IMPORTANT INFORMATION FROM WISDOMTREE DIGITAL

WisdomTree Digital, which operates through one or more subsidiaries of WisdomTree Inc., is providing access to the security audit reports conducted by OpenZeppelin which follow this page. By accessing the accompanying reports, you hereby acknowledge and agree that you have read and understood the contents herein.

Statements in the accompanying reports regarding the design, implementation, integration, suitability or functionality of the token standards framework or one or more smart contracts, such as those related to compliance, fraud prevention, risk management, security, audits, regulatory adherence and operations, address activities, events or developments that generally will or may occur in the future, are based on certain assumptions of the audit firm, are not tailored to any particular entity and no assurances can be made regarding the foregoing, notwithstanding statements in the accompanying reports such as those related to ensuring, enabling, effectiveness, robustness or suitability of certain functionality, operations or otherwise. Actual implementation, integration, suitability or functionality are all highly dependent on potential technical implementation and an issuer's, intermediary's or other applicable market participants' specific considerations, which may include those related to domicile, current or future investor or user base, regulatory requirements and other regulatory considerations, operating, compliance or risk management processes or procedures, desired functionality, technological know-how or comfort, general expectations or predictions, all of which are subject to a number of uncertainties, may change, may not come to fruition or may not work as intended.

In addition, blockchain and related technology, including smart contracts, are a relatively new and untested technology. The findings in the accompanying reports should not be considered a comprehensive list of security or other issues, flaws or defects in the framework, system or codebase. Token frameworks and smart contracts are subject to vulnerabilities and coding errors and no assurance can be made regarding the token standards framework or smart contract suitability, functionality, quality or reliability, even after audit recommendations have been implemented. WisdomTree Digital expressly disclaims all liability arising from the use of this information or reliance on the information provided and to the fullest extent permitted by applicable law, all warranties, whether express or implied, are hereby disclaimed by WisdomTree Digital. Use of this information is at the sole risk of the user. WisdomTree Digital does not provide financial, legal, regulatory or other professional advice.

This disclosure statement and the accompanying reports cannot and do not disclose all the risks and other factors necessary to evaluate a potential product or service from WisdomTree Digital. Before you decide to purchase or use a product or service from WisdomTree Digital or its affiliates, carefully read the specific information regarding such product or service (e.g., prospectus or other disclosure document, as applicable), including applicable risk disclosures.

OpenZeppelin | security

WisdomTree Whitelist Contexts Audit

February 26, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-01-27 To 2025-02-06
Languages	Solidity

Total Issues	20 (15 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (2 resolved)
Notes & Additional Information	17 (13 resolved)

Scope

We audited the wisdomtreeam/whitelist-contexts repository at commit 4476078.

The following files were fully audited:

```
src
├── common
│   ├── access-control
│   │   ├── AccessControl.sol
│   │   └── IAccessControl.sol
│   ├── interfaces
│   │   ├── IBeacon.sol
│   │   └── IERC165.sol
│   └── libraries
│       ├── BytesHelper.sol
│       └── StorageSlot.sol
├── oracles
│   ├── interfaces
│   │   ├── ICompliance.sol
│   │   ├── IOracle.sol
│   │   ├── IOracleBeaconUpgrade.sol
│   │   ├── IOracleInit.sol
│   │   ├── IWhitelistComplianceOracle.sol
│   │   └── IWhitelistOracle.sol
│   └── whitelist-compliance
│       └── WhitelistComplianceOracle.sol
├── proxies
│   ├── Beacon.sol
│   └── Proxy.sol
└── tokens
    ├── ERC721
    │   ├── ERC721BasicToken.sol
    │   └── ERC721SoulboundToken.sol
    └── interfaces
        ├── erc1155
        │   └── IERC1155.sol
        ├── erc721
        │   ├── IERC721.sol
        │   ├── IERC721BeaconUpgrade.sol
        │   ├── IERC721Burnable.sol
        │   ├── IERC721Enumerable.sol
        │   ├── IERC721Errors.sol
        │   ├── IERC721Events.sol
        │   ├── IERC721Metadata.sol
        │   ├── IERC721Mintable.sol
        │   ├── IERC721Receiver.sol
        │   ├── IERC721Token.sol
        │   └── IERC721TokenInit.sol
```

```
        └── erc721Soulbound
            ├── IERC721Soulbound.sol
            ├── IERC721SoulboundBeaconUpgrade.sol
            ├── IERC721SoulboundBurnable.sol
            ├── IERC721SoulboundEnumerable.sol
            ├── IERC721SoulboundErrors.sol
            ├── IERC721SoulboundEvents.sol
            ├── IERC721SoulboundMetadata.sol
            ├── IERC721SoulboundMintable.sol
            ├── IERC721SoulboundReceiver.sol
            ├── IERC721SoulboundToken.sol
            └── IERC721SoulboundTokenInit.sol
```

