Arbitrum Strategy Manager Security Review

Pashov Audit Group

Conducted by: Said, Hals, jesjupyter

May 5th 2025 - May 6th 2025

Contents

1. About Pashov Audit Group 2
2. Disclaimer 2
3. Introduction 2
4. About Arbitrum Strategy Manager 2
5. Risk Classification 3
   5.1. Impact 3
   5.2. Likelihood 3
   5.3. Action required for severity levels 4
6. Security Assessment Summary 4
7. Executive Summary 5
8. Findings 7
   8.1. Low Findings 7
      [L-01] Missing threshold check in depositIntoAaveV3() may cause scale-down 7
      [L-02] No automatic check for scaleDown after updateMaxPositionThreshold 7
      [L-03] Hardcoded Aave V3 pool address may not match future upgrade 8
      [L-04] Sub-optimal role ID implementation 9
      [L-05] Missing claim for Aave incentives leads to unclaimed rewards 10
      [L-06] Inefficiency from fixed BPS_BUFFER value in position scaling 10

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **TokenLogic-com-au/asset-manager** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Arbitrum Strategy Manager

Arbitrum Strategy Manager is a smart contract that manages the Arbitrum Foundation's wstETH holdings on Aave V3, enabling yield generation through deposits, withdrawals, and reward claims. It includes automated position scaling and emergency functions to adjust exposure based on Aave's liquidity conditions.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - bf3b2f80c56de0bacf9bfee54527f8dce1883576

fixes review commit hash - af9e8308b1a036a97cc6faf5de81a159831454a1

Scope

The following smart contracts were in scope of the audit:

- `ArbitrumStrategyManager`

7. Executive Summary

Over the course of the security review, Said, Hals, jesjupyter engaged with TokenLogic to review Arbitrum Strategy Manager. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	Arbitrum Strategy Manager
Repository	https://github.com/TokenLogic-com-au/asset-manager
Date	May 5th 2025 - May 6th 2025
Protocol Type	Yield Strategy

Findings Count

Severity	**Amount**
Low	6
Total Findings	**6**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Missing threshold check in depositIntoAaveV3() may cause scale-down	Low	Resolved
[L-02]	No automatic check for scaleDown after updateMaxPositionThreshold	Low	Resolved
[L-03]	Hardcoded Aave V3 pool address may not match future upgrade	Low	Resolved
[L-04]	Sub-optimal role ID implementation	Low	Resolved
[L-05]	Missing claim for Aave incentives leads to unclaimed rewards	Low	Resolved
[L-06]	Inefficiency from fixed BPS_BUFFER value in position scaling	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] Missing threshold check in `depositIntoAaveV3()` may cause scale-down

The `depositIntoAaveV3()` function allows addresses with the `CONFIGURATOR_ROLE` to deposit arbitrary amounts of `wstETH` into the Aave V3 pool. However, it does not check whether the resulting position will exceed the `_maxPositionThreshold + BPS_BUFFER`, which is the upper bound that triggers an emergency scale-down in the `scaleDown()` function.

This omission allows the `CONFIGURATOR_ROLE` to unintentionally deposit an amount that will immediately make the position non-compliant with the threshold limit. As a result, an automated scale-down might be triggered soon after deposit, leading to unnecessary gas usage and increased operational overhead.

```
function depositIntoAaveV3(
        uint256 amount
    ) external onlyRole(CONFIGURATOR_ROLE) {
        require(amount > 0, InvalidZeroAmount());

        IERC20(WST_ETH).forceApprove(_aaveV3Pool, amount);
        IPool(_aaveV3Pool).supply(WST_ETH, amount, address(this), 0);

        emit DepositIntoAaveV3(amount);
    }
```

Recommendation: Add a check in `depositIntoAaveV3()` to ensure that the resulting position percentage remains below `_maxPositionThreshold + BPS_BUFFER`.

[L-02] No automatic check for `scaleDown` after `updateMaxPositionThreshold`

The `updateMaxPositionThreshold` function in the `ArbitrumStrategyManager` contract allows the configurator role to update the maximum position threshold.

```
uint256 old = _maxPositionThreshold;
        _maxPositionThreshold = newThreshold;
```

However, there is currently no mechanism in place to **automatically trigger a check for the `scaleDown` function after the threshold is updated**. This oversight could lead to situations where the position percentage exceeds the newly set threshold without the necessary adjustments being made in time, potentially exposing the contract to risks.

[L-03] Hardcoded Aave V3 pool address may not match future upgrade

In the `ArbitrumStrategyManager` contract, the `_aaveV3Pool` variable is defined as an `immutable` address and is set to a hardcoded value of `0x794a61358D6845594F94dc1DB02A252b5b4814aD` during deployment.

```
/// @dev Address of the Aave V3 Pool
    address internal immutable _aaveV3Pool;
```

```
address public constant AAVE_V3_POOL =
        0x794a61358D6845594F94dc1DB02A252b5b4814aD;
    address public constant MERKL_DISTRIBUTOR =
        0x3Ef3D8bA38EBe18DB133cEc108f4D14CE00Dd9Ae;

    function run() public {
        vm.startBroadcast();

        manager = new ArbitrumStrategyManager(
            ADMIN,
            AAVE_V3_POOL,
            TREASURY,
            MERKL_DISTRIBUTOR,
            HYPERNATIVE
        );

        vm.stopBroadcast();
    }
```

However, hardcoding the Aave V3 pool address poses a risk:

The `AaveV3 pool` address is typically managed through the `Aave Addresses Provider`, which allows for updates to the pool address if necessary.

