OpenZeppelin | security

Wenia TokenizationW Smart Contract Security Audit

December 23, 2025

Table of Contents

Summary

Type	Stablecoins
Timeline	From 2025-12-02 To 2025-12-08
Languages	Solidity

Total Issues	13 (13 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	10 (10 resolved)

Scope

OpenZeppelin audited the Wenia TokenizationW Smart Contract repository at commit 8e3519a.

In scope were the following files:

```
contracts
├── TokenizationWBase.sol
├── TokenizationWExtension.sol
├── EmergencyOracle.sol
├── interfaces
│   └── AggregatorV3Interface.sol
└── scripts
    └── upgradeContract.ts
```

In addition to the files listed above, the audit included an upgrade-safety review of the deployed `StableWBase` and `StableWPolygon` contracts.

During the testing process, the Wenia team identified an issue in the `deployContract.ts` script pertaining to the initialization data structure and an outdated term "blacklist" that should have been replaced by "denylist". Since this script was outside the audit scope, the Wenia team specifically provided pull request #45 for review. The changes introduced in this pull request were reviewed for correctness, confirming that the update was implemented correctly and resolved the relevant issue.

Note that a full review of the `deployContract.ts` file was not conducted as it was not part of the original scope.

System Overview

The TokenizationW system is an upgradeable ERC-20-based framework for issuing fully backed digital assets, including stablecoins, real-world assets (RWAs), and digitized securities. Each deployment represents an independent tokenized asset with its own operational parameters and reserve backing. The system integrates a Proof-of-Reserve (PoR) mechanism via Chainlink-compatible oracles to ensure the circulating supply remains backed by verifiable reserves.

Key Features

- **UUPS Upgradeability (EIP-1822)**: Controlled and secure contract upgrades while preserving state
- **EIP-7201 Namespaced Storage**: Collision-free storage layout across upgrade versions
- **Proof-of-Reserve Integration**: Minting constrained by oracle-reported reserve values
- **Role-Based Access Control**: Granular privilege separation across operational functions
- **Compliance Controls**: Denylist for sanctioned addresses; optional allowlist for permissioned transfers

Contract Architecture

- `TokenizationWBase.sol`: Core ERC-20 functionality, role definitions, denylist/allowlist management, metadata URI storage
- `TokenizationWExtension.sol`: Proof-of-Reserve enforcement, mint/burn logic, oracle heartbeat validation, and limit configuration
- `EmergencyOracle.sol`: Chainlink-compatible fallback oracle under owner control for emergency reserve updates

Storage Layout

The TokenizationW system uses namespaced storage following the EIP-7201 pattern to isolate state across upgradeable modules and prevent storage collisions. The `TokenizationWBase` contract stores the denylist and allowlist mappings, the allowlist enable flag, and the metadata

URI. The `TokenizationWExtension` contract stores the oracle reference, the oracle heartbeat interval, the current maximum mint/burn limit, and the maximum transfer limit. Each module maps its state to a dedicated storage slot selector, ensuring layout stability and safe upgrades across future versions.

Trust Assumptions

During the audit, the following trust assumptions were identified:

- The initial supply mint is limited to `MAX_MINT_BURN_LIMIT` which is assumed to be enough by the owners.
- It is assumed that no account holding a privileged role (e.g. `DEFAULT_ADMIN_ROLE`, `FINANCIAL_ROLE`) will act maliciously or be compromised.
- The system assumes that the Oracle will return data with 6 decimals, matching `TokenizationWExtension`'s decimals. This is critical given that most Chainlink USD-based feeds default to 8 decimals. A mismatch here will result in comparison failures during the minting process.
- Safe proxy upgrades rely on the `upgradeContract` script using the correct contract name and passing 0 bytes as calldata. Deviating from this procedure may result in initialization errors or the proxy pointing to an incorrect implementation.
- The `oracleDataHeartbeat` is configured by governance (Financial or Admin roles). While the contract enforces a range (non-zero, < 7 days), it cannot verify the oracle's actual on-chain update rate. It is assumed governance will configure a heartbeat that matches the oracle's real-world behavior to prevent the rejection of valid data.