The following files were diff-audited against the OpenZeppelin library release `v5.0.0`:

```
src
└── common
    └── libraries
        └── Arrays.sol
```

The following files were diff-audited against the OpenZeppelin library release `v4.8.0`:

```
src
└── common
    └── libraries
        ├── Math.sol
        └── Strings.sol
```

Executive Summary

OpenZeppelin was engaged by WisdomTree to conduct a comprehensive audit of their implementation of an oracle whitelist system that allows token standards to authenticate transfers through whitelist membership verification. Our goal was to identify potential vulnerabilities, verify adherence to best practices, and provide recommendations for enhancing the overall security and functionality of the contracts. The audit only reviewed the smart contract components intended to be deployed on-chain and did not review any deployment scripts, configurations, mock contracts, or tests.

The system includes a Whitelist Oracle designed to authorize transfers for token standards such as ERC-20 by verifying whether the recipient holds any token from a specified ERC-721 or ERC-1155 contract or is directly allowed by a custom WhitelistOracle contract. In addition, the system implements two versions of an ERC-721 token. The first, used by the Whitelist Compliance Oracle, is a non-transferable "soulbound" version that can be minted to a single

address and never transferred, designed to be held by whitelisted accounts. The second is a standard implementation that, although not part of the system, serves as a foundation for more complex token designs.

This audit engaged two full-time auditors performing a manual, line-by-line review of the entire codebase as well as the use of automated code analysis for vulnerability detection. In particular, the audit validated that:

- All functionality performs as intended.
- The token contracts conform with the ERC-721 standard.
- The whitelist implementation is robust and cannot be bypassed when properly integrated into a token contract.
- External actors can only interact with the system as intended.
- Best practices have been followed for code style and documentation.

This audit identified 20 issues. In particular, a recommendation was made to make the access control more granular to ensure that each account holding a privileged role is only capable of performing a subset of actions to reduce the overall risk from a compromised account. The remainder of the issues raised were aimed at improving the readability and maintainability of the codebase, addressing minor inconsistencies and redundancies within the code, and reducing gas costs. Overall, the codebase was found to be well-structured and to follow the best practices.

System Overview

The Whitelist Oracle system provides a compliance framework that allows token standards to authenticate transfers through whitelist membership verification. Central to this system is the `WhitelistComplianceOracle` contract, which serves as the core of the system's verification processes. In addition, the project includes two token implementations: a classic ERC-721 token and a non-transferable "Soulbound" ERC-721 token. The ERC-721 standard implementations are adapted from OpenZeppelin's library and have been modified to meet the project's requirements.

Whitelist Compliance Oracle

The `WhitelistComplianceOracle` contract serves as the central compliance verification hub of the Whitelist Oracle system. It implements the `ICompliance` interface, facilitating seamless interaction with a registry of whitelisting contracts. This registry is crucial to the system, as it maintains a comprehensive list of contracts used to authorize token transfers. These contracts can be tokens contracts, such as `ERC721SoulboundToken` and `ERC1155`, which allow a transfer if the receiver holds at least one of the specified tokens, or an external `WhitelistOracle` contract, which can be queried to check the authorization of the transfer.

ERC-721 Soulbound Token

The `ERC721SoulboundToken` is a non-transferable NFT implementation. Its purpose is to define membership and identity within the system. By extending the ERC-721 standard, this token variant introduces a "Soulbound" feature, ensuring that tokens are irrevocably linked to their initial receiving addresses, thereby preventing any form of transfer and enhancing the system's security and compliance capabilities. Such tokens are minted after a user has passed through KYC or AML processes.

ERC-721 Basic Token

Although not actively utilized within the current scope of the Whitelist Oracle system, the `ERC721BasicToken` contract serves as a foundational pillar, offering essential ERC-721

functionalities complemented by role-based access control and upgradeability features. This standard implementation lays the groundwork for more complex token implementations, offering role-based access control and an upgradeability pattern as well.

