WisdomTree Prime™

WISDOMTREE DIGITAL TOKEN STANDARDS V3 FRAMEWORK SECURITY AUDIT REPORT

OXORIO

AUGUST 23, 2024

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the security audit conducted by Oxorio for WisdomTree Digital's Token Standards v3 framework.

WisdomTree Digital leverages its principles of innovation and responsible DeFi to lead in the digital asset and blockchain space through advanced technology, while embracing regulation, transparency, accessibility, security, reliability, and liquidity.

WisdomTree Digital's Token Standards v3 framework is designed to integrate compliance functionalities directly into digital tokens. It allows issuers to embed rule sets that automate multi-jurisdictional compliance, fraud prevention, and other risk management processes. The Token framework supports real-time compliance through smart contracts and a compliance oracle, ensuring all token operations meet regulatory standards. This system enhances security, reduces manual compliance efforts, and facilitates seamless auditing and regulatory adherence across various jurisdictions.

The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the framework's security and functionality. The audit covered a total of 39 smart contracts, encompassing 1499 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with WisdomTree Digital and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

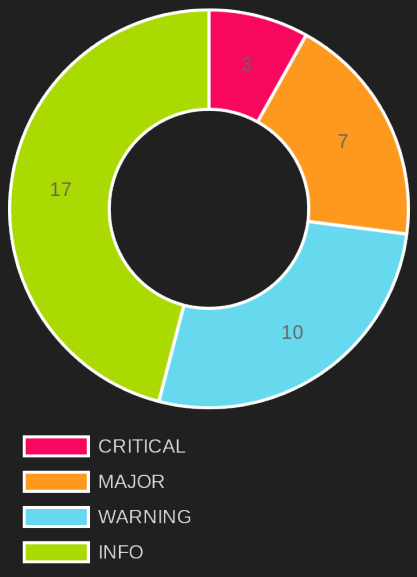
Throughout the audit, we maintained a collaborative approach with WisdomTree, ensuring that all concerns within the audit's scope were thoroughly addressed. Each issue was either resolved or formally acknowledged by WisdomTree. However, new issues emerged during the verification of the implemented fixes, which have been documented and require further attention. As a result, we cannot fully confirm that the framework, as of audited commit `5ed116a1a5d6b071b9b7f85038389c81fe00eeb6`, meets all the security and functionality requirements established for this audit, based on the code and documentation provided. We recommend an additional re-audit to address these concerns and enhance the framework's overall security and reliability.

