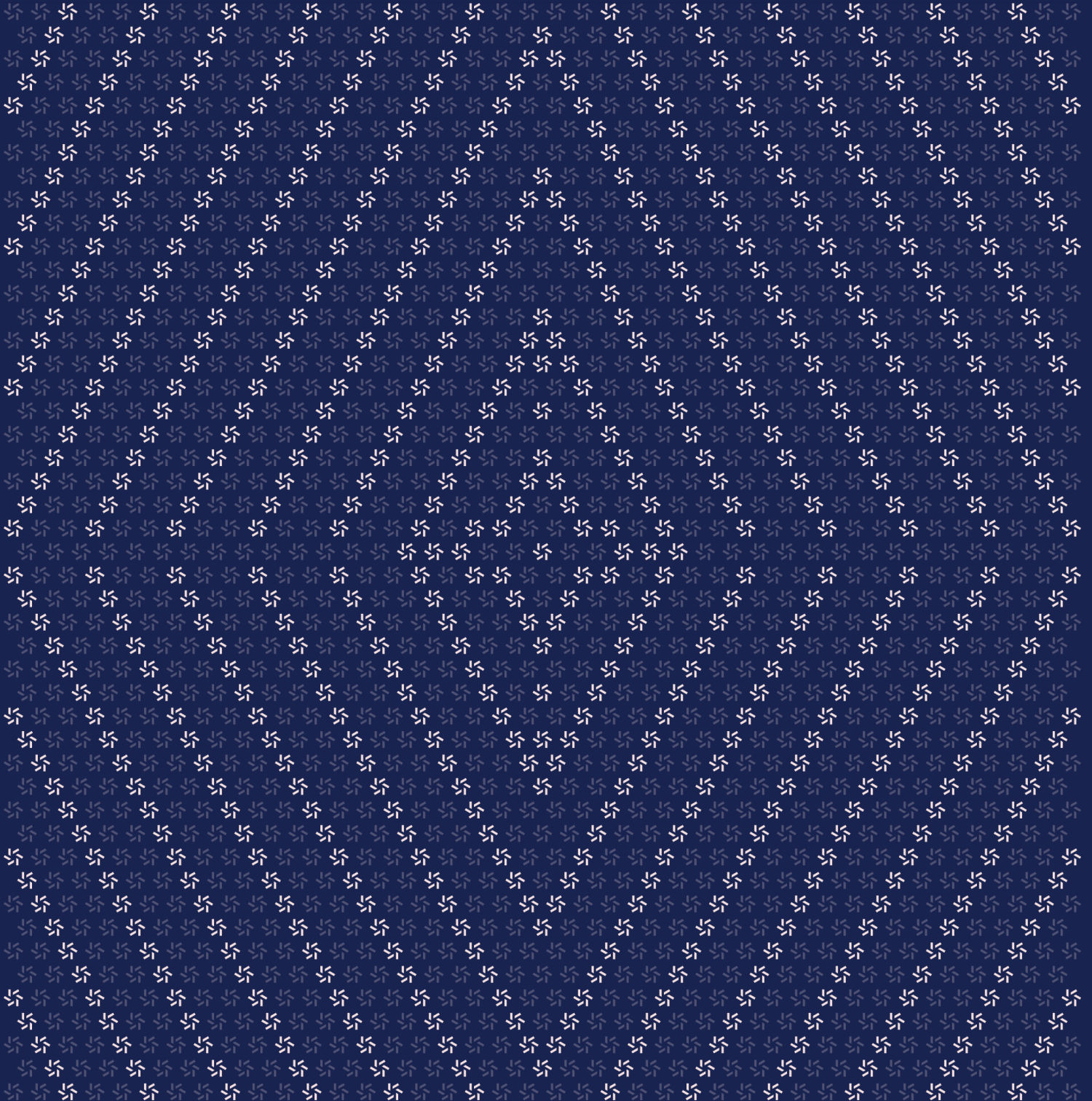


October 21, 2025

Whitelabel

Smart Contract Security Assessment



Contents

About Zellic	4
1. Overview	4
1.1. Executive Summary	5
1.2. Goals of the Assessment	5
1.3. Non-goals and Limitations	5
1.4. Results	5
2. Introduction	6
2.1. About Whitelabel	7
2.2. Methodology	7
2.3. Scope	9
2.4. Project Overview	9
2.5. Project Timeline	10
3. Detailed Findings	10
3.1. Frozen caller can initiate withdrawals for unfrozen accounts	11
3.2. Test coverage	13
3.3. The maxDeposit, maxMint, maxWithdraw, and maxRedeem functions do not consider pause/freeze state	15
4. System Design	16
4.1. Component: Whitelabel	17

5.	Assessment Results	18
5.1.	Disclaimer	19

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 Whitelabel'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-4626 operations blocked while the contract is paused?
 - Are frozen users prevented from transferring, depositing, or withdrawing from the vault?
-

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

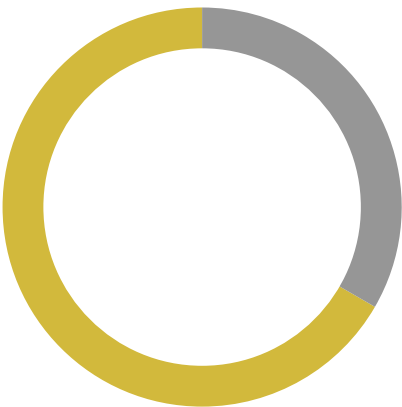
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Whitelabel contracts, we discovered three findings. No critical issues were found. Two findings were of medium impact and the remaining finding was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	2
Low	0
Informational	1



2. Introduction

2.1. About Whitelabel

The Agora team contributed the following description of Whitelabel:

Upgradeable ERC4626 vault contract that wraps AUD (AgoraDollar) tokens. The vault allows users to deposit AUD tokens and receive shares 1:1 in return.

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:

Whitelabel Contracts

Type	Solidity
Platform	EVM-compatible
Target	whitelabel-evm-dev