Access Control

All components inherit the `AccessControl` contract which defines specific roles for each contract within the system, ensuring a granular level of control and security. Furthermore, the adoption of the Beacon Proxy pattern for atomic upgrades and the Storage Slots pattern to prevent upgrade collisions ensures the system's longevity and adaptability.

Security Model and Trust Assumption

The Whitelist Oracle system implements multiple privileged roles through the `AccessControl` contract. The holders of such roles are expected to be non-malicious and to act in the project's best interest.

Privileged Roles

The system introduces several distinct privileged roles, each with specific responsibilities and capabilities:

- `DEFAULT_ADMIN_ROLE`:
  - Assigned during the initial contract setup.
  - Empowered to assign roles to other participants and initiate upgrades to beacons.
  - This role is exclusively reserved for a maximum of three addresses.
- `DELEGATED_ADMIN_ROLE`:
  - This role is governed by both `DEFAULT_ADMIN_ROLE` and `DELEGATED_ADMIN_ROLE` holders.
  - Admin role for `ISSUER_ROLE` and `REGISTRAR_ROLE` roles.
  - When a `DEFAULT_ADMIN_ROLE` role is rescinded, all of its delegated roles are simultaneously revoked.

- `REGISTRAR_ROLE`:
  - This role is governed by both `DEFAULT_ADMIN_ROLE` and `DELEGATED_ADMIN_ROLE` holders.
  - Authorized to oversee token-related activities, including minting or burning tokens.
- `ISSUER_ROLE`:
  - This role is governed by both `DEFAULT_ADMIN_ROLE` and `DELEGATED_ADMIN_ROLE` holders.
  - Allocated for functionalities yet to be introduced.

The system also implements a mechanism to prevent the loss of access to the `DEFAULT_ADMIN_ROLE` by explicitly requiring that at least one address always holds the role.

Low Severity

L-01 Lack of Granular Permission Control in `WhitelistComplianceOracle`

The `WhitelistComplianceOracle` contract inherits the `AccessControl` contract but relies solely on the owner with the `DEFAULT_ADMIN_ROLE` to manage the contract. This role has the ability to upgrade the beacon address, add and remove address contexts, enable and disable the oracle, and set the maximum number of contract contexts.

Consider implementing more granular access controls.

Update: *Acknowledged, not resolved. The team stated:*

> *Finding acknowledged. The restriction on the `WhitelistComplianceOracle` to only be managed by the `DEFAULT_ADMIN_ROLE` is intentional by design, this is an admin-level oracle that is only modifiable by admins within our protocol.*

L-02 Missing Event Emission

The `setBaseURI` and `setTokenURI` functions of the `ERC721SoulboundToken` contract do not emit events. This could make it challenging for off-chain applications to track changes in the contract.

Consider emitting events when updating the base URI or token URI.

Update: *Resolved in pull request #5 at commit 9d6b857. The team stated:*

> *Implemented auditors' recommended fix by adding event emits to the `setBaseURI` and `setTokenURI` functions of the `ERC721SoulboundToken` contract.*

L-03 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled. Throughout the codebase, multiple instances of floating pragma directives (`^0.8.19`) being used were identified.

Consider using fixed pragma directives.

Update: *Resolved in pull request #6 at commit 7fefad4. The team stated:*

> *Implemented auditors' recommended fix by changing pragma directives from floating to fixed across the codebase.*

Notes & Additional Information

N-01 Inconsistent Access Control When Granting The Delegated Admin Role

The `grantDelegateAdminRole` function of the `AccessControl` contract can be executed by `DEFAULT_ADMIN_ROLE` or `DELEGATED_ADMIN_ROLE`. However, the batch version of the same functionality, the `batchGrantDelegateAdminRole` function, can be only executed by the `DEFAULT_ADMIN_ROLE`.

Consider adjusting the access control for the `batchGrantDelegateAdminRole` function.

Update: *Acknowledged, not resolved. The team stated:*

> *This finding is acknowledged. For now, the restriction of the `batchGrantDelegateAdminRole` to only the `DEFAULT_ADMIN_ROLE` will remain because we want to align all of our AccessControl implementations across our protocol. In the future, once our protocol is under scope for an AccessControl upgrade, we can explore modifying this restriction.*

N-02 Redundant Code

Multiple instances of redundant or unnecessary code were identified throughout the codebase:

- The `upgradeBeaconToAndCall` function of the `WhitelistComplianceOracle`, `ERC721BasicTokens`, and `ERC721SoulboundToken` contracts checks whether the beacon address, passed as `newBeacon` argument, is not `address(0)`. However, the `_setBeacon` function checks for that as well, making the first `require` statement

redundant. Consider removing the `require` statements within the `upgradeBeaconToAndCall` functions for those contracts.

