Moby Security Scan Results

by Pessimistic

This is not a security audit

This report is public

May 27, 2024

Abstract

This report considers the security of smart contracts of the Moby 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 Moby protocol.

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

The developers fixed four issues and commented on one.

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 Moby Trade project on a private GitHub repository. We started the work on the commit 2ab64f2fdd6edc366ae8d52a4f522d2fd9951b97 and switched to 70a2fd22ff6d67bea1e8afe29b625e3ca2568da9 in the process.
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, updated the code accordingly, and gave comments on issues they do not intend to fix. 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	2 issues found (2/2 fixed)
Arbitrum Integration	3	Passed
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	1 issue found
Re-entrancy	6	Passed
Weak PRNG	2	Passed
Upgradability	2	1 issue found (1/1 fixed)
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 (1/1 fixed)
Best Practices	4	Passed
Gas	7	Passed

Discovered Issues

Compilation error

Two assignments had a different number of components on their left and right sides, which led to a failure during contract compilation.

This issue has been fixed at the commit 70a2fd22ff6d67bea1e8afe29b625e3ca2568da9

Contract name collision

The codebase contained two libraries named **Address**: in **TransparentUpgradeableProxy.sol** and in **Address.sol**. However, only one compilation artifact is created when a codebase includes two or more contracts with the same name. I.e., some contracts are effectively discarded.

This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884

Unchecked return values of ERC20 calls

The returned value of ERC20 call should be checked explicitly:

- In **BasePositionManager.sol** contract at line 97 in function `_transferOutETHWithGasLimitFallbackToWeth`;
- In **Faucet.sol** contract at line 65 in function `distribute` and line 80 in function `recoverToken`;
- In **OlpManager.sol** contract at line 336 in function `_removeLiquidity`.

According to the ERC20 standard:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned.
```

This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884

Lack of initializer protection

In **RewardTracker.sol** contract, `isSettingInitialized` variable should track if the contract was properly initialized after deployment. However, the variable is never updated, rendering the "initialized" check ineffective. This issue allows to call `initSetting` more than once and change sensitive settings.

Besides, it is recommended by OpenZeppelin to call `_disableInitializers` in the `constructor` of the upgradeable contracts.

This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884

Potentially unauthorized token transfers

The `pluginERC20Transfer` function in **Controller.sol** uses an arbitrary `from` address for a `transferFrom` call, potentially allowing unauthorized token transfers at line 233. A malicious `msg.sender` can steal any tokens that `_account` has approved to the contract. Note that in the current implementation the `_account` also has to approve the plugin to the `msg.sender`.

> *Comment from the developers: We know that it is an unusual pattern for transferring tokens, however, as **PositionManager.sol** can only be the plugin, there won't be any possibility of `msg.sender` stealing tokens from `_account`.*

This analysis was performed by Pessimistic:

Oleg Bobrov, Security Engineer
Egor Dergunov, Junior Security Engineer
Nikita Kirillov, Product Owner
Evgeny Marchenko, CTO
Konstantin Zherebtsov, Business Development Lead

May 27, 2024