Zellic

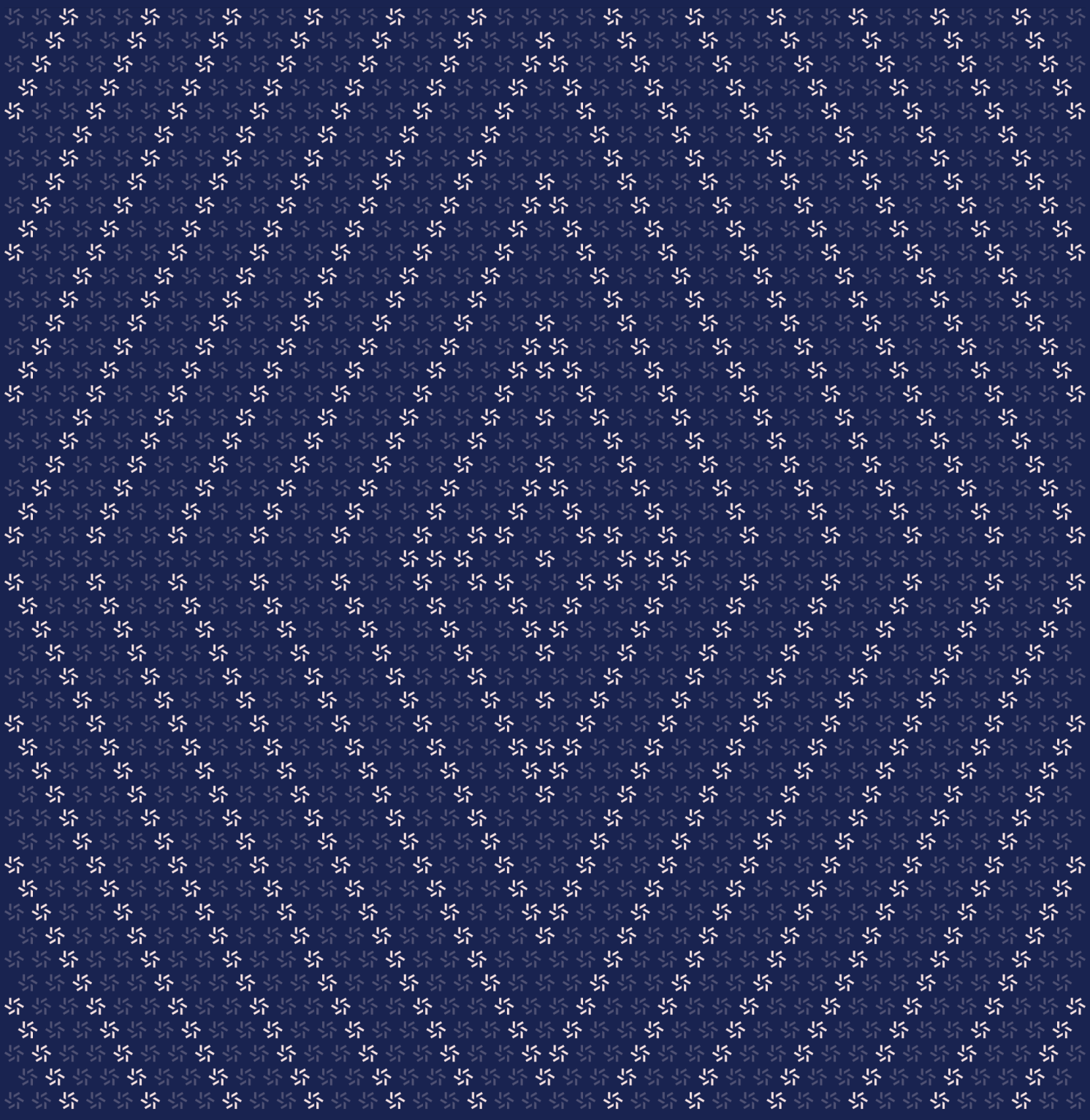
Prepared for
Agora team

Prepared by
Katerina Belotskaia
Jaeeu Kim
Zellic

October 21, 2025

Agora EVM Contracts

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for the Agora team from October 13th to October 22nd, 2025. During this engagement, Zellic reviewed Agora EVM contracts' code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Has the upgradability been implemented correctly?
- Is access control implemented correctly and not bypassable?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

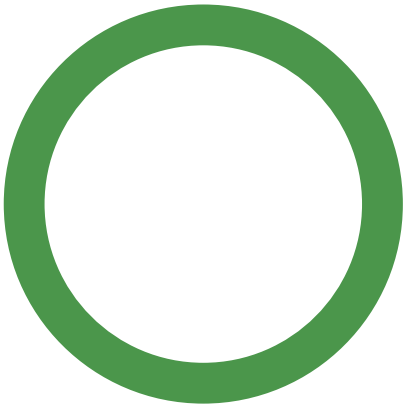
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Agora EVM contracts, we discovered four findings. No critical issues were found. Three findings were of low impact and the remaining finding was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	3
Informational	0



2. Introduction

2.1. About Agora EVM Contracts

The Agora team contributed the following description of Agora EVM contracts:

> Upgradeable LayerZero mint-and-burn OFT Adapter for Agora Dollar and access control layer. This component is reponsible for upgradeability and proxy implement.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Agora EVM Contracts

Type	Solidity
Platform	EVM-compatible
Target	contracts-dev
Repository	https://github.com/amphora-atlas/contracts-dev ↗
Version	`bf8e9db9b5c4c2dfd02cf22037e08236610475e9`
Programs	`src/contracts/**/*.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 2.3 person-weeks. The assessment was conducted by two consultants over the course of one and a half calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