Repository	https://github.com/amphora-atlas/whitelabel-evm-dev ↗
Version	2763e4d8564634ae4a5e0253e6b5a985802be736
Programs	Whitelabel TwoStepProxyAdmin TransparentUpgradeableProxy

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. Frozen caller can initiate withdrawals for unfrozen accounts

Target	Whitelabel		
Category	Business Logic	Severity	Medium
Likelihood	Medium	Impact	Medium

Description

In the Whitelabel contract, the update function validates the from and to addresses to ensure they are not frozen during deposit and withdrawal operations.

```
function _update(
    address from,
    address to,
    uint256 value
) internal override(ERC20Upgradeable, ERC20PausableUpgradeable) {
    if (AUSD.isAccountFrozen({ _account: from }))
        revert IAgoraDollar.AccountIsFrozen({ frozenAccount: from });
    if (AUSD.isAccountFrozen({ _account: to }))
        revert IAgoraDollar.AccountIsFrozen({ frozenAccount: to });
    super._update(from, to, value);
}
```

However, while the deposit functionality enforces `caller == owner`, the withdraw and redeem function allows a caller to use allowance and initiate a withdrawal on behalf of another user.

```
function withdraw(uint256 assets, address receiver, address owner)
    public virtual returns (uint256) {
    // [...]
    _withdraw(_msgSender(), receiver, owner, assets, shares);
    return shares;
}

function redeem(uint256 shares, address receiver, address owner)
    public virtual returns (uint256) {
    // [...]
    _withdraw(_msgSender(), receiver, owner, assets, shares);
    return assets;
}
```

Because the update function does not validate `msg.sender`, a frozen caller can still successfully

trigger withdrawals for an unfrozen user, if sufficient allowance has been granted.

