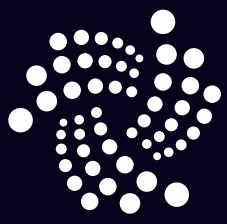

IOTA

SMART CONTRACTS REVIEW

zokyo

February 21th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING IOTA

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issues: 0 points deducted
- 0 Medium issues: = 0 points deducted
- 6 Low issues: 6 resolved = 0 points deducted
- 8 Informational issues: 6 resolved = 0 points deducted

Thus, 100

TECHNICAL SUMMARY

This document outlines the overall security of the IOTA smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the IOTA smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the IOTA team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the IOTA repository:
Repo: https://github.com/iotaledger/legacy-migration-tool

Last commit: f0f71adcc0472b7388b902a59e6da72a7b38c961

The changes were made in these 2 PRs

- https://github.com/iotaledger/legacy-migration-tool/pull/123
- https://github.com/iotaledger/legacy-migration-tool/pull/124

Focus area: changes that were made in the commit hash b4f915d1a25e8d06e1da5a60f8d6a8a013ba9b8c comparing to the f5bb517ad0c9c5570f0080695d161f3d1229901b

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- migration.ts,
- utils.ts,
- Balance.svelte,
- Congratulations.svelte,
- Welcome.svelte,
- Migrate.svelte,
- TransferFragmentedFunds.svelte,
- electrol-builder-config.js,
- main.js,
- electronApi.js,
- build-desktop.yml

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of IOTA smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contract/s by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team discovered issues with low and informational levels of severity, which the IOTA team promptly addressed. Detailed explanations of these findings can be found in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the IOTA team and the IOTA team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Unused constant	Low	Resolved
2	Function parameter never used	Low	Resolved
3	Unused variables	Low	Resolved
4	Using strings in the function	Low	Resolved
5	TODOs in the code	Low	Resolved
6	Unused variables	Low	Resolved
7	The function could be declared as async	Informational	Resolved
8	The function parameter is not described in the JSDoc	Informational	Resolved
9	The function parameter is not described in the JSDoc	Informational	Resolved
10	Function parameters are inconsistent with the JSDoc	Informational	Resolved
11	A local variable is redundant	Informational	Resolved
12	The function parameter is not described in the JSDoc	Informational	Resolved
13	Function parameters are inconsistent with the JSDoc	Informational	Resolved
14	A local variable is redundant	Informational	Resolved

LOW-1 | RESOLVED

Unused constant

File: migration.ts
Constant: PERMANODE
Details:
The given constant was used in the removed code parts.

Recommendation:
Remove the constant declaration

LOW-2 | RESOLVED

Function parameter never used

File: TransferFragmentedFunds.svelte
Function: migrateFunds
Details:
The function has a parameter declared as "isRerun?: boolean", however, it is never used in the function code. Looking for the function calls we can differentiate the two types of call: without parameter (defaulting to false) and with true as a value (from the rerunMigration function).

Recommendation:
Double-check the functionality and remove the parameter if there is no need for it

LOW-3 | RESOLVED

Unused variables

File: Welcome.svelte
Variables: error, busy
Details:
There are two global variables added in the given file, and their values even being assigned during the execution, but never read

Recommendation:
Double-check the functionality and remove the parameters if there is no need for those

LOW-4 | RESOLVED

Using strings in the function

File: Congratulations.svelte
Function: consultExplorerClick
Details:
In the function, there is a variable "urlToOpen", which is being assigned to a URL of the explorer depending on either developer account being used or not. It is better, for refactoring and possible next modifications, to keep them outside the function and declare them as constants

Recommendation:
Declare URL string prefixes as constants

LOW-5 | RESOLVED

TODOs in the code

File: Balance.svelte
Details:
There are multiple "TODO"s in the given file.

Recommendation:
Double check needed implementation is done and remove TODO tags

LOW-6 | RESOLVED

Unused variables

File: Migrate.svelte
Variable: singleMigrationBundleHash
Details:
There is a global variable added in the given file, and its value even being assigned during the execution, but never read

Recommendation:
Double-check the functionality and remove the parameter if there is no need for it

INFORMATIONAL-1 | RESOLVED

Function could be declared as async

File: migration.ts
Function: getLedgerMigrationData
Details:
The given function is already asynchronous, but adding an "async" keyword to it will make it more visible for debugging/refactoring in the future

Recommendation:
onvert function declaration to an "async" with "export const getLedgerMigrationData = async ("

INFORMATIONAL-2 | RESOLVED

Function parameter is not described in the JSDoc

File: migration.ts
Function: createLedgerMigrationBundle
Parameters: migrationAddress, callback
Details:
The given parameters are present in the function, however, they are not described in the JSDoc

Recommendation:
Describe all function parameters in the JSDoc

INFORMATIONAL-3 | RESOLVED

Function parameter is not described in the JSDoc

File: migration.ts
Function: sendOffLedgerMigrationRequest
Parameters: bundleIndex
Details:
The given parameters are present in the function, however, they are not described in the JSDoc

Recommendation:
Describe all function parameters in the JSDoc

INFORMATIONAL-4 | RESOLVED

Function parameters are inconsistent with the JSDoc

File: migration.ts
Function: createMigrationBundle
Parameters: bundle
Details:
The given parameters are present in the function, however they are not described in the JSDoc, instead JSDoc contains the bundleIndex parameter

Recommendation:
Describe all function parameters in the JSDoc

INFORMATIONAL-5 | RESOLVED

Local variable is redudant

File: migration.ts
Function: fetchReceiptForRequest
Variable: receipt
Details:
The receipt variable is being set on line#943, and right after that, it is returned from the function.

Recommendation:
Instead of assigning the result of the "response.json()" to a local variable, return its value from the function directly

INFORMATIONAL-6 | RESOLVED

Function parameter is not described in the JSDoc

File: migration.ts
Function: assignBundleHash
Parameters: didMine
Details:
The given parameters are present in the function, however, they are not described in the JSDoc

Recommendation:
Describe all function parameters in the JSDoc

INFORMATIONAL-7 | RESOLVED

Function parameters are inconsistent with the JSDoc

File: migration.ts
Function: updateLedgerBundleState
Parameters: migrationBundleCrackability
Details:
The given parameters are present in the function, however they are not described in the JSDoc, instead JSDoc contains the migrationBundleCrackability parameter

Recommendation:
Describe all function parameters in the JSDoc

INFORMATIONAL-8 | RESOLVED

Function parameter is not described in the JSDoc

File: migration.ts
Function: createMigrationBundle
Variable: bundleTrytes
Details:
The receipt variable is being set on line#866, and right after that, it is returned from the function.

Recommendation:
Instead of assigning the result of the "prepareTransfers()" to a local variable, return its value from the function directly

	migration.ts, utils.ts, Balance.svelte, Congratulations.svelte, Welcome.svelte, Migrate.svelte,
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	TransferFragmentedFunds.svelte, electrol-builder-config.js, main.js, electronApi.js, build-desktop.yml
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the IOTA team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the IOTA team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo