Yoki Security Scan Results

by Pessimistic

This is not a security audit

This report is public

May 3, 2024

Abstract

This report considers the security of smart contracts of the Yoki protocol. Our task is to find and describe security issues using the static-analysis tools Slither and Slitherin and help resolve them.

The work is financially covered by the Arbitrum Foundation grant.

Disclaimer

Current work does not give any warranties on the security of the code. It is not an audit or its replacement. Performing this scan, we focused on finding as many crucial issues as possible rather than making sure that the protocol was entirely secure. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Summary

In this report, we described issues found in smart contracts of the Yoki protocol.

We scanned the codebase and manually rejected or verified all automated findings, revealing five relevant issues.

The developers fixed three issues.

The entire process is described in the section below.

Scan process

Under the Arbitrum Foundation grant, we researched and developed Arbitrum-specific detectors. They became publicly available with Slitherin v0.6.0 release.

Workflow

This work consisted of five stages:

1. For the scan, we were provided with the Yoki project on a public GitHub repository, commit 0e74992e79d9ac45b9525a9c1a722c473f73a281.
2. For the analysis of the protocol, we launched Slither v0.10.1 and Slitherin v0.6.1 on the provided codebase.
3. One auditor manually checked (rejected or accepted) all findings reported by the tools. The second auditor verified this work. We shared all relevant issues with the protocol developers and answered their questions.
4. The developers reviewed the findings and updated the code accordingly. We reviewed the fixes and found no new issues.
5. We prepared this final report summarizing all the issues and comments from the developers.

Issue categories

Within the confines of this work, we were looking for:

- Arbitrum-specific problems;
- Standard vulnerabilities like re-entrancy, overflow, arbitrary calls, etc;
- Non-compliance with popular standards like ERC20 and ERC721;
- Some access control problems;
- Integration issues with some popular DeFi protocols;
- A wide range of code quality and gas efficiency improvement opportunities.

This scan does not guarantee that these issues are not present in the codebase.

Scan results

Issue category	Number of detectors	Status
Compilation	1	Passed
Arbitrum Integration	3	1 issue found
AAVE Integration	1	Passed
Uniswap V2 Integration	7	Passed
OpenZeppelin	2	Passed
ERC-20	7	Passed
ERC-721	2	Passed
Known Bugs	15	1 issue found (1/1 fixed)
Access Control	3	Passed
Arbitrary Call	5	Passed
Re-entrancy	6	1 issue found (1/1 fixed)
Weak PRNG	2	Passed
Upgradability	2	Passed
Ether Handling	3	Passed
Low-level Calls	2	Passed
Assembly	2	Passed
Inheritance	3	Passed
Arithmetic	2	Passed
Old Solidity Versions Bugs	10	Passed
Code Quality	15	Passed
Best Practices	4	Passed
Gas	7	2 issues found (1/2 fixed)

Discovered Issues

CEI pattern

In the **RPSV1.sol** contract, the `processPayment` function does not follow the Check-Effects-Interactions (CEI) pattern, potentially exposing it to reentrancy attacks through token transfers. We recommend making storage changes before external calls to align with the CEI pattern.

This issue has been fixed at the commit 1f20869d0028a25c25f484eb6ea63c7e3d859b23.

Unnecessary check

In the **RPSV1.sol** contract, the `require` check on line 38 is redundant and can be simplified. Given that the `processingFee` is of type `uint8`, the Solidity compiler already ensures the value is non-negative. Therefore, it is only required to verify that the value is less than 100.

This issue has been fixed at the commit 0db34d1c418d636ad25dd7eab6e3b80162c473e3.

Block timestamp on the Arbitrum chain

The `canExecute` and `processPayment` functions in the **RPSV1.sol** contract rely on the `block.timestamp` value within the Arbitrum contract's code. This behaves differently than on Ethereum, since consecutive blocks can share the same `block.timestamp`. It's important to ensure that the contract logic remains correct despite these differences.

Mark functions as external instead of public

In the **RPSV1.sol** contract, the `initialize`, `subscribe`, `execute`, and `terminate` functions could be declared as `external` instead of `public`. This change not only improves code readability but also optimizes gas consumption.

This issue has been fixed at the commit f127f7ff46ea44bf13fe1ba9cd26239cf8fc74a8.

Immutable variable

The protocol contains **RPSV1Factory.sol** contract where `rpsImpl` variable is set during contract deployment and never change later. We recommend declaring it as `immutable` to reduce gas consumption and improve code quality.

This analysis was performed by Pessimistic:

Yhtyyar Sahatov, Security Engineer
Egor Dergunov, Junior Security Engineer
Pavel Kondratenkov, Senior Security Engineer
Nikita Kirillov, Product Owner
Evgeny Marchenko, CTO
Konstantin Zherebtsov, Business Development Lead

May 3, 2024