1.2 SUMMARY OF FINDINGS

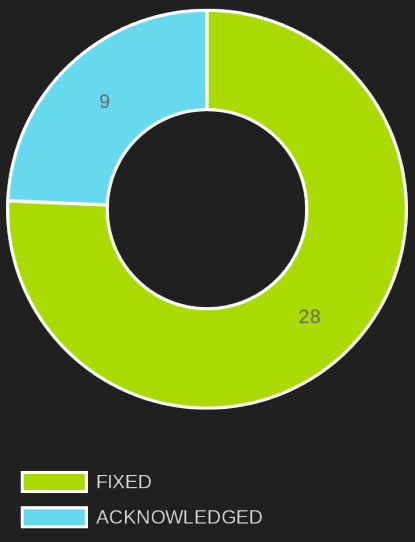
The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Finding Report section for further reference.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**3**	0	3	0	0
MAJOR	**7**	0	5	2	0
WARNING	**10**	0	7	3	0
INFO	**17**	0	13	4	0
TOTAL	**37**	**0**	**28**	**9**	**0**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is," without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	WisdomTree Digital
Project name	Token Standards v3
Category	Token Framework
Repository	https://bitbucket.org/wisdomtreeam/tokenstandardsv3
Documentation	WT Roles & Key Management Memo
Initial Commit	`3c38613cf25424ec0f08263c3fc21c9020ab8661`
Reaudited Commit	`5ed116a1a5d6b071b9b7f85038389c81fe00eeb6`
Platform	L1
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Nataly Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
June 6, 2024	Client engaged Oxorio to request an audit.
July 17, 2024	The audit team initiated work on the project.
July 29, 2024	Preliminary report for Round 1 audit was submitted.
July 31, 2024	Comprehensive report for Round 1 audit was submitted.
August 12, 2024	Client's feedback on the report was received.
August 14, 2024	The audit team commenced the re-audit of the project.
August 23, 2024	Final report for Round 1 audit, incorporating client's verified fixes, was submitted.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	src/common/access-control/AccessControl.sol	398	35	169	**194**	15
2	src/common/access-control/IAccessControl.sol	137	13	109	**15**	0
3	src/common/libraries/Arrays.sol	127	16	51	**60**	25
4	src/common/libraries/BytesHelper.sol	146	17	43	**86**	37
5	src/common/libraries/Context.sol	24	3	12	**9**	0
6	src/common/libraries/Math.sol	181	10	44	**127**	24
7	src/common/libraries/RoleAgencyLib.sol	22	2	1	**19**	26
8	src/common/libraries/StorageSlot.sol	135	15	59	**61**	13
9	src/common/libraries/Strings.sol	71	7	19	**45**	20
10	src/proxies/Beacon.sol	42	9	5	**28**	7
11	src/proxies/Proxy.sol	95	14	5	**76**	8
12	src/tokens/common/BaseERC20.sol	324	46	119	**159**	14
13	src/tokens/interfaces/erc20/IERC20.sol	85	9	60	**16**	0
14	src/tokens/interfaces/erc20/IERC20BatchBasic.sol	44	3	35	**6**	0
15	src/tokens/interfaces/erc20/IERC20BatchClawback.sol	26	1	17	**8**	0
16	src/tokens/interfaces/erc20/IERC20BatchFreeze.sol	27	2	20	**5**	0
17	src/tokens/interfaces/erc20/IERC20Burnable.sol	15	1	10	**4**	0
18	src/tokens/interfaces/erc20/IERC20Clawback.sol	20	1	11	**8**	0
19	src/tokens/interfaces/erc20/IERC20ClawbackEvents.sol	12	1	7	**4**	0
20	src/tokens/interfaces/erc20/IERC20Events.sol	18	2	11	**5**	0
21	src/tokens/interfaces/erc20/IERC20Freeze.sol	25	3	16	**6**	0
22	src/tokens/interfaces/erc20/IERC20FreezeEvents.sol	17	2	10	**5**	0
23	src/tokens/interfaces/erc20/IERC20Mintable.sol	19	2	12	**5**	0
24	src/tokens/interfaces/erc20/IERC20Pausable.sol	22	3	13	**6**	0
25	src/tokens/interfaces/erc20/IERC20PausableEvents.sol	17	2	10	**5**	0
26	src/tokens/interfaces/erc20/IERC20RevocableCompliance.sol	33	5	19	**9**	0
27	src/tokens/interfaces/erc20/IERC20Token.sol	39	3	4	**32**	0
28	src/tokens/interfaces/erc20/IERC20WhitelistContextControlled.sol	86	10	58	**18**	0
29	src/tokens/interfaces/erc20/IERC20WhitelistContextRevocable.sol	118	12	78	**28**	0
30	src/tokens/interfaces/ICompliance.sol	26	1	17	**8**	0
31	src/tokens/interfaces/IContext.sol	27	5	13	**9**	0
32	src/tokens/interfaces/IContextFactory.sol	46	7	25	**14**	0
33	src/tokens/interfaces/IController.sol	51	9	31	**11**	0
34	src/tokens/interfaces/IERC165.sol	14	1	9	**4**	0
35	src/tokens/interfaces/IRoleAgency.sol	44	5	31	**8**	0
36	src/tokens/standards/ERC20BasicStandard.sol	65	8	14	**43**	21
37	src/tokens/standards/ERC20ControlledStandard.sol	315	36	100	**179**	11

	File	Lines	Blanks	Comments	Code	Complexity
38	src/tokens/standards/ERC20RevocableComplianceStandard.sol	222	23	59	**140**	17
39	src/tokens/standards/ERC20RevocableStandard.sol	56	7	15	**34**	21
	Total	**3191**	**351**	**1341**	**1499**	**15**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

WisdomTree Digital's Token Standards v3 framework is designed to integrate compliance functionalities directly into digital tokens. It allows issuers to embed rule sets that automate multi-jurisdictional compliance, fraud prevention, and other risk management processes. The Token framework supports real-time compliance through smart contracts and a compliance oracle, ensuring all token operations meet regulatory standards. This system enhances security, reduces manual compliance efforts, and facilitates seamless auditing and regulatory adherence across various jurisdictions.