Impact

A frozen caller is able to initiate withdrawals for other users using allowance, bypassing the intended freeze restrictions.

Recommendations

Consider updating the `update` function to also validate `msg.sender`, ensuring that frozen callers cannot perform any operations with tokens.

Remediation

This issue has been acknowledged by Agora team, and a fix was implemented in commit [3c342d4d](#).

3.2. Test coverage

Target	Project-wide		
Category	Protocol Risks	Severity	Medium
Likelihood	N/A	Impact	Medium

Description

When building a complex contract with multiple moving parts (state changes) and dependencies, comprehensive testing is essential. This includes testing for both positive and negative scenarios on every function that touches the contract's state. Positive tests should verify that each function's side effect is as expected, while negative tests should cover every revert, preferably in every logical branch.

Impact

Good test coverage has multiple effects:

- It finds bugs and design flaws early (preaudit or prerelease).
- It gives insight into areas for optimization (e.g., gas cost).
- It displays code maturity.
- It bolsters customer trust in the product.
- It improves understanding of how the code functions, integrates, and operates — for developers and auditors alike.
- It increases development velocity long-term.

The last point may seem contradictory given the time investment to create and maintain tests. To expand upon it, tests help developers trust their own changes. It is difficult to know if a code refactor — or even just a small one-line fix — breaks other code if there are no tests. This is especially true for new developers or those returning to the code after a prolonged absence.

Tests have your back here. They are an excellent indicator that the existing functionality was most likely not broken by a change to the code. Without a comprehensive test suite, the above benefits are lost. This increases the likelihood of bugs and vulnerabilities going unnoticed until after deployment, which can be costly and damaging to the project's reputation.

Recommendations

Despite extensive test coverage, some edge cases are not yet covered; therefore, we recommend to implement the test cases provided below:

- After donation, the conversion rate is not affected.
- The Approve for the frozen account is not reverted, but `transferFrom`, which uses updated allowance is blocked.
- Revoking allowance by a frozen account succeeds.
- The Approve is not blocked when the contract is paused.
- The permit is not blocked when the contract is paused.
- Upgrading preserves the same storage state (balances, allowances, owner, pause status, name/symbol/decimals).

Remediation

This issue has been acknowledged by Agora team, and a fix was implemented in commit [bdf7df5c](#).

3.3. The maxDeposit, maxMint, maxWithdraw, and maxRedeem functions do not consider pause/freeze state

Target	Whitelabel		
Category	Business Logic	Severity	Informational
Likelihood	N/A	Impact	Informational

The current Whitelabel ERC-4626 implementation does not override maxDeposit, maxMint, maxWithdraw, or maxRedeem, so these functions return the maximum values that ignore pause and account-freeze constraints. This causes mismatches with runtime behavior where operations revert due to paused() or account freezing.

```
function maxDeposit(address) public view virtual returns (uint256) {
    return type(uint256).max;
}

function maxMint(address) public view virtual returns (uint256) {
    return type(uint256).max;
}

function maxWithdraw(address owner) public view virtual returns (uint256) {
    return _convertToAssets(balanceOf(owner), Math.Rounding.Floor);
}

function maxRedeem(address owner) public view virtual returns (uint256) {
    return balanceOf(owner);
}
```

Currently, maxDeposit, maxMint, maxWithdraw, and maxRedeem return unconstrained values even when the contract is paused or the relevant account is frozen. However, these functions should reflect both global and user-specific limits and return zero when the corresponding action is temporarily disabled. Since withdrawing/redeeming receives only the owner address, limits can only reflect the owner's freeze status.

Impact

Integrations and UIs that rely on maxDeposit, maxMint, maxWithdraw, or maxRedeem can reflect incorrect capabilities, leading to transaction failures and poor UX.

Recommendations

Override `maxDeposit`, `maxMint`, `maxWithdraw`, and `maxRedeem` in `Whitelabel` to return 0 when the contract is paused or the provided account address is frozen. Otherwise, delegate to `super`.