Pedro Moura
Engagement Manager
pedro@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

October 13, 2025	Start of primary review period
October 14, 2025	Kick-off call
October 22, 2025	End of primary review period

3. Detailed Findings

3.1. Missing zero-address validation in the execute function

Target	AgoraAccessControlWithExecutor		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The `_execute` function in AgoraAccessControlWithExecutor does not validate that the target address is not the zero address before making the call.

```
function _execute(address _target, bytes memory _data, uint256 _value)
    internal {
    _requireSenderIsRole({ _role: ACCESS_CONTROL_MANAGER_ROLE });
    (bool success, ) = _target.call{ value: _value }(_data);
    if (!success) revert ExecuteCallFailed();
}
```

Impact

If a zero address is passed as the target, the call will be made to the zero address, which will consume gas but not execute any meaningful code. This could lead to unexpected behavior and gas waste.

Recommendations

Add zero-address validation.

```
function _execute(address _target, bytes memory _data, uint256 _value)
    internal {
    _requireSenderIsRole({ _role: ACCESS_CONTROL_MANAGER_ROLE });
    if (_target == address(0)) revert InvalidTarget();
    (bool success, ) = _target.call{ value: _value }(_data);
    if (!success) revert ExecuteCallFailed();
}
```

Remediation

This issue has been acknowledged by Agora team.

3.2. Missing zero-address validation in role-granting functions

Target	AgoraAccessControl		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The `grantAccessControlManagerRole` function does not validate that the provided member address is not the zero address.

```
function grantAccessControlManagerRole(address _member) public virtual {
    _requireSenderIsRole({ _role: ACCESS_CONTROL_MANAGER_ROLE });
    _assignRole({ _role: ACCESS_CONTROL_MANAGER_ROLE, _member: _member,
    _addRole: true });
}
```

Impact

Granting a role to the zero address treats the zero address as a valid role member, which is likely not intended.

Recommendations

Add zero-address validation.

```
function grantAccessControlManagerRole(address _member) public virtual {
    _requireSenderIsRole({ _role: ACCESS_CONTROL_MANAGER_ROLE });
    if (_member == address(0)) revert InvalidMember();
    _assignRole({ _role: ACCESS_CONTROL_MANAGER_ROLE, _member: _member,
    _addRole: true });
}
```

Remediation

This issue has been acknowledged by Agora team.

3.3. Unnecessary event emission in `_assignRole` function

Target	AgoraAccessControl		
Category	Code Maturity	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The `_assignRole` function emits `RoleAssigned` and `RoleRevoked` events even when the state has not changed. This occurs when an address is already assigned to a role, already removed from a role, or not assigned to a role yet.

```
function _assignRole(string memory _role, address _member, bool _addRole)
    internal virtual {
    _requireRoleExists({ _role: _role });
    _setRoleMembership({ _role: _role, _member: _member, _insert: _addRole });

    // Events are emitted regardless of whether the state actually changed
    if (_addRole) emit RoleAssigned({ role: _role, member: _member });
    else emit RoleRevoked({ role: _role, member: _member });
}
```

Impact

This causes unnecessary event emissions and confusion for off-chain systems monitoring role changes.

Recommendations

Check if the state actually changed before emitting events

Remediation

This issue has been acknowledged by Agora team.

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

4.1. Component: Agora EVM contracts

Description

This component implements base contracts for access control and contract upgradability. The contracts are designed to be inherited by other contracts in the ecosystem.

AgoraAccessControl provides role-based access control with enumerable membership tracking using ERC-7201 namespaced storage. It supports role assignment, revocation, and membership enumeration. AgoraAccessControlWithExecutor extends this with executor functionality for arbitrary calls.

AgoraProxyAdmin manages proxy upgrades with access control, extending AgoraAccessControl for role-based proxy management. AgoraTransparentUpgradeableProxy is a transparent upgradable proxy that integrates with AgoraProxyAdmin.

Erc1967Implementation provides visibility into ERC-1967 proxy admin storage and storage-slot management.

The contracts use ERC-7201 namespaced storage to prevent storage collisions during upgrades.

Invariants

- Only addresses with appropriate roles can perform restricted operations; `ACCESS_CONTROL_MANAGER_ROLE` manages other roles.
- Roles can only be added/removed by `ACCESS_CONTROL_MANAGER_ROLE`. A role cannot be removed if it has active members.
- ERC-7201 namespaced storage prevents storage collisions during upgrades.
- Only `ACCESS_CONTROL_MANAGER_ROLE` can upgrade proxy implementations.
- Each role is uniquely identified by its string name.
- Role membership is tracked using `EnumerableSet`.
- The `ACCESS_CONTROL_MANAGER_ROLE` members cannot remove themselves.

Test coverage

Cases covered

- Role assignment and revocation by `ACCESS_CONTROL_MANAGER_ROLE`
- Role membership enumeration using `EnumerableSet`
- Role-existence validation
- Cannot remove roles with active members
- Event emission for role operations
- The `AbstractRoleTest` framework
- Batch role operations
- Unauthorized access prevention
- Role-gated function access control

Cases not covered

- Self-protection (cannot revoke own `ACCESS_CONTROL_MANAGER_ROLE`)
- Role-name–length validation (32 bytes max)

5. Assessment Results

During our assessment on the scoped Agora EVM contracts, we discovered four findings. No critical issues were found. Three findings were of low impact and the remaining finding was informational in nature.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.