All token standards are built on the `BaseERC20` standard, which provides the basic functionality of an ERC-20 token including features like minting, burning, transfers, and allowances. This standard ensures that all tokens adhere to the ERC-20 standard, which is a widely adopted standard for fungible tokens on the Ethereum blockchain. It serves as the foundation for more advanced token standards by ensuring basic functionality and security in token transactions.

The ERC20BasicStandard builds on the foundational features of the `BaseERC20`, which include minting, burning, transfers, and allowances. In addition to these basic operations, this standard introduces batch processing capabilities for minting, burning, and transfers. These enhancements enable the efficient handling of multiple operations in a single transaction, making the standard particularly well-suited for larger-scale operations where transaction throughput is a priority.

The `ERC20ControlledStandard` inherits all the functionalities of the ERC20BasicStandard, including minting, burning, batch operations, transfers, and allowances. It further extends these capabilities by introducing control features such as pausing, unpausing, freezing, and unfreezing of tokens. These added functions give issuers the ability to manage token circulation more effectively, allowing them to temporarily halt operations or restrict access to tokens under specific conditions, such as during a security breach or when required for regulatory compliance.

The `ERC20RevocableStandard` includes all the functionalities of the `ERC20ControlledStandard`, incorporating minting, burning, batch processing, transfers, allowances, pausing, unpausing, freezing, and unfreezing. Additionally, this standard introduces the ability to perform clawback operations, both individually and in batches. This critical feature allows issuers to revoke tokens under specific circumstances, such as compliance violations or fraud, providing an extra layer of security and control over token management.

The `ERC20RevocableComplianceStandard` builds upon the `ERC20RevocableStandard`, encompassing all inherited functionalities, including minting, burning, batch operations, transfers, allowances, pausing, unpausing, freezing, unfreezing, and clawback features. This

most advanced standard integrates compliance checks directly into token operations, ensuring that every transaction automatically adheres to the relevant regulatory requirements. This ensures that all transactions adhere to specified regulatory requirements, leveraging the token framework. It automates compliance, making the system robust and suitable for multi-jurisdictional operations.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	The project employs a role-based access control system; however, the centralization around the `REGISTRAR_ROLE` introduces critical vulnerabilities. Specific issues identified include `C-01`, `C-03`, `C-04`, and `M-06`. Immediate remediation is required to prevent potential exploits.	**Poor**
Arithmetic	The project implements standard token arithmetic and does not involve complex mathematical operations, thereby mitigating the risks of overflows and underflows inherent in earlier versions of Solidity. The arithmetic operations are simple and primarily involve basic calculations for token transfers and balance updates.	**Excellent**
Complexity	Despite the small codebase, certain constructs appear unnecessarily complex and suboptimal, particularly the logic in `M-04` and `W-04`. These complexities could potentially lead to maintenance challenges and excessive gas usage. Additionally, there is noticeable code duplication, increasing the likelihood of inconsistencies and making the codebase more difficult to maintain and refactor.	**Fair**
Data Validation	The project implements data validation techniques; however, the validation logic could be improved. Notable issues include `M-01`, `W-11` and `I-05`, which require attention to ensure robust data integrity.	**Fair**
Decentralization	The project's control is highly centralized, with the administrator and associated roles having extensive control over the system.	**Not Applicable**

