Agora Access Control Security Review

Pashov Audit Group

Conducted by: 0xbepresent, Shaka, 0xunforgiven, zark

June 5th 2025 - June 11th 2025

Contents

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 **agora-finance/contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Agora Access Control

Agora Access Control contracts provide a modular, role-based permission system with support for multiple managers, upgradeable proxies, and strict access verification. They include components for managing roles (`AgoraAccessControl`), executing arbitrary calls (`AgoraAccessControlWithExecutor`), handling upgradeable proxy logic and access (`AgoraProxyAdmin`, `AgoraTransparentUpgradeableProxy`), and exposing low-level storage slot access for ERC-1967 proxy admin and implementation addresses (`Erc1967Implementation`).

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:

- 4f1387699378ff67e8d4bf29bd853184c86d1216

Scope

The following smart contracts were in scope of the audit:

- `AgoraAccessControl`
- `AgoraAccessControlWithExecutor`
- `AgoraProxyAdmin`
- `AgoraTransparentUpgradeableProxy`
- `Erc1967Implementation`

7. Executive Summary

Over the course of the security review, 0xbepresent, Shaka, 0xunforgiven, zark engaged with Agora to review Agora Access Control. In this period of time a total of **1** issues were uncovered.

Protocol Summary

Protocol Name	Agora Access Control
Repository	https://github.com/agora-finance/contracts
Date	June 5th 2025 - June 11th 2025
Protocol Type	Access Control contracts

Findings Count

Severity	**Amount**
Low	1
Total Findings	**1**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[L-01]	Manager revocation front-running keeps unauthorized access possible	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Manager revocation front-running keeps unauthorized access possible

The public `assignRole` function in `AgoraAccessControl.sol` allows any existing manager to both add and remove the manager role in one atomic call, with only a guard against removing the *last* manager. A malicious manager (M_2) can exploit two front-running vectors:

1. **Revocation Front-run** M_2 sees M_1's pending revocation of M_2 and front-runs it by revoking M_1 first. This causes M_1's revocation tx to revert the moment it executes (because M_1 is no longer a manager code line 52), leaving M_2 as manager.
2. **Reassignment Front-run** Before any attempt to remove M_2 occurs, M_2 can preemptively call `assignRole` to grant the `ACCESS_CONTROL_MANAGER_ROLE` to a fresh address (M_3). Even if later M_2 is revoked, M_3 remains a manager—effectively evading removal.

```
50: function assignRole
  (string memory _role, address _newAddress, bool _addRole) public virtual {
51:     // Checks: Only Admin can transfer role
52:@>   _requireIsRole
  ({ _role: ACCESS_CONTROL_MANAGER_ROLE, _address: msg.sender });
53:
54:@>   _assignRole
  ({ _role: _role, _newAddress: _newAddress, _addRole: _addRole });
55:@>   if (
56:         bytes(_role).length == bytes(ACCESS_CONTROL_MANAGER_ROLE).length &&
57:         keccak256(bytes(_role)) == keccak256(bytes
  (ACCESS_CONTROL_MANAGER_ROLE)) &&
58:         _getPointerToAgoraAccessControlStorage
  ().roleMembership[_role].length() == 0
59:     ) revert CannotRemoveLastManager();
60: }
```

- At **line 52**, the contract checks the caller holds the manager role.
- At **line 54**, it performs either an add or remove.
- The guard at **lines 55–59** only prevents *removing the final manager*, but does not stop reassignment to a new address.

Consider the following scenario:

1. **Initial state:** `managers = [M1, M2]`.
2. M_1 submits tx (pending):

```
assignRole("ACCESS_CONTROL_MANAGER_ROLE", M2, false);
```

3. **Reassignment Front-run:** M_2 preempts with:

```
assignRole("ACCESS_CONTROL_MANAGER_ROLE", M3, true);
```

   - After tx: `managers = [M1, M2, M3]`.

4. Transaction from step 2 is executed:

```
assignRole("ACCESS_CONTROL_MANAGER_ROLE", M2, false);
```

   - After tx: `managers = [M1, M3]`.

5. **Result:** M_2 remains a manager and has a co-manager M_3 under its control—evading any removal attempt.

Recommendations: Disallow granting the manager role to any address for the "reassignment front-run" problem. For the "Revocation Front-run" problem, I consider it's best to not allow instant revocation between managers and to have a better mechanism for manager roles.