- The implementation of the `supportsInterface` function in `ERC721BasicToken`, `ERC721SoulboundToken`, and `WhitelistComplianceOracle` returns `false` regardless of the result of the interface ID comparison with `0xffffffff`. Consider simplifying by directly returning false.

Consider implementing the above code changes to improve code clarity and eliminate redundancies.

Update: *Resolved in pull request #7 at commit 49814d1 and commit 9fecb7b. The team stated:*

> *Implemented auditors' recommended fixes by reducing redundancy in `upgradeBeaconToAndCall` and `supportInterface` functions of `WhitelistComplianceOracle`, `ERC721BasicToken`, and `ERC721SoulboundToken` contracts.*

N-03 Using `int/uint` Instead of `int256/uint256`

Throughout the codebase, multiple instances of `int/uint` being used instead of `int256/uint256` were identified:

- In lines 23, 31, 50, and 60 of `IWhitelistComplianceOracle.sol`
- In line 101 of `StorageSlot.sol`
- In lines 44, 170, 213, 221, 297, 299, 303, 317, and 369 of `WhitelistComplianceOracle.sol`

In favor of explicitness, consider replacing all instances of `int/uint` with `int256/uint256`.

Update: *Resolved in pull request #8 at commit 17c7c0a. The team stated:*

> *Implemented auditors' recommendation by replacing all instances of `int/uint` with `int256/uint256`.*

N-04 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability

in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the @custom:security-contact convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #8 at commit 7958ca9. The team stated:*

> *Implemented auditors' recommendations by adding a NatSpec comment containing a security contact above each contract definition.*

N-05 Duplicate Imports

Throughout the codebase, multiple instances of duplicate imports were identified:

- The `IERC721SoulboundToken.sol` file imports `IERC721SoulboundEvents.sol` which is already imported as a result of importing `IERC721Soulbound.sol`.
- The `IERC721Token.sol` file imports `IERC721Events.sol` which is already imported as a result of importing `IERC721.sol`.

Consider removing any duplicate imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #9 at commit b8bd4e8. The team stated:*

> *Implemented auditors' recommendations by removing duplicate imports of various interfaces across the codebase.*

N-06 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

Throughout the codebase, multiple instances of `revert` and/or `require` messages were identified:

- In lines 20 and 30 of `Beacon.sol`
- In lines 97, 119, 120, 121, 122, 124, 129, 162, 170, 199, 229, 764, and 889 of `ERC721BasicToken.sol`
- In lines 91, 113 114 115, 116, 118, 123, 156, 164, 227, 235, 304, 316, 329, 341, 353, 366, 380, 563, and 713 of `ERC721SoulboundToken.sol`
- In lines 87, 100, 115, 116, 118, 119, 151, 159, 172, 173, 174, 179, 185, 191, 198, 215, 236, 253, 257, 286, and 359 of `WhitelistComplianceOracle.sol`

For conciseness and gas savings, consider replacing `require` and `revert` messages with custom errors.

Update: *Resolved in pull request #10 at commit 521ea7f and commit a80fbd3. The team stated:*

> *Implemented auditors' recommendations by replacing `require` and `revert` messages with custom errors across the codebase.*

N-07 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `Beacon.sol`, the `Upgraded` event, the `implementation` function, and the `upgradeTo` function
- In `ERC721SoulboundToken.sol`, the `safeMint` function and the `burn` function
- In `IOracle.sol`, the `OracleDisabled` event, the `OracleEnabled` event, the `IOracleBeaconUpgrade` interface, and the `upgradeBeaconToAndCall` function

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 #11 at commit 6c4e527. The team stated:*

> *Implemented auditors' recommendations by adding missing docstrings using natspec across codebase.*

N-08 Incorrect Docstrings

Throughout the codebase, multiple instances of incorrect docstrings were identified:

- The comments in the `IERC721Soulbound` interface are incorrect as they reference a standard ERC-721 token contract instead of the intended soulbound token implementation.
- The comments for the `initializeWithRoles` function within the `IERC721TokenInit` and `IERC721SoulboundTokenInit` interfaces are incorrect. Specifically, they assert that the `ISSUER_ROLE` is designated for the issuance of new tokens, which is misleading and not the case.