Link Link

`setAddress` Sets the address of the protocol contract stored at the given id, replacing the address saved in the addresses map.

```
/// @inheritdoc IPoolAddressesProvider
  function setAddress
    (bytes32 id, address newAddress) external override onlyOwner {
    address oldAddress = _addresses[id];
    _addresses[id] = newAddress;
    emit AddressSet(id, oldAddress, newAddress);
  }
```

```
/// @inheritdoc IPoolAddressesProvider
  function getPool() external view override returns (address) {
    return getAddress(POOL);
  }
```

Even though `setAddress` should be called with care, it's still possible that it is called and does a hard replacement of the current pool address in the addresses map.

Currently, the `pool` address is hardcoded and may not catch up with the address upgrade.

By hardcoding the address, the contract may interact with an outdated pool, leading to potential issues in functionality and security.

A recommended approach is to use `IPoolAddressesProvider(addressProviderAddress).getPool()` to reference `pool` instead of `hardcoding`

[L-04] Sub-optimal role ID implementation

The `ArbitrumStrategyManager` contract defines role identifiers as public constants using string literals, specifically for the `CONFIGURATOR_ROLE` and `EMERGENCY_ACTION_ROLE`.

```
/// @notice Returns the identifier of the Configurator Role
    /// @return The bytes32 id of the Configurator role
    bytes32 public constant CONFIGURATOR_ROLE = "CONFIGURATOR";

    /// @notice Returns the identifier of the Emergency Action Role
    /// @return The bytes32 id of the Emergency Action role
    bytes32 public constant EMERGENCY_ACTION_ROLE = "EMERGENCY_ACTION";
```

According to the `OpenZeppelin` documentation on `AccessControl`, role identifiers should be defined using the `keccak256` hash function to ensure uniqueness and security. The current implementation does not follow this best practice.

```
* Roles are referred to by their `bytes32` identifier. These should be exposed
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 * ```solidity
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 * ```
```

[L-05] Missing claim for Aave incentives leads to unclaimed rewards

The `ArbitrumStrategyManager` contract supplies `wstETH` into the Aave V3 pool to earn yield. However, it does not implement any function to claim incentive rewards from **Aave's** `RewardsController`. As a result, any LP incentives accrued from Aave remain unclaimed and locked in the Aave pool, never being utilized.

This contradicts the intended design stated in the project documentation, which indicates that funds (including rewards) should be transferred to the `_arbFoundation` address. Without an explicit claim mechanism, the Arbitrum Foundation loses access to yield opportunities provided by Aave's incentive system.

Recommendations

Introduce a function that allows claiming rewards from Aave's `RewardsController`.

[L-06] Inefficiency from fixed `BPS_BUFFER` value in position scaling

The `BPS_BUFFER` constant is currently set to `500`, representing a 5% buffer when scaling down positions in the `ArbitrumStrategyManager` contract.

```
/// @dev Buffer used when scaling down a position to not be close to
    // threshold
    uint256 public constant BPS_BUFFER = 500;
```

This fixed value can lead to **inefficiencies in liquidity management**, particularly when the position percentage (`positionPct`) approaches the maximum position threshold (`_maxPositionThreshold`).

```
if (positionPct >= _maxPositionThreshold) {

                        uint256 bpsToReduce = positionPct + BPS_BUFFER - _maxPosition
            uint256 excessAmount = (availableLiquidity * bpsToReduce) / MAX_BPS;

            /// this happens when positionPct and _maxPositionThreshold
            /// have lower values compared to BPS_BUFFER
            /// for example: if positionPct is 2 bps and _maxPositionThreshold
            // is 1 bps
            /// due to BPS_BUFFER being 500 bps, the amount needed to be
            // withdrawn
            /// (excessAmount) will be bigger than current position.
            /// aave only allows to have a withdraw amount value above
            /// the current position amount, if type(uint256).max is used
            if (excessAmount > suppliedAmount) {
                excessAmount = suppliedAmount;
            }
```

For instance, if both `positionPct` and `_maxPositionThreshold` are equal to `400` (4%), the calculation for `excessAmount` could lead to a situation where all liquidity is withdrawn, even though the position is precisely at the threshold. This creates a cliff operation:

1. If `positionPct` is less than `_maxPositionThreshold`, no action will be taken.
2. If `positionPct` equals `_maxPositionThreshold`, all available liquidity is suddenly withdrawn.

This behavior can result in unintended consequences, **as it does not allow for any buffer or flexibility when the position is at the threshold.**

This behavior contradicts the expected functionality, as the buffer's presence can lead to unnecessary liquidity removal at some times, ultimately causing inefficiencies in fund utilization.

Also, under such circumstances, interest rates usually rise as a result of high utilization. Fully withdrawing the deposit may also hinder effective yield generation.

Recommendations

The comments in the code acknowledge this potential issue, but the reliance on a static buffer value does not allow for adjustments based on changing market conditions or the specific context of the position. As `_maxPositionThreshold` is a variable, it would be prudent for the `BPS_BUFFER` to also be adjustable to better align with the current position dynamics.