Remediation

This issue has been acknowledged by Agora team, and a fix was implemented in commit [bdf7df5c](#).

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: Whitelabel

Description

This is an upgradable ERC-4626 vault contract with a 1:1 conversion rate that wraps AUDS (AgoraDollar) tokens. This contract implements the standard ERC-4626 functionality, like depositing, withdrawing, minting, redeeming, and so on. Users can deposit AUDS tokens and receive shares 1:1 in return; the same amount of shares are burned when withdrawing.

The depositing, withdrawing, minting, redeeming, and transferring are all blocked when the contract is paused, which is controlled by the contract owner. The AgoraDollar contract supports the freeze mechanism, so the transfers, deposits, and withdrawals are all blocked when the account is frozen by the AgoraDollar contract.

The freezing status of the caller and receiver are performed separately.

- For the depositing action, the AUDS tokens are transferred from the user to the contract, so the freeze check is performed for the user's account (sender) during the `transferFrom` function. Further, during minting shares, the receiver's account is checked for freeze status by calling `AUDS.isAccountFrozen({ _account: to })` during the `_update` function execution.
- For the withdrawing action, the owner of shares is verified during the burning action by calling `AUDS.isAccountFrozen({ _account: from })` during the `_update` function execution, and the receiver of AUDS tokens is checked for freeze status during transferring from this contract by calling `transferFrom`.

Invariants

- The amount of AUDS tokens deposited should be equal to the amount of shares minted. And the amount of shares burned should be equal to the amount of AUDS tokens withdrawn. This is guaranteed by the `_convertToShares` and `_convertToAssets` functions, which return the provided assets and shares, respectively.
- Donation does not affect the conversion rate.
- When paused, the depositing, withdrawing, minting, redeeming, and transferring functionalities are all blocked.
- Only the owner of the contract can pause and unpause the contract.
- Frozen accounts cannot send or receive shares tokens and cannot deposit or withdraw AUDS tokens.

- The contract cannot be initialized more than once.

Test coverage

Cases covered

- The version of the contract should be 1.0.0.
- The two-step ownership-transfer process is supported.
- The owner of the contract can cancel the ownership transfer by setting the `pendingOwner` to the zero address.
- Token transfer is performed successfully if account is not frozen or contract is not paused.
- Users can assign the allowance for other accounts to transfer shares tokens.
- During pausing, users cannot deposit or mint.
- During pausing, users cannot withdraw or redeem.
- Transferring tokens is blocked when the contract is paused.
- The contract cannot be initialized more than once.
- Decimals, name, and symbol are set correctly during initialization.
- Frozen accounts cannot transfer or receive shares tokens.
- Frozen accounts cannot deposit/mint or withdraw/redeem AUD tokens.
- Nonfrozen accounts cannot withdraw or redeem on behalf of a frozen account.
- Nonfrozen accounts can deposit/mint or withdraw/redeem as expected.
- The `TotalAssets` does not include the donation amount.
- Approving with permit is supported.
- The `implementationAddress` and `proxyAdminAddress` are set correctly.
- The `renounceOwnership` function is supported.
- After `renounceOwnership`, the `pendingOwner` is set to the zero address.
- The `ProxyAdmin` can successfully upgrade the contract to a new implementation.

Cases not covered

- After donation, the conversion rate is not affected.
- The `Approve` for the frozen account is not reverted, but `transferFrom` is blocked.
- Revoking allowance by a frozen account succeeds.
- The `Approve` is not blocked when the contract is paused.
- The `permit` is not blocked when the contract is paused.
- Upgrading preserves the same storage state (balances, allowances, owner, pause status, name/symbol/decimals).

5. Assessment Results

During our assessment on the scoped Whitelabel contracts, we discovered three findings. No critical issues were found. Two findings were of medium 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.