Consider thoroughly documenting all functions that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #12 at commit 86ef758. The team stated:*

> *Implemented auditors' recommendations by updating incorrect docstrings with correct natspec.*

N-09 Predictable Storage Slot Preimage

The contracts store state variables using the `StorageSlot` library, which stores values in unique slots. The issue with this approach is that the slot has a known preimage for the used `keccak256`. This could lead to problems if the contracts utilizing the `StorageSlot` library grow in functionality and complexity, and contain issues that allow the attacker to write to a storage slot of their choosing.

Consider subtracting 1 from the resulting `keccak256` value.

Update: *Acknowledged, not resolved. The team stated:*

> *Acknowledged, we understand this is a reasonable precaution for future codebases/ modifications, however, the existing logic will no longer access the previously stored data if we upgrade our existing implementation contracts to use this new slot calculation. Unless we manually migrate the state from the old slots to the new ones, our Proxy contracts will effectively "lose" the stored data. While the intent is to make it computationally infeasible for an attacker to determine a slot's preimage, doing so in an upgrade would break backward compatibility within our protocol at this time. In our current state, no logic allows writing to these slots maliciously, thus this risk is very low. Switching the existing scheme for our protocol would be a breaking change with a*

> *significant risk of losing state data unless we carefully plan and execute a storage migration.*

N-10 Token URL Cannot Be Set for Non-Existing Tokens

The `setTokenURI` functions of the `ERC721BasicToken` and `ERC721SoulboundToken` contracts do not allow setting the URI for a non-existent `tokenId`. This prohibits using these contracts with a common use case in which `tokenId` metadata is set even before minting the corresponding token.

Consider lifting this restriction to not limit the flexibility of the `ERC721BasicToken` and `ERC721SoulboundToken` contracts.

Update: *Acknowledged, not resolved. The team stated:*

> *We acknowledge this finding. This restriction was intentional, it is a functionality we don't need to utilize in our current product scope and can be upgraded as required.*

N-11 Discrepancies Between Interfaces and Contracts

Throughout the codebase, multiple instances of discrepancies between interfaces and implementation contracts were identified:

- In the `IERC721` interface, the `setTokenURI` function uses the parameter name `tokenURI`, whereas, the implementation uses `tokenURI_`.
- In the `IERC721` interface, the `setBaseURI` function uses the parameter name `baseURI`, whereas, the implementation uses `baseURI_`.
- In the `IERC721Soulbound` interface, the `setTokenURI` function uses the parameter name `tokenURI`, whereas, the implementation uses `tokenURI_`.
- In the `IERC721Soulbound` interface, the `setBaseURI` function uses the parameter name `baseURI`, whereas, the implementation uses `baseURI_`.
- In the `IERC165` interface, the `supportsInterface` function uses the parameter name `interfaceId`, whereas, the `WhitelistComplianceOracle`, `ERC721BasicToken`, and `ERC721SoulboundToken` implementation contracts use `interfaceID`.

Consider aligning parameter names and storage location keywords between interfaces and their implementation contracts to ensure consistency and reduce potential errors.

Update: *Resolved in pull request #13 at commit 9ccc869. The team stated:*

> *Implemented auditors' recommendations by aligning parameter names and storage keywords in interfaces with their implementation contracts.*

N-12 Non-Explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity file or when inheritance chains are long.

Throughout the codebase, multiple instances of non-explicit imports were identified:

- The import in `Beacon.sol`, `ERC721BasicToken.sol`, `ERC721SoulboundToken.sol`, and `WhitelistComplianceOracle.sol`
- The imports in `IERC721Token.sol`, `IERC721SoulboundToken.sol`, and `IWhitelistComplianceOracle.sol`
- The imports in `IERC1155.sol`, `IERC721.sol`, `IERC721Enumerable.sol`, `IERC721Metadata.sol`, `IERC721Soulbound.sol`, `IERC721SoulboundEnumerable.sol`, `IERC721SoulboundMetadata.sol`, and `IWhitelistOracle.sol`

Following the principle that clearer code is better code, consider using the named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #14 at commit 906e1b2 and commit 32317ad. The team stated:*

> *Implemented auditors' recommendations by converting imports into more explicitly named imports.*

N-13 Unused `internal` Functions

Throughout the codebase, multiple instances of unused `internal` functions were identified:

- The `_increaseBalance` and `_increaseBalanceVanila` functions in `ERC721BasicToken.sol`
- The `_increaseBalance` and `_increaseBalanceVanila` functions in `ERC721SoulboundToken.sol`

To improve the overall clarity, intentionality, and readability of the codebase, consider using or removing any currently unused functions.