Privileged Roles

The system implements role-based access control with the following privileges:

- **`DEFAULT_ADMIN_ROLE`** : This role has full system authority and is exempt from denylist checks. It can grant/revoke any role, set maximum values for mint/burn and transfers, add and remove addresses from denylist and allowlist, activate or deactivate the allowlist, destroy denylisted addresses' funds, configure the oracle and uri, mint and burn, pause, unpause, and upgrade the proxy.

- `UPGRADE_ROLE` : This role can upgrade the proxy to new contract implementations and can update the oracle address.
- `MINTER_ROLE` : This role can mint tokens up to the oracle-reported reserve limit.
- `BURN_ROLE` : This role can burn tokens from caller's own balance.
- `GRANT_ROLE` : This role can grant roles to addresses.
- `REVOKE_ROLE` : This role can revoke roles from addresses.
- `PAUSER_ROLE` : This role can pause and unpause all token operations.
- `FINANCIAL_ROLE` : This role can configure mint/burn/transfer limits and the oracle heartbeat.
- `COMPLIANCE_ROLE` : This role can manage denylist entries.
- `ALLOWLIST_ROLE` : This role can add or remove addresses from the allowlist and set the allowlist status flag.
- `DENYLISTED_BURN_ROLE` : This role can destroy tokens held by denylisted addresses.
- `URI_ROLE` : This role can update the tokenization metadata URI.

High Severity

H-01 Denylisted Spenders Can Bypass Restrictions via `transferFrom` Method

The protocol implements an allowlist and denylist to prevent specific addresses from participating in token transfers. However, the `transferFrom` function does not verify the status of the spender (the `msg.sender`) against these lists. Consequently, if a spender is granted an allowance and is subsequently denylisted or removed from the allowlist, they are still able to move funds on behalf of the token owner.

Consider modifying the `transferFrom` function to validate that the `msg.sender` is allowed to participate in token operations. Doing so will ensure that restricted accounts cannot utilize previously granted allowances.

Update: *Resolved in pull request #33 at commit 85c306d.*

Medium Severity

M-01 Permit Function Bypasses Pause and Access Control Checks

The protocol implements access-control mechanisms such as pausing, denylisting, and allowlisting to restrict token operations. These checks are correctly enforced within the standard `approve` function. However, the `permit` function, inherited from the `ERC20PermitUpgradeable` module, does not include these validations. Consequently, it is possible to approve an allowance via signature even when the token is paused or when the caller or approver addresses are denylisted or not allowlisted.

Consider overriding the `permit` function to ensure that it adheres to the same access-control logic and state checks as the `approve` function.

Update: *Resolved in pull request #31 at commit e12e7ad.*

Low Severity

L-01 Default Admin Role Revocation Bypasses Safety Mechanisms

The protocol implements a two-step process and a time delay for the default admin role transfers to ensure secure governance changes. Furthermore, the `renounceRole` function is designed to revert to prevent the accidental loss of administrative privileges.

However, the `revokeRole` function allows an account with the `REVOKE_ROLE` to execute `revokeRole(0x00, owner)`. This action effectively removes the `DEFAULT_ADMIN_ROLE` from the owner, bypassing the intended two-step transfer process and the time delay. While it is possible for a granter account to re-assign the `DEFAULT_ADMIN_ROLE` if it is currently set to `address(0)`, this behavior undermines the security guarantees provided by the time delay and transfer safeguards.

Consider restricting the `revokeRole` function to prevent the revocation of the `DEFAULT_ADMIN_ROLE`. Alternatively, consider ensuring that such revocations adhere to the same time delay and safety checks as the standard transfer process.

Update: *Resolved in pull request #32 at commit 0c97c9c.*

Notes & Additional Information

N-01 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

`EmergencyOracle.sol` and `AggregatorV3Interface.sol` use the `solidity ^0.8.0` floating pragma directive.

Consider using fixed pragma directives.