Category	Assessment	Result
Documentation	The project documentation is minimal and does not sufficiently describe the business logic or the intricacies of the system. The documentation is currently provided only in the form of a README file. Improved and expanded documentation is necessary to facilitate better understanding and maintenance.	**Poor**
External Dependencies	The project relies on an external Compliance Oracle for critical operations. However, the implementation of this dependency is flawed, as detailed in issue `C-02`. This external dependency should be reviewed and correctly implemented.	**Absent**
Error Handling	The project predominantly uses `require` statements for error handling, providing clear and descriptive error messages. Nevertheless, attention should be given to issues `W-03` and `I-08` to enhance error handling robustness.	**Good**
Logging and Monitoring	The project includes sufficient event logging mechanisms to track system operations. All state-changing functions emit events, and custom errors are used to signal specific reasons for reverts. However, improvements are needed to address event logging-related issues, specifically `W-08`.	**Good**
Low-Level Calls	The codebase utilizes `delegateCall` for the implementation of an upgradeable proxy pattern. This low-level call is properly handled, ensuring safe and efficient upgradeability.	**Excellent**
Testing and Verification	The project has a limited suite of unit and integration tests, achieving a code coverage of `50.92%`. Several critical components, including proxy handling, are insufficiently tested. Enhancing the test coverage and incorporating comprehensive test scenarios is recommended.	**Fair**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
src/tokens/common/BaseERC20.sol	**11**	0	1	2	8
src/common/access-control/AccessControl.sol	**8**	2	0	4	2
src/tokens/standards/ERC20RevocableComplianceStandard.sol	**7**	1	2	2	2
src/tokens/standards/ERC20ControlledStandard.sol	**6**	0	3	2	1
src/proxies/Beacon.sol	**4**	0	1	1	2
src/proxies/Proxy.sol	**3**	0	0	2	1
src/tokens/standards/ERC20BasicStandard.sol	**3**	0	1	2	0
src/common/libraries/RoleAgencyLib.sol	**2**	0	0	0	2
src/tokens/interfaces/erc20/IERC20WhitelistContextControlled.sol	**2**	0	0	0	2
src/tokens/interfaces/erc20/IERC20WhitelistContextRevocable.sol	**2**	0	0	0	2
src/tokens/standards/ERC20RevocableStandard.sol	**2**	0	1	1	0
src/common/libraries/Strings.sol	**1**	0	0	0	1
src/src/common/access-control/AccessControl.sol	**1**	0	1	0	0
src/tokens/interfaces/ICompliance	**1**	1	0	0	0
src/tokens/interfaces/IContext.sol	**1**	0	0	0	1
src/tokens/interfaces/IContextFactory.sol	**1**	0	0	0	1
src/tokens/interfaces/IController.sol	**1**	0	0	0	1
src/tokens/interfaces/IRoleAgency.sol	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was performed on 39 smart contracts, initially uncovering 3 critical and 7 major issues, in addition to several warnings and informational notes. The audit highlighted various attack vectors and potential vulnerabilities, with significant findings related to code optimization, system access controls, use of external dependencies, and role and permissions logic.

Following our initial audit, WisdomTree Digital collaborated closely with our team to resolve the identified issues. The recommended changes were designed to align with best practices, minimize the potential attack surface, simplify code maintenance, and improve readability. Following multiple rounds of interaction, all identified issues have been either resolved or formally acknowledged.

However, during the verification of the implemented fixes, new issues related primarily to access control emerged as a result of the changes made to address the findings outlined in this report. These issues have been documented in a separate report. As a result, we cannot fully confirm that the framework, as of audited commit `5ed116a1a5d6b071b9b7f85038389c81fe00eeb6`, meets all the security and functionality requirements established for this audit, based on the code and documentation provided. We recommend conducting an additional round of re-audit to address these newly discovered issues and to increase test coverage, as doing so would help ensure that similar issues are identified and resolved more effectively in the future.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

C-01	Elevated Permissions Risk in `AccessControl` for `ISSUER_ROLE` and `REGISTER_ROLE`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl`	143

Description

In the contract `AccessControl` the function `shareRegistrarRole` allows a user with `ISSUER_ROLE` and `REGISTER_ROLE` to assign their additional address with `REGISTER_ROLE` rights. This circumvents the intended restriction that only a user with `DELEGATED_ADMIN_ROLE` can assign these roles, as defined in the `BaseERC20` contract:

```
_setRoleAdmin(ISSUER_ROLE, DELEGATED_ADMIN_ROLE);
_setRoleAdmin(REGISTRAR_ROLE, DELEGATED_ADMIN_ROLE);
```

This design flaw means that an attacker compromising a user with `ISSUER_ROLE` and `REGISTER_ROLE` can assign `REGISTER_ROLE` to another address and gain comprehensive control over the system. The `REGISTER_ROLE` allows the execution of critical functions such as `pause`, `unpause`, `mint`, `batchMint`, `burn`, `batchBurn`, `freeze`, `unfreeze`, `setComplianceOracle`, `removeCompliance`, `clawback` and `batchClawback` in the standards `ERC20BasicStandard`, `ERC20ControlledStandard`, `ERC20RevocableComplianceStandard`, and `ERC20RevocableStandard`.