Update: *Acknowledged, not resolved. The team stated:*

> *Acknowledged. The `_increaseBalance` and `_increaseBalanceVanila` functions are internal utility functions that are part of the ERC721 token implementation pattern. They serve specific purposes: `_increaseBalance` is a safety wrapper that prevents batch minting in enumerable tokens. `_increaseBalanceVanila` is the actual balance increase implementation These functions are actually used indirectly through the `_update` and `_updateVanila` functions, which are called during minting and transferring operations. However, they are called through a different code path than direct usage. Public functions like `safeMint`, `transferFrom` call `_update` which calls `_updateVanila` that modifies balances directly using similar logic as `_increaseBalanceVanila`. The reason these functions exist is to provide a hook point for extensions that "mint" tokens using an `ownerOf` override. They are currently not being used within the current implementation but we would like to keep them as they provide extension points for future functionality. We have added extra natspec to these functions explaining this in pull request #15.*

N-14 Inconsistent Order Within Contracts

Throughout the codebase, multiple instances of contracts having an inconsistent ordering of functions were identified:

- The `Beacon` contract
- The `WhitelistComplianceOracle` contract

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

Update: *Resolved in pull request #16 at commit 6b471d6. The team stated:*

> *Implemented auditors' recommended fixes by re-ordering functions more closely to the Solidity Style Guide.*

N-15 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions with unnecessarily permissive visibility were identified:

- The `implementation` function in `Beacon.sol` with `public` visibility could be limited to `external`.
- The `setBaseURI`, `setTokenURI`, `approve`, `setApprovalForAll`, `safeTransferFrom`, `safeMint`, `batchSafeMint`, `burn`, `batchBurn`, `ownerOf`, `tokenOfOwnerByIndex`, `tokenByIndex`, `getApproved`, and `supportsInterface` functions in `ERC721BasicToken.sol` with `public` visibility could be limited to `external`.
- The `safeMint`, `burn`, `batchSafeMint`, `batchBurn`, `setBaseURI`, `setTokenURI`, `tokenURI`, `ownerOf`, `tokenOfOwnerByIndex`, `tokenByIndex`, `safeTransferFrom`, `approve`, `setApprovalForAll`, `getApproved`, `isApprovedForAll`, `transferFrom`, `safeTransferFrom`, and `supportsInterface` functions in `ERC721SoulboundToken.sol` with `public` visibility could be limited to `external`.
- The `supportsInterface` function in `WhitelistComplianceOracle.sol` with `public` visibility could be limited to `external`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

Update: *Resolved in pull request #17 at commit e69bc2d. The team stated:*

> *Implemented auditors' recommended fix by changing function visibility to be only as permissive as required across the codebase.*

N-16 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In `ERC721BasicToken.sol`:
  - For the `name`, `symbol`, `baseURI`, `tokenURI`, `totalSupply`, `getApproved`, and `isApprovedForAll` functions, not all return values are documented.
  - For the `tokenURI`, `setTokenURI`, `approve`, `transferFrom`, `getApproved`, `setApprovalForAll`, `batchSafeMint`, `batchBurn`, and `isApprovedForAll` functions, the parameters are not documented.
- In `ERC721SoulboundToken.sol`:
  - For the `name` and `symbol` functions, the return values are not documented.
  - For the `batchSafeMint`, `batchBurn`, `setBaseURI`, `setTokenURI`, `tokenURI`, and `setTokenURI` functions, the parameters are not documented.
- In `WhitelistComplianceOracle.sol`:
  - For the `canTransfer` function, not all return values are documented.
  - For the `addContractAddress`, `removeContractAddress`, and `canTransfer` functions, the parameters are not documented.
- In `IERC1155.sol`:
  - For the `setApprovalForAll`, `safeTransferFrom`, `safeBatchTransferFrom`, `balanceOf`, `balanceOfBatch`, and `isApprovedForAll` functions, the parameters and return values are not documented.
- In `IERC721.sol`:
  - For the `balanceOf`, `ownerOf`, `getApproved`, and `isApprovedForAll` functions, the return values are not documented.
  - For the `safeTransferFrom`, `safeTransferFrom`, `transferFrom`, `approve`, `setApprovalForAll`, `balanceOf`, `ownerOf`, `getApproved`, and `isApprovedForAll` functions, the parameters are not documented.