Update: *Resolved in pull request #35 at commit b00ade5.*

N-02 Mismatch Between Allowlist Toggle Name and Behavior

The `TokenizationWBase` contract includes a `toggleAllowlist` function that accepts a boolean parameter and assigns it directly to the `_allowlistEnabled` variable. However, the function name and the `AllowlistToggled` event suggest an inversion of the current state (a toggle) rather than a direct assignment (a setter). This discrepancy between naming and behavior can be misleading.

Consider renaming the function to `setAllowlistEnabled` and the event to `AllowlistSet` to accurately reflect the setter semantics. Alternatively, consider implementing a parameter-less function that strictly inverts the state of `_allowlistEnabled`.

Update: *Resolved in pull request #38 at commit d4acd32.*

N-03 Incomplete and Missing Docstrings

Throughout the codebase, multiple instances of incomplete and missing docstrings were identified. The examples include:

`TokenizationWBase.sol`

Functions

- `grantRole`: missing documentation for `role` and `account`
- `revokeRole`: missing documentation for `role` and `account`
- `renounceRole`: missing parameter documentation
- `decimals`: return value not documented

State Variables

- `MINTER_ROLE`
- `GRANT_ROLE`
- `REVOKE_ROLE`
- `PAUSER_ROLE`
- `BURN_ROLE`

- `UPGRADE_ROLE`
- `FINANCIAL_ROLE`
- `COMPLIANCE_ROLE`
- `DENYLISTED_BURN_ROLE`
- `URI_ROLE`
- `ALLOWLIST_ROLE`

`TokenizationWExtension.sol`

Functions

- `getMaxMintBurnLimit`: return value not documented
- `getMaxTransferLimit`: return value not documented
- `getMaxMintBurnValue`: return value not documented
- `getMaxTransferValue`: return value not documented

`EmergencyOracle.sol`

State Variables

- `version`
- `reserveValue`
- `lastUpdate`
- `decimals`
- `description`

Functions

- `getRoundData`: missing return-value documentation.
- `latestRoundData`: missing return-value documentation.

`AggregatorV3Interface.sol`

- `decimals`, `description`, `version`, `getRoundData`, `latestRoundData`: missing parameter and return-value documentation

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #39 at commit d3a3228.*

N-04 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as that function's output. They are an alternative to explicit in-line `return` statements.

Within `EmergencyOracle.sol`, multiple instances of unused named return variables were identified:

- The `roundId` return variable of the `latestRoundData` function
- The `answer` return variable of the `latestRoundData` function
- The `startedAt` return variable of the `latestRoundData` function
- The `updatedAt` return variable of the `latestRoundData` function
- The `answeredInRound` return variable of the `latestRoundData` function

Consider either using or removing any unused named return variables.

Update: *Resolved in pull request #36 at commit 78307b7.*

N-05 Misleading NatSpec in User Status Getters

The NatSpec documentation for `isUserDenylisted` and `isUserAllowlisted` functions incorrectly states that they check the status of "the calling user". However, the actual implementation queries the status of the account provided as an argument.

Consider updating the NatSpec documentation to accurately reflect that the functions query the status of the provided account parameter.

Update: *Resolved in pull request #41 at commit d0f8934.*

N-06 Missing Named Parameters in Mappings

Since Solidity 0.8.18, mappings can include named parameters to provide more clarity about their purpose. Named parameters allow mappings to be declared in the form `mapping(KeyType KeyName? => ValueType ValueName?)`. This feature enhances code readability and maintainability.

Within `TokenizationWBase.sol`, multiple instances of mappings without named parameters were identified:

- The `_denylist` state variable
- The `_allowlist` state variable

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Resolved in pull request #42 at commit 443e61e.*

N-07 Gas Optimizations

Throughout the codebase, multiple opportunities for gas optimizations were identified:

