Zellic

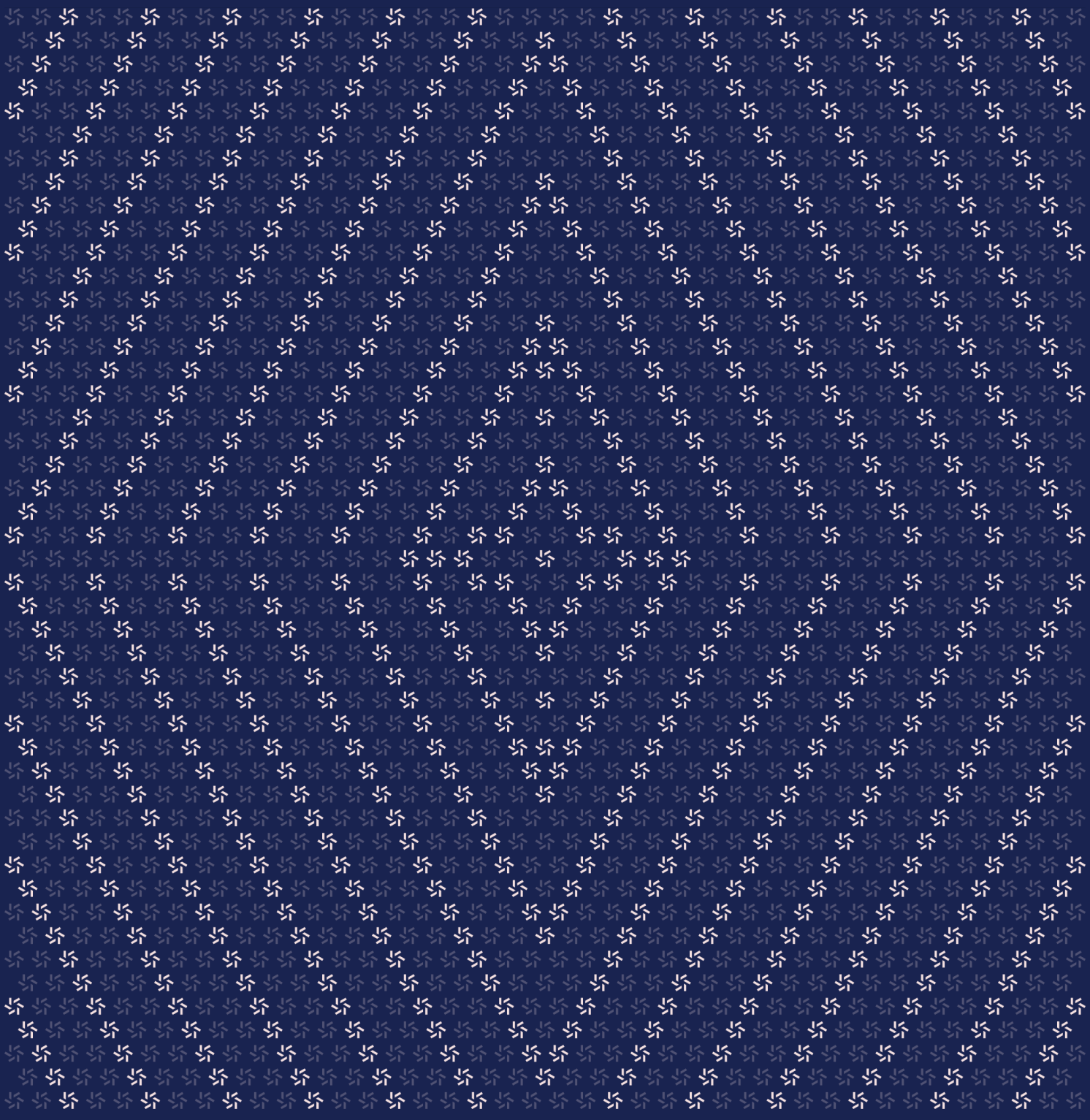
Prepared for
Agora team

Prepared by
Katerina Belotskaia
Jaeeu Kim
Zellic

October 21, 2025

AgoraDollar (AUSD)

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 AgoraDollar (AUSD)'s 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?
- Are token transfers and ERC-20 operations blocked while the contract is paused?
- Are frozen users prevented from transferring, depositing, or withdrawing tokens?
- Are authorized transfers implemented correctly and secure?
- Is the permit functionality implemented correctly and not exploitable?