Recommendation

We recommend removing or overloading the `shareRegistrarRole` function when inheriting contracts, and reviewing the access policy for the `REGISTER_ROLE`.

Update

Fixed in commit `4606fc644b2dd4ee4be78e9f8d8829a698b304aa`

Client's response

Implemented auditor's recommendation and removed the `shareRegistrarRole` function after review.

C-02	Incorrect call to compliance oracle in `ERC20RevocableComplianceStandard`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `isAddressWhitelisted`	195
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `_areWhitelisted`	208
ICompliance	contract `ICompliance.sol` > function `canTransfer`	21

Description

In the `ERC20RevocableComplianceStandard` contract, the `transfer` and `transferFrom` functions are validated through the Compliance Oracle. However, this validation is performed incorrectly.

```
function transfer(address to, uint value)
external virtual override notPaused notFrozen(msg.sender) notFrozen(to) returns (bool)
{
    require(to != address(0x00), "transfer: Invalid address");
    require(isAddressWhitelisted(to), "transfer: Address not whitelisted");
    return _transfer(msg.sender, to, value);
}

// ...

function transferFrom(
    address from,
    address to,
    uint tokens
) external virtual override notPaused notFrozen(from) notFrozen(msg.sender) notFrozen(to)
returns (bool) {
    require(to != address(0x00), "transfer: Invalid address");
    require(isAddressWhitelisted(to), "transferFrom: Address not whitelisted");
    return _transferFrom(from, to, tokens);
```

```
}

// ...

function isAddressWhitelisted(address addr) public view override returns (bool) {
    if (_complianceAddress == address(0x00)) {
        return true; // No whitelist set, skip the check
    }
    return ICompliance(_complianceAddress).canTransfer(address(0), addr, 0);
}
```

As shown above, the `isAddressWhitelisted` function is called only for the `to` address and does not check the `from` address and `amount`. The compliance oracle is passed `0` as the `amount` parameter instead of the actual transfer amount. The documentation for the `canTransfer` function in the `ICompliance` interface specifies:

```
/**
 * @notice Determines if a transfer is allowed under compliance rules
 * @dev Checks if a transfer from one address to another with a specific amount is compliant
 * @dev Compliance rules can include, but are not limited to:
 * @dev - Checking if both sender and receiver are whitelisted
 * @dev - Ensuring the amount does not exceed certain limits
 * @dev - Verifying transfer does not violate the regulatory requirements set by the contract
 * @param from Address of the sender of the tokens
 * @param to Address of the receiver of the tokens
 * @param amount Amount of tokens to be transferred
 * @return bool Returns true if the transfer is compliant, false otherwise
 */
```

As a result, depending on the implementation of the compliance oracle, the following issues may arise:

- A transfer may be approved by the compliance oracle even if the `from` address is under sanctions, as the `from` address is not validated.
- A transfer may be approved with an amount of `0`, bypassing restrictions on the `to` address receiving the actual transfer amount.

Recommendation

We recommend using compliance strictly according to the documentation and verifying the `from` address and the `amount` through the `canTransfer` function of the `ICompliance` interface.

Update

Fixed in commit `7c1aee292f3b72b0fed116a90cf79ba63e3b7574`

Client's response

Implemented auditor's recommendation to properly verify both the from address and the amount through the `canTransfer` function of the `ICompliance` interface. Here's a detailed explanation of how these changes were made to achieve that:

1. Function `isAddressWhitelisted`
   a. Updated to `isAddressWhitelisted(address from, address to, uint256 amount)` to take in the `from` address, `to` address, and `amount` to be transferred.
   b. Calls `ICompliance(_complianceAddress).canTransfer(from, to, amount)` to check compliance for the specific transfer.
2. Function transfer
   a. Updated to call `isAddressWhitelisted(msg.sender, to, value)`.
3. Function `transferFrom`
   a. Updated to call `isAddressWhitelisted(from, to, tokens)`.
4. Function `mint`
   a. Updated to call `isAddressWhitelisted(address(0), to, value)` for minting. The `address(0)` is used since minting is from the contract itself.
5. Function `clawback`
   a. Updated to call `isAddressWhitelisted(from, to, value)`.
6. Function `_areWhitelisted`
   a. Updated to check both `fromList` and `toList` addresses and the corresponding `amounts` using `canTransfer`.
7. Function `batchTransfer`
   a. Updated to call `_areWhitelisted (from, _toList, _amounts)`.
   b. Added type conversion for `from` since only `msg.sender` is used for `from` and `_areWhitelisted` expects an array.
8. Function `batchMint`
   a. Updated to call `_areWhitelisted (from, _toList, _amounts)`.
   b. Added type conversion for `from` since only `address(0)` is used for `from`, since minting is from the contract itself, and `_areWhitelisted` expects an array.
9. Function batchClawback
   a. Updated to call `_areWhitelisted (_fromList, _toList, _amounts)`.

