# Atomica Security Scan Results

by Pessimistic

This is not a security audit

This report is public

June 1, 2024

## Abstract

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

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

The developers successfully fixed five 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 Atomica project on a private GitHub repository with the following commit: 4aa84c884b7adc5bf1789bf08c1612022cf8fc76.
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 provided commits with fixes.
5. We prepared this final report summarizing all the issues.

## 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 | Passed |
| 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 | 1 issue found (1/1 fixed) |
| Arbitrary Call | 5 | 1 issue found |
| Re-entrancy | 6 | Passed |
| 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 | 1 issue found (1/1 fixed) |
| Best Practices | 4 | 1 issue found (1/1 fixed) |
| Gas | 7 | 1 issue found (1/1 fixed) |

## Discovered Issues

### Arbitrary call

The **PoolImpl** contract includes multiple functions with callback functionality. In `withdraw`, `processWithdrawRequest`, `deposit`, and `depositFor` functions anyone can specify both the target and calldata for an external call. This allows an attacker to exploit existing approvals given to the **PoolImpl** contract. They just call `Token.transferFrom(User, Attacker, Approved amount)` from the name of **PoolImpl** to steal assets from the users.

> *Comment from the developers:* *Fixed, callback in `withdraw`, `processWithdrawRequest`, `deposit`, and `depositFor` functions removed from the codebase.*

### Lack of access control

In the **PoolImpl** contract, setter functions: `setSettlementDistribution` and `setPremiumDistribution` don't have any access control restrictions imposed. This allows anyone to set the value of `premiumDistributionRootHash`, `premiumDistributionUri`, `settlementDistributionRootHash`, and `settlementDistributionUri` variables. We recommend adding a restriction for the caller.

*This issue has been fixed at the commit 70a2fd22ff6d67bea1e8afe29b625e3ca2568da9*

## Unchecked return values of ERC20 calls

There are several places in contracts when the returned value after interactions with token contracts is not checked:

- In **Distributor** contract in functions `expire` at L56 and `claim` at L44;
- In **RewardsImpl** contract in functions `newRewardDistribution` at L105, `_claimReward` at L86 and `addRewardAmount` at L112;
- In **PoolImpl** contract in functions `_claimSettlementAccount` at L328, `release` at L38, `_deposit` at L269, `claimPremiumDistribution` at L399, `contributePremium` at L84, `contributeSettlement` at L96, `_withdraw` at L301, `_claimPremiumAccount` at L316, `claimSettlementDistribution` at L432, and `requestCapital` at L70.

According to the ERC20 standard:

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

Consider using SafeTransfer function to transfer ERC20 tokens.

*This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884*

## Uninitialized variables

In the **RiskPoolStorageModel** contract that is inherited by **RiskPool** contract, a variable `externalAssetBalance` is not initialized. Failure to initialize this variable could lead to errors in the contract logic.

*This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884*

## Lack of events upon crucial parameters change

In the **OperationsImpl** contract, there is a function `setPayoutProcessor`, which does not emit an event. Emitting events in setter functions allows one to notify the contract owner and relevant parties about important state changes within the contract.

*This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884*

## Public functions could be turned into external

In the **PoolImpl** contract, there is a function `totalAssetBalanceAvailable` can be declared external instead of public. It helps to improve code readability and optimize gas consumption in the project.

*This issue has been fixed at the commit c1e0b9ecf7f68e6ffc238e1deb03ea8a86037884*

This analysis was performed by Pessimistic:

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

June 1, 2024