- In `IERC721Enumerable.sol`:
  - For the `totalSupply`, `tokenOfOwnerByIndex`, and `tokenByIndex` functions, the return values are not documented.
  - For the `tokenOfOwnerByIndex` and `tokenByIndex` functions, the parameters are not documented.
- In `IERC721Events.sol`:
  - For the `Transfer`, `Approval`, `ApprovalForAll`, `BaseURIUpdated`, and `TokenURIUpdated` events, the parameters are not documented.
- In `IERC721Metadata.sol`:
  - For the `name`, `symbol`, and `baseURI` functions, not all return values are documented.
  - For the `tokenURI` function, the parameter is not documented.
- In `IERC721Receiver.sol`:
  - For the `onERC721Received` function, the parameters and return value are documented.
- In `IERC721Soulbound.sol`:
  - For the `balanceOf` and `ownerOf` functions, not all return values are documented.
  - For the `safeTransferFrom`, `safeTransferFrom`, `balanceOf`, `ownerOf`, `transferFrom`, `approve`, `getApproved`, `setApprovalForAll`, and `isApprovedForAll` functions, the parameters are not documented.
- In `IERC721SoulboundEnumerable.sol`:
  - For the `totalSupply` and `tokenOfOwnerByIndex` functions, not all return values are documented.
  - For the `tokenOfOwnerByIndex` and `tokenByIndex` functions, the parameters are not documented.
- In `IERC721SoulboundEvents.sol`:
  - The parameters of `Transfer`, `Approval`, and `ApprovalForAll` events are not documented.

- In `IERC721SoulboundMetadata.sol`:
  - For the `name`, `symbol`, and `tokenURI` functions, not all return values are documented.
  - For the `tokenURI` function, the parameter is not documented.
- In `IERC721SoulboundReceiver.sol`:
  - For the `onERC721Received` function, the parameters and the return value are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #18 at commit ec85574. The team stated:*

> *Implemented auditors' recommended fix by adding more comprehensive and complete docstrings using NatSpec across the codebase.*

N-17 Gas Optimizations

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

- The `upgradeBeaconToAndCall` functions within the `ERC721BasicToken`, `ERC721SoulboundToken`, and `WhitelistComplianceOracle` contracts read the beacon address from `_BEACON_SLOT` storage after setting its value to `newBeacon`. This results in an unnecessary storage read. Instead, `newBeacon` can be used to retrieve the implementation address directly.
- Consider using the prefix-increment operator (`++i`) instead of the postfix-increment operator (`i++`) in order to save gas. This optimization skips storing the value before the incremental operation, as the return value of the expression is ignored.
  - Within the `batchSafeMint` and `batchBurn` functions of the `ERC721BasicToken` contract.
  - Within the `batchSafeMint` and `batchBurn` functions of the `ERC721SoulboundToken` contract.
  - In the `i++`, `j++`, and `i++` increments of the `removeContractAddress`, `isAddressWhitelisted`, and `getContractAddresses` functions in `WhitelistComplianceOracle.sol`.

Update: *Resolved in pull request #19 at commit 554ed97. The team stated:*

> *Implemented auditors' recommendations by applying gas optimization by reducing direct storage read and using a prefix-increment operator for loops.*

Conclusion

The Whitelist Oracle system provides a compliance framework that enables token standards such as ERC-721 and ERC-1155, as well as external oracle contracts, to authenticate transfers. During the audit, several minor issues were identified, and recommendations for improving code consistency, readability, and gas efficiency were accordingly made.

In addition, it was noted that during the audit, the branch coverage for the codebase was approximately 58%, with the coverage of key components (ERC-721 Tokens and Whitelist Compliance Oracle) ranging between 60% and 68%. It was recommended to strengthen the test suite and incorporate fuzz testing to enhance the maturity of the codebase. After the audit, the team significantly improved the test suite and reached branch coverage of 80-82% for the core components. The WisdomTree team was highly cooperative and provided clear explanations throughout the audit process.

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.