Open Dollar

Open Dollar Scan Results

by Pessimistic

This is not a security audit

This report is public

May 2, 2024

Abstract

This report considers the security of smart contracts of the Open Dollar 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 Open Dollar protocol.

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

The developers acknowledged the findings, commented on all issues and created a GitHub issue for one of it.

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 Open Dollar project on a public GitHub repository, commit c0c2034397e6fe08ac5540a739f00444e24b825e .
2. For the analysis of the protocol, we launched Slither v0.10.1 and Slitherin v0.6.0 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, commented on all issues and created GitHub issue to fix one of it in future.
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	Passed
Access Control	3	Passed
Arbitrary Call	5	Passed
Re-entrancy	6	Passed
Weak PRNG	2	Passed
Upgradability	2	1 issue found
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	1 issue found
Best Practices	4	1 issue found
Gas	7	1 issue found

Discovered Issues

Unprotected initializers

The `initializeManager` and `initializeRenderer` functions of the **Vault721** contract are not a part of the `initialize` function and do not have any access control, allowing a malicious user to perform a front-run attack to take over roles of the `safeManager` and `nftRenderer` after the proxy contract deployment. We recommend restricting access to these functions and disabling their initialization on the implementation contract.

> *Comment from the developers: We acknowledge the finding.*

Missing events

There are several contracts where setter functions do not emit an event:

- `updateRedemptionRate` function in the **OracleRelayer.sol** contract;
- `updateBlockDelay` and `updateTimeDelay` functions in the **Vault721.sol** contract.

Emitting of events in setter functions allows to notify the contract owner and relevant parties about important state changes within the contract.

> *Comment from the developers: We acknowledge the finding.*

The developers created a GitHub issue 561.

Unchecked return value

We recommend checking the return value of the external function calls or considering removing the return value for such calls. We also recommend checking the returned value from library calls when appropriate.

> *Comment from the developers: We acknowledge the finding.*

Immutable variables

The protocol contains several contracts where variables are set during contract deployment and never change later. We recommend declaring them as `immutable` to reduce gas consumption.

> *Comment from the developers: We acknowledge the finding.*

Arbitrum block number

We recommend verifying that the `transferFrom` function of the **Vault721** contract is intended to use the Ethereum block number for the block delay logic, according to Arbitrum concepts.

> *Comment from the developers: We acknowledge the finding.*

This analysis was performed by Pessimistic:

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

May 2, 2024