0xCommit

Security Assessment Report

KCash

KCash Token Audit

Version: Final

Date: 23 Apr 2024

Table of Contents

License

THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION-NODERIVATIVES 4.0 INTERNATIONAL 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.

COPYRIGHT OF THIS REPORT REMAINS WITH THE AUTHOR.

Introduction

Purpose of this report

0xCommit has been engaged by **IndiGG** to perform a security audit of several Token contract components.

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).

Codebases Submitted for the Audit

The audit has been performed on the following commits:

Github Link: https://github.com/indi-gg/kcash_v2_contract

Version	Commit hash
Initial	b0edfb744ac48957e76e5edeafea022a1be4b710
Final	

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.
High	An attacker can successfully execute an attack that clearly results in operational issues for the service. This also includes any value loss of unclaimed funds permanently or temporary.
Medium	The service may be susceptible to an attacker carrying out an unintentional action, which could potentially disrupt its operation. Nonetheless, certain limitations exist that make it difficult for the attack to be successful.
Low	The service may be vulnerable to an attacker executing an unintended action, but the impact of the action is negligible or the likelihood of the attack succeeding is very low and there is no loss of value.
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

The status of an issue can be one of the following: **Pending**, **Acknowledged**, 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.

Overview

Methodology

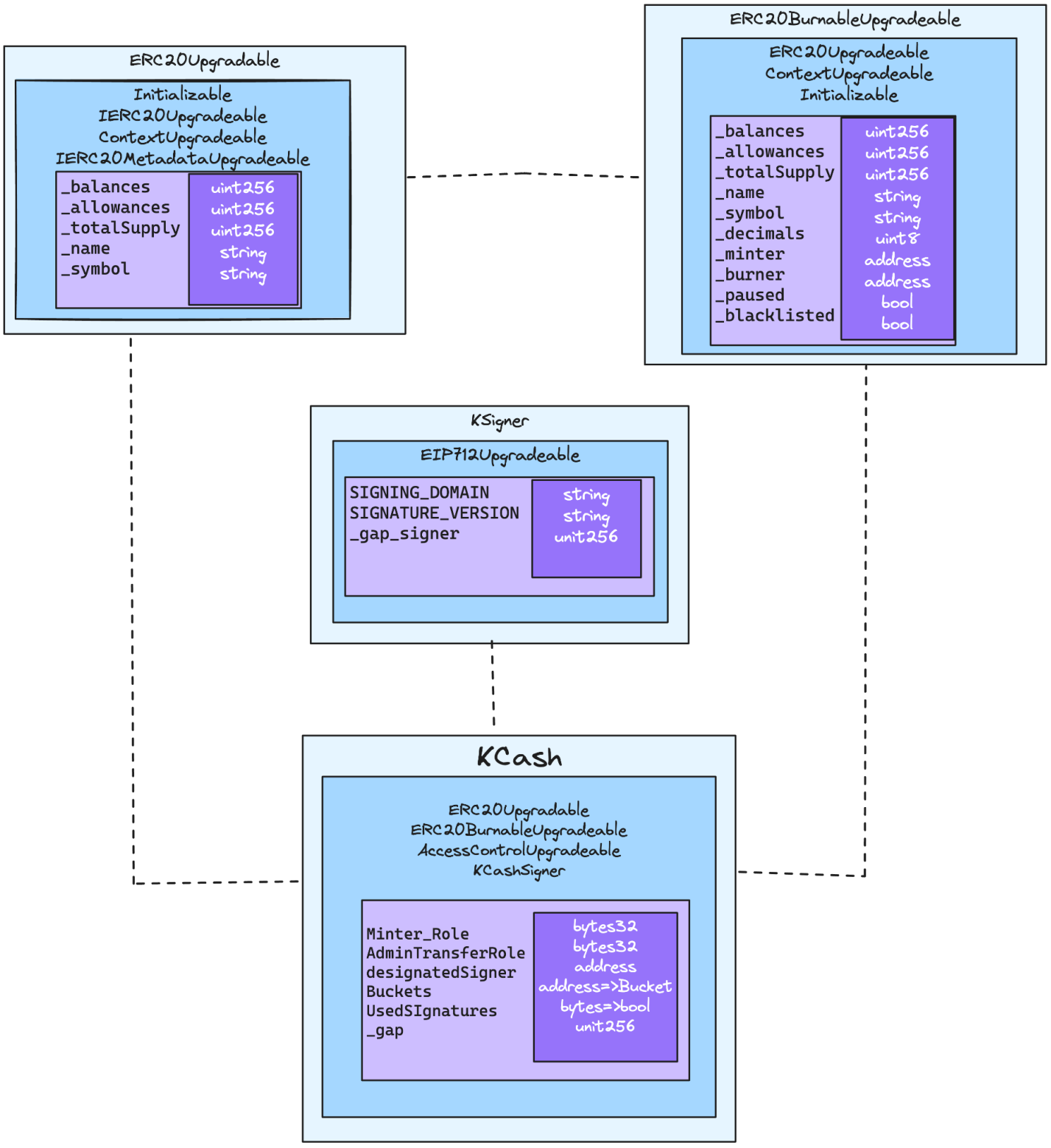
The audit has been performed in the following steps:

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
    d. Access Control Issues
    e. Boundary Analysis
4. Report preparation

Functionality Overview

Kcash is a token contract for the IndiGG ecosystem. It has a bucket token balance in addition to the traditional token balance. The bucket comprises the three reward variables.

Storage Diagram



Summary of Findings

Sr. No.	Description	Severity	Status
1	Burn Functions do not reduce the balance in reward buckets.	High	Resolved
2	adminTransferWithSignature lacks access control	Low	Acknowledged
3	Missing events	Informatio...	Acknowledged

Detailed Findings

1. Burn Functions do not reduce the balance in reward buckets

Severity: High

Description

The burn() and burnFrom() functions reduce the amount solely from the balances mapping, leaving the rewards unchanged. It violates the invariant:

balance[user] = reward1 + reward2 + reward3

Furthermore, this oversight could result in the reward buckets variable overflowing. This is because the addition of rewards within the mint() function occurs in an unchecked block.

Remediation

Refactor burn() and burnFrom() functions where they will update reward buckets along with the balance mapping.

Status

Resolved

Burn functions were restricted in the contract

2. adminTransferWithSignature lacks access control

Severity: Low

Description

If a user gets the special admin signature, they can execute the adminTransferWithSignature() function. As there is no access control over this function.

Remediation

Consider implementing appropriate access control on this function.

Status

Acknowledged

3. Missing events

Severity: Informational

Description

Events are important to track configuration changes off chain. Events are missing which prevent the intended data from being observed easily by off-chain interfaces.

Remediation

Add relevant events and emit where required.

Status

Acknowledged