C-03	Inability To Modify Or Revoke `DEFAULT_ADMIN_ROLE` in `AccessControl`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `grantRole`	119
AccessControl.sol	contract `AccessControl` > function `revokeRole`	222
AccessControl.sol	contract `AccessControl` > function `renounceRole`	249

Description

In the `AccessControl` contract, it is not possible to assign or revoke the `DEFAULT_ADMIN_ROLE` due to the `notDefaultAdminRole(role)` and `notDelegatedAdminRole(role)` modifiers in the `grantRole`, `revokeRole`, and `renounceRole` functions. These modifiers prevent setting or revoking roles for a user with the `DEFAULT_ADMIN_ROLE`.

If an address with the `DEFAULT_ADMIN_ROLE` is compromised, it is impossible to revoke the compromised role or grant additional roles to other addresses to mitigate the issue. This leaves the system without a means to regain control, potentially allowing the attacker to maintain their privileged access indefinitely.

It is important to note that situations involving the compromise of admin keys are fairly common. For example, BXH Exchange lost $139M due to a hack involving the loss of admin keys. Therefore, it is crucial to have the ability to change admin keys to more secure ones.

Additionally, the `Beacon` and `Proxy` contracts utilize the `AccessControl` contract. In the `Beacon` contract, the `DEFAULT_ADMIN_ROLE` is assigned to the deployer's address through the constructor. This design flaw means there is no mechanism to change the `DEFAULT_ADMIN_ROLE` to another address, such as a multisig address, which could provide enhanced security.

Recommendation

We recommend adding additional functions to manage the `DEFAULT_ADMIN_ROLE`, similar to the implementation in the `AccessControlDefaultAdminRules` contract.

Update

Fixed in commit `7a60ec378a136114a1825cb71e4cba29e4879e8d`

Client's response

Implemented auditor's recommendation of allowing the adding and revoking of the DEFAULT_ADMIN_ROLE with restrictions.

3.2 MAJOR

M-01	Incorrect Address Validation Through Whitelist in `ERC20RevocableComplianceStandard`
Severity	**MAJOR**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `transfer`	38
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `transferFrom`	55

Description

In the `ERC20RevocableComplianceStandard` contract, the `transfer` and `transferFrom` functions are validated through the Compliance Oracle. However, this validation is performed incorrectly.

```
function transfer(address to, uint value)
external virtual override notPaused notFrozen(msg.sender) notFrozen(to) returns (bool)
{
    require(to != address(0x00), "transfer: Invalid address");
    require(isAddressWhitelisted(to), "transfer: Address not whitelisted");
    return _transfer(msg.sender, to, value);
}

// ...

function transferFrom(
    address from,
    address to,
    uint tokens
) external virtual override notPaused notFrozen(from) notFrozen(msg.sender) notFrozen(to)
returns (bool) {
```

```
    require(to != address(0x00), "transfer: Invalid address");
    require(isAddressWhitelisted(to), "transferFrom: Address not whitelisted");
    return _transferFrom(from, to, tokens);
}

// ...

function isAddressWhitelisted(address addr) public view override returns (bool) {
    if (_complianceAddress == address(0x00)) {
        return true; // No whitelist set, skip the check
    }
    return ICompliance(_complianceAddress).canTransfer(address(0), addr, 0);
}
```

As shown above, the `isAddressWhitelisted` function is called only for the `to` address and does not check the `from` address.

This allows operations to be performed from a `from` address that is not included in the whitelist of the Compliance Oracle and may be under sanctions.

Recommendation

We recommend adding a whitelist check for both `from` and `to` addresses.

M-02	Inability To Modify `allowance` for Frozen Addresses in `ERC20ControlledStandard`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `approve`	87

Description

In the `approve` function of the `ERC20ControlledStandard` contract, the `allowance` can only be modified for addresses that are not frozen.

This limitation creates the following scenario:

- Alice grants an `allowance` to Bob.
- Bob's address is subsequently frozen.
- Alice attempts to revoke the `allowance` for Bob but is unable to do so because Bob's address is frozen.
- Later, when Bob's address is unfrozen, Bob can withdraw funds from Alice, contrary to Alice's intent to revoke the `allowance`.

Furthermore, it is noteworthy that even an administrator cannot modify the `allowance` for a frozen address.

Recommendation

We recommend reviewing the logic of the `approve` function to allow the `allowance` to be set to zero for frozen addresses.

Update

Fixed in commit `2d8b26fb806af700fdc93a3557eb4585665145fa`

Client's response

Implemented auditor's recommendation by adjusting the `approve` function to allow setting the `allowance` to zero even if the address is frozen.

M-03	Inability to Freeze an Address While the Contract Is Paused in `ERC20ControlledStandard`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard`	176

Description

In the `ERC20ControlledStandard` contract, the same role is responsible for both pausing and freezing addresses. This creates the following potentially scenario:

Consider a scenario where malicious activity is detected in the protocol. The admin pauses the protocol using the `pause` function to investigate. During the investigation, the admin identifies the malicious address. However, the admin cannot freeze this address because the `freeze` function is not operable while the contract is paused.

To freeze the address, the admin must first unpause the contract and then invoke the `freeze` function. During the interval between these two transactions, the malicious address can continue its harmful activity.

Recommendation

We recommend allowing the `freeze` function to be called while the contract is in a paused state.

Update

Fixed in commit `5c0e1a4c29ba08e79c113919d8871c05c519ffcf`

Client's response

Implemented auditor's recommendation by removing the `notPaused` modifier from the `freeze` and `batchFreeze` functions, allowing them to be called by the `REGISTRAR_ROLE` while the contract is in a paused state.

M-04	Non-Optimal Recursive Calls in Role Removal in `AccessControl`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_recursiveHelper`	315

Description

In the `_recursiveHelper` function of the `AccessControl` contract, roles assigned with `DELEGATE_ADMIN_ROLE` are removed recursively. This leads to inefficient execution. For example, the final action within `_recursiveHelper` is a role reset by invoking the `_revokeRole` function. However, after the `_recursiveHelper` function returns, `_revokeRole` is called again for the same address, causing redundant operations.

If the recursion depth is, for instance, `10` (considering each delegated admin can assign their delegate and the maximum number of delegates is limited by `MAX_DELEGATES`), each recursive call will involve iterating through all delegates. This can create a gas bomb effect, which means that the transaction consumes an excessive amount of gas. The higher the value of `MAX_DELEGATES`, the more gas will be used, potentially making the transaction prohibitively expensive or even causing it to fail due to exceeding the block gas limit.

Recommendation

We recommend limiting the recursion depth, for example, by allowing the assignment of delegates only up to a certain level (e.g., a delegated admin cannot assign another delegate). Alternatively, consider revising the use of loops within the `_recursiveHelper` function to make it more efficient.

Update

Initial fix in commits `e82f57b920f2c9c74a0a64f96c3aed10a5c08d5c` and `ece880391fa06a7543f97e73929f3a95cac11e3e`. Final fix in `598cbcda125b9fe22f524751b03a38a8f931b4c9`

Client's response

Implemented auditor's recommendation by revising the `_recursiveHelper` function. The `_recursiveHelper` function has been replaced by `_recursiveRemoveDelegate`. This function now directly removes the delegated role in a single pass and removes the delegate from the mapping in the same call, avoiding redundant operations.

Oxorio's response

The `_recursiveRemoveDelegate` function in the `AccessControl` contract is intended to recursively remove a delegate and their descendants. However, the current implementation contains significant issues:

1) Incomplete descendant removal: The recursion only removes the first-level descendant in the delegate chain due to a flawed condition that compares the parent address (`delegatedAdmins[account].from`) with `msg.sender` at each level:

```
if (delegatedAdmins[account].from == msg.sender || _isDefaultAdmin) {
```

As a result, deeper levels in the delegate chain are not reached unless `_isDefaultAdmin` is set to `true`, which severely limits the effectiveness of the function.

2) Unupdated ancestor `delegates` array: When a delegate `account` is removed from the `DELEGATED_ADMIN_ROLE`, the `delegates` array of the ancestor who originally delegated the role remains unchanged. This leads to the `delegates` array containing addresses of delegates who no longer hold any delegation rights. For example:

- If delegate `Alice` assigns a role to `Carol` and later revokes it, the `delegatedAdmins[Alice].delegates` array will still contain `Carol's` address.
- If another delegate, `Bob`, assigns the same role to `Carol`, `Carol's` address will appear in both `delegatedAdmins[Alice].delegates` and `delegatedAdmins[Bob].delegates`.
- If `Alice` is later revoked, `Carol` might incorrectly lose the role granted by `Bob`.

These issues can lead to incorrect delegation states and potential access control vulnerabilities.

We recommend revisiting the recursive logic in the `_recursiveRemoveDelegate` function. Specifically, consider splitting the delegate removal into a separate internal function and ensuring that the `delegates` arrays are accurately updated. This can be achieved by removing delegates from the ancestor's `delegates` array upon role revocation and correctly propagating the removal through the entire delegate chain:

```
function _recursiveRemoveDelegate(address account, bool _isDefaultAdmin) internal virtual {
    require(account != address(0x00), "0x address");
```

```
    address parent = delegatedAdmins[account].from;
    require(parent == msg.sender || _isDefaultAdmin, "no rights to revoke");

    if (parent != address(0)) {
        Delegate memory parentInfo = delegatedAdmins[parent];
        uint256 delegatesLength = parentInfo.delegates.length;
        for (uint256 i; i < delegatesLength; ++i) {
            if (parentInfo.delegates[i] == account) {
                parentInfo.delegates[i] = parentInfo.delegates[delegatesLength - 1];
                parentInfo.delegates.pop();
                break;
            }
        }
    }

    _removeDelegate(account);
}

function _removeDelegate(address account) internal virtual {
    Delegate memory delegateInfo = delegatedAdmins[account];
    uint256 delegatesLength = delegateInfo.delegates.length;

    for (uint256 i; i < delegatesLength; ++i) {
        _removeDelegate(delegateInfo.delegates[i]);
    }

    delete delegatedAdmins[account];
    _revokeRole(DELEGATED_ADMIN_ROLE, account);
    lastDelegatedAdmin--;
}
```

Client's response

Implemented auditor's recommendation: In `_recursiveRemoveDelegate`, we first identify the parent of the delegate being removed. We then update the parent's delegates array to remove the reference to this delegate, ensuring that once the delegate is removed, their address is no longer stored in the parent's array. The `_removeDelegate` function is then called, which recursively traverses the delegate chain, ensuring that all descendants are correctly removed. The function iterates over each delegate in the delegates array, removing them one by one and ensuring no stale references are left behind. After removing the delegate and all of their descendants, we revoke the `DELEGATED_ADMIN_ROLE` from the account, ensuring that they no longer hold any administrative privileges within the system.

M-05	Lack of `upgradeToAndCall` Function for Secure Implementation Reinitialization in `Beacon.sol`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
Beacon.sol	-	14

Description

In `Beacon.sol`, there is a function `upgradeTo` that allows the implementation to be changed. However, the function `upgradeToAndCall`, which permits reinitializing the implementation in a single transaction, is missing. This limitation requires reinitializing the implementation through a separate transaction, which introduces significant security risks.

The absence of `upgradeToAndCall` function means that reinitialization must occur in a second, separate transaction. This introduces a window of vulnerability where a malicious actor could intercept the transaction, leading to unauthorized modifications or disruptions. Furthermore, during the gap between the upgrade and the reinitialization transactions, there is a risk of race conditions, where the contract's behavior may become unpredictable or exploitable. The necessity for multiple transactions also increases the complexity and the attack surface of the system, making it more susceptible to various forms of attacks.

Recommendation

We recommend adding the function `upgradeToAndCall`, which allows changing the implementation and reinitializing the contract in one transaction.

Update

Fixed in commit `ed71ecabedc2561d4562fbae8d4a812c3b3df93d`

Client's response

Given that the implementation contract doesn't utilise and initalization explicitly we don't understand the need for this. What exactly would be "called" at the new