passport Token

v1.0

December 18, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

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

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Impossible Cloud Network Foundation to perform a security audit of ICN Passport Token.

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 behavior.
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 target:

Repository	https://github.com/ICN-Protocol/icn-passport-smart-contract
Commit	`7cee3ededff8e1c953ab71036b3f34c71830d118`
Scope	All contracts were in scope.
Fixes verified at commit	`114512b8feba64132a41ed18e3a355f17f1660bc` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

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 Impossible Cloud Network (ICN) is transforming cloud computing by building the world's first open cloud platform. By combining the power of cutting-edge blockchain technology with traditional cloud services, the protocol ensures unmatched flexibility and performance for all network participants.

Thereby, the NFT smart contract ICN Passport can be used to join the ICN either as a Guardian Oracle Node or for staking purposes.

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**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving 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 in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Low-Medium**	-
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium**	-
Test coverage	**High**	`forge coverage` reports `100%` coverage

Summary of Findings

No	Description	Severity	Status
1	Potential for stuck tokens when mint receiver is a non-compliant ERC721 contract	Minor	Resolved
2	A batch transfer can be subject to DoS due to a malicious receiver	Minor	Acknowledged
3	Inconsistent revert pattern in case of disabled transfers	Minor	Resolved
4	Inaccurate date computation and lack of localization	Informational	Resolved
5	Incorrect expiration information if the contract is not active	Informational	Resolved
6	The contract's admin can self-renounce their role	Informational	Acknowledged
7	Miscellaneous comments	Informational	Resolved

Detailed Findings

1. Potential for stuck tokens when mint receiver is a non-compliant ERC721 contract

Severity: Minor

In `src/ICNPassport.sol:114-131`, the `batchMint` function uses the `_mint` method to create tokens and assign them to specified addresses.

However, `_mint` does not verify whether the recipient address is a contract capable of handling ERC721 tokens.

If tokens are minted to a contract that does not implement the ERC721Receiver interface, the recipient will be unable to process or retrieve the tokens, potentially resulting in them being permanently stuck.

Recommendation

We recommend replacing `_mint` with `_safeMint` in the `batchMint` function.

The `_safeMint` method ensures that tokens are minted only to addresses that are either externally owned accounts (EOAs) or contracts capable of handling ERC721 tokens by implementing the `ERC721Receiver` interface.

Status: Resolved

2. A batch transfer can be subject to DoS due to a malicious receiver

Severity: Minor

The `ICNPassport` token contract's `batchSafeTransferFrom` method, defined in `src/ICNPassport.sol:146-160`, relies on the `safeTransferFrom` method for each transfer of the whole batch. This method performs an underlying `onERC721Received` acceptance check which reverts in case the receiver does not support ERC-721 tokens, or if it is purposefully reverted by a malicious receiver.

Consequently, one failing receiver is sufficient to revert a whole batch transfer effectively causing a temporary denial of service.

Recommendation

We recommend implementing a boolean switch and a try-catch pattern which allows the user to decide whether to revert on failing transfers or skip them.

Status: Acknowledged

The ICN Team recognizes that batch transfers could be vulnerable to DoS attacks due to a malicious recipient. However, the team has decided against duplicating OpenZeppelin's implementation to avoid taking on responsibility for maintaining that specific code.

3. Inconsistent revert pattern in case of disabled transfers

Severity: Minor

The `ICNPassport` token contract's `batchSafeTransferFrom` method, defined in `src/ICNPassport.sol:146-160`, checks for the `transfersDisabled` state within the `transferAllowed` modifier, i.e. it immediately reverts in case transfers are disabled.

However, the `ICNPassport` token contract also inherits the public methods `transferFrom` and `safeTransferFrom` from the `ERC721Upgradeable` base contact. These methods are not overridden and therefore do not have the `transferAllowed` modifier. In case a user attempts a transfer using these methods, the revert will happen at a later point within the overridden `_update` method which also performs the `transfersDisabled` check.

Consequently, the current revert pattern in case of disabled transfers is inconsistent and further leads to multiple superfluous `transfersDisabled` checks in the `batchSafeTransferFrom` method due to the underlying `_update` method.

Recommendation

We recommend the following mutually exclusive mitigation measures:

1. Remove the `transferAllowed` modifier from the `batchSafeTransferFrom` method and purely rely on the `transfersDisabled` check in the underlying `_update` method.
2. Or, override the `transferFrom` and `safeTransferFrom` methods and add the `transferAllowed` modifier. This allows to reconsider removing the overridden `_update` method.

Status: Resolved

4. Inaccurate date computation and lack of localization

Severity: Informational

In `src/libraries/DateTime.sol:17-18`, the date computation truncates the last day by removing any remaining hours without any rounding strategy. As a result, even hours as late as 23:59 are truncated.

Additionally, the computation assumes the UTC timezone without accounting for the user's localization.

Recommendation

We recommend storing only the timestamp within the smart contract and delegating all date-time formatting and localization to the user interface.

Status: Resolved

5. Incorrect expiration information if the contract is not active

Severity: Informational

In `src/ICNPassport.sol:166-168`, the `isExpired` function determines whether a token is expired based on its activation time and duration.

However, if the contract is not yet activated, the `getDurationTime` function returns `0`, and when combined with the `activationTime`, which is also `0`, the result is always less than the current block timestamp.

This causes the function to incorrectly indicate that the token has expired, even though the contract has not been activated. Such behavior misleads users and could disrupt the token's intended lifecycle.

Recommendation

We recommend modifying the `isExpired` function to include a preliminary validation check that verifies whether the contract is activated. If the contract is not activated, the function should return an explicit error or status message.

Status: Resolved

6. The contract's admin can self-renounce their role

Severity: Informational

The `ICNPassport` token contract inherits the public methods `revokeRole` and `renounceRole` from the `AccessControlUpgradeable` base contact.

However, these methods also facilitate (accidental) self-renouncing of the `DEFAULT_ADMIN_ROLE` which can be detrimental to the token contract's functionality in case the contract's other roles were not properly set up before.

Recommendation

We recommend reevaluating this risk and overriding the `revokeRole` and `renounceRole` methods to prevent self-renouncing of the admin if deemed necessary for the given application.

Status: Acknowledged

The ICN Team acknowledges that the contract's administrator has the ability to renounce their role. To avoid the complexities of maintaining a copied implementation from OpenZeppelin, the team has opted not to duplicate their code. Instead, ICN plans to use multi-signature wallets for each contract role, providing an added layer of security and minimizing the risk of human error.

7. Miscellaneous comments

Severity: Informational

Miscellaneous recommendations can be found below.

Recommendation

The following are some recommendations to improve the overall code quality and readability:

- The `SECONDS_PER_DAY` constant in `src/libraries/DateTime.sol:5` could be simplified using Solidity's time units, e.g. `SECONDS_PER_DAY = 1 days`.
- In `src/ICNPassport.sol:219-224`, we recommend considering an explicit way to render expired NFTs where `years_ == months_ == days_ == 0`.
- The JSON string, defined in `src/ICNPassport.sol:232-236`, does not exactly match the ERC721 Metadata JSON Schema. We recommend revising the JSON string to avoid potential compatibility issues if deemed necessary for the given application.

Status: Resolved