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

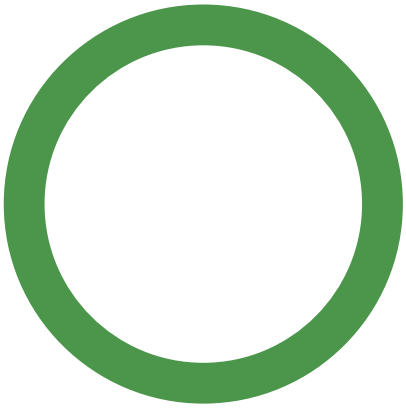
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped AgoraDollar (AUSD) contracts, we discovered three findings, all of which were low impact.

Breakdown of Finding Impacts

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



2. Introduction

2.1. About AgoraDollar (AUSD)

The Agora team contributed the following description of AgoraDollar (AUSD):

> Upgradeable LayerZero mint-and-burn OFT Adapter for Agora Dollar and access control layer. This component is responsible for token implements which are transfers, authorized transfers, permit functionality, and user freezing.

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:

AgoraDollar (AUSD) Contracts

Type	Solidity
Platform	EVM-compatible
Target	agora-dollar-evm-dev
Repository	https://github.com/amphora-atlas/agora-dollar-evm-dev ↗
Version	`3bed93380d393709bb0169d1b62e35ac1f4ff0e7`
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. Inconsistent pausable checks in the proxy contract

Target	AgoraDollarErc1967Proxy		
Category	Protocol Risks	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

In the AgoraDollarErc1967Proxy contract, if the implementation has not been upgraded yet, `isTransferPaused` and `isSignatureVerificationPaused` are checked directly. After the upgrade, these checks are delegated to the new implementation. If the pausable functionality is intended to remain consistent across all implementations, these validations should be performed before the delegate call.

```
// Before upgrade: direct checks
if (isTransferPaused()) revert TransferPaused();
if (isSignatureVerificationPaused()) revert SignatureVerificationPaused();

// After upgrade: delegated to implementation
// Implementation may have different pausable logic
```

Impact

This inconsistency leads to different pausable behavior before and after upgrades, causing unexpected behavior if the pausable logic is not maintained across implementations.

Recommendations

Perform pausable checks before delegate calls to ensure consistent behavior.

Remediation

This issue has been acknowledged by Agora team.

Agora team provided the following response to this finding:

This is by design for the proxy to be un-opinionated. We are replacing the implementation explicitly.

3.2. Missing zero-address validation in the transfer function

Target	AgoraDollarErc1967Proxy		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The transfer function does not check that the receiver address is not the zero address. This could lead to tokens being sent to the zero address, effectively burning them.

```
function transfer(address _to, uint256 _transferValue) external returns (bool)
    {
    // Get data from implementation slot as a uint256
    uint256 _contractData = StorageLib.sloadImplementationSlotDataAsUint256();

    bool _isTransferUpgraded = _contractData.isTransferUpgraded();
    if (_isTransferUpgraded) {
        // new implementation address is stored in the least significant 160
    bits of the contract data
        address _newImplementation = address(uint160(_contractData));
        _delegate({ implementation: _newImplementation });
    } else {
        // Checks: contract-wide access control
        if (_contractData.isTransferPaused())
    revert StorageLib.TransferPaused();

        // Effects: Transfer the tokens
        _transfer({ _from: msg.sender, _to: _to, _transferValue:
    _transferValue.toUint248() });
        return true;
    }
}
```

Impact

If a zero address is passed as the receiver, tokens are permanently lost.