- Superfluous storage read on `$._tokenizationUri` in `setTokenizationUri` function because it is already cached in `newTokenizationUri`
- Superfluous storage read on `$._tokenizationReserveOracle` in `setOracleAddress` function because it is already cached in `newOracleAddress`
- Superfluous storage read on `$._oracleDataHeartbeat` in `setOracleDataHeartbeat` function because it is already cached in `newHeartbeat`
- The `balanceOf` function is called twice in the same `_destroyDenylistedFunds` function
- The `totalSupply` function is called twice in the same `mint` function
- The `version` state variable could be immutable
- The `decimals` state variable could be immutable
- The `revert('Function disabled')` statement could be replaced with a custom error to avoid saving the entire string in the bytecode
- The `revert('Emergency feed: Only latestRoundData() can be used')` statement could be replaced with a custom error to avoid saving the entire string in the bytecode
- The `revert('Function disabled')` statement could be replaced with a custom error to avoid saving the entire string in the bytecode
- Within `TokenizationWBase.sol`, the `newTokenizationUri` parameter could be stored in `calldata` instead of `memory` because it does not change
- The `mint` and `burn` functions could get rid of the `whenNotPaused` modifier because they internally call the `_update` function which already has this modifier.

Consider implementing the aforementioned modifications to achieve gas savings and code improvement.

Update: *Resolved in pull request #37 at commit 491f8ce.*

N-08 Allowlist Toggle Restricted to Compliance Role Contradicts Role Specifications

The project documentation designates the `ALLOWLIST_ROLE` for managing allowlisting functionality, while the `COMPLIANCE_ROLE` is reserved for denylisting and compliance-related tasks. However, the `toggleAllowlist` function of the `TokenizationWBase` contract restricts access to the `COMPLIANCE_ROLE`. This implementation creates a discrepancy between the documented role definitions and the code, as it grants control of the allowlist switch to the compliance role instead of the designated allowlist role. Consequently, operators assigned the `ALLOWLIST_ROLE` are unable to enable or disable the mechanism they are responsible for.

Consider restricting `toggleAllowlist` to the `ALLOWLIST_ROLE` to align with the documented specifications. Alternatively, consider updating the documentation to clarify that global control is intended for the `COMPLIANCE_ROLE`.

Update: *Resolved in pull request #40 at commit e2ad9a5.*

N-09 Configuration Functions Return Incorrect Revert Parameters

The `TokenizationWExtension` contract allows admins to update transaction limits via the `updateMaxAmountMintBurn` and `updateMaxAmountTransfer` functions. When the provided amount exceeds the hardcoded limits, these functions revert using custom errors. However, the implementation passes the limit constants (e.g. MAX_MINT_BURN_LIMIT) to the custom errors instead of the offending amount. This contradicts the error documentation and hinders debugging by obscuring the actual input value that caused the revert.

Consider updating the `revert` statements to pass the amount parameter to the custom errors to accurately reflect the invalid input.

Update: *Resolved in pull request #43 at commit c39f0be.*

N-10 Inconsistent Balance Logging Units

In `upgradeContract`, the account balance is logged in wei units, whereas in other scripts like `deployContract`, the balance is formatted to `ether`.

Consider formatting the output to `ether` to maintain consistency and improve readability.

Update: *Resolved in pull request #44 at commit fb334a3.*

Conclusion

The TokenizationW system introduces an upgradeable ERC-20-based framework designed for the issuance of fully backed digital assets, encompassing stablecoins, real-world assets (RWAs), and digitized securities. By treating each deployment as an independent tokenized asset with distinct operational parameters and reserve backing, the protocol ensures modularity and isolation. Critical to its design is the integration of a Proof-of-Reserve (PoR) mechanism utilizing Chainlink-compatible oracles, ensuring the circulating supply remains strictly validated against verifiable off-chain reserves.

During the audit, one high-severity, one medium-severity, and one low-severity issue were identified. Additionally, several trust assumptions and opportunities to improve code clarity, consistency, and efficiency were noted. Overall, the codebase was found to be well structured and clearly documented, enhancing both auditability and ease of integration.

The Wenia team is commended for their responsiveness and transparency throughout the review process. Their readiness to provide detailed technical explanations, clarify design decisions, and share comprehensive documentation greatly facilitated the assessment and demonstrated a strong commitment to delivering a secure and reliable tokenization framework.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.