Recommendations

Add zero-address validation.

Remediation

This issue has been acknowledged by Agora team.

Agora team provided the following response to this finding:

> Not a security issue. Checking for zero-address is up to the user.

3.3. Uninitialized implementation address in the proxy constructor

Target	AgoraDollarErc1967Proxy		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

In the AgoraDollarErc1967Proxy contract, the implementation address is not provided to the constructor, so it equals zero until the admin directly upgrades the implementation. This could lead to unexpected behavior if the proxy is used before the implementation is set.

```
constructor(
    ConstructorParams memory _params
) payable Eip712(_params.eip712Name, _params.eip712Version, address(this)) {
    // Effects: Set the proxy admin address in both bytecode and storage
    // Stored directly in bytecode for gas efficiency
    PROXY_ADMIN_ADDRESS = address(new AgoraProxyAdmin({ _initialOwner:
    _params.proxyAdminOwnerAddress }));
    // Stored again in storage to comply with Erc1967 standard
    StorageLib.getPointerToAgoraDollarErc1967ProxyAdminStorage().
        proxyAdminAddress = PROXY_ADMIN_ADDRESS;

    // Emit event
    emit AdminChanged({ previousAdmin: address(0), newAdmin:
    PROXY_ADMIN_ADDRESS });
}
```

Impact

If the proxy is used before the implementation is upgraded, all calls fail since there is no valid implementation to delegate to.

Recommendations

Set a default implementation address in the constructor, or add validation to prevent usage before the implementation is set.

Remediation

This issue has been acknowledged by Agora team.

Agora team provided the following response to this finding:

> We need to know the proxy address before deploying the implementation. So this is an intentional design choice.

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: AgoraDollar (AUSD)

Description

This component implements an upgradable ERC-20 stablecoin with advanced compliance features. The token supports authorized transfers, permit functionality, user freezing, and pausable operations.

AgoraDollarErc1967Proxy is the main token contract that implements ERC-20 functionality with additional features for compliance and governance. It supports authorized transfers where only designated addresses can transfer tokens and with permit functionality for gasless approvals and user freezing to prevent specific addresses from transferring tokens.

The contract is upgradable using a transparent proxy pattern and includes pausable functionality to halt all token operations when needed. Access control is implemented to manage administrative functions and authorized transfer permissions.

The token uses ERC-7201 namespaced storage to prevent storage collisions during upgrades and implements EIP-2612 permit functionality for gasless approvals.

Invariants

- Token balances must always be nonnegative and cannot exceed the total supply.
- EIP-3009 authorized transfers require valid signatures and nonexpired authorizations.
- Frozen users cannot transfer, approve, or perform any token operations.
- When the contract is paused, all token operations (transfer, approve, permit) are blocked.
- The total supply must equal the sum of all individual balances.
- Permit signatures must be valid and nonexpired to be accepted.
- Only the contract owner or designated roles can freeze/unfreeze users.
- Only the contract owner or designated roles can pause/unpause the contract.
- ERC-7201 namespaced storage prevents storage collisions during upgrades.

Test coverage

Cases covered

- Basic ERC-20 functionality (transfer, approve, allowance)

- Authorized transfers (EIP-3009) with valid/invalid signatures
- User freezing and unfreezing operations
- Pausable functionality for minting and burning
- Permit functionality (EIP-2612) with valid signatures
- Access control for administrative functions
- Upgrade functionality for transfer functions
- Role-based access control (minter, burner, pauser, freezer)

Cases not covered

- Self-protection (cannot revoke own `ACCESS_CONTROL_MANAGER_ROLE`)
- Role-name–length validation (32 bytes max)
- Explicit signature-malleability checks (mitigated by nonce-based replay protection)

5. Assessment Results

During our assessment on the scoped AgoraDollar (AUSD) contracts, we discovered three findings, all of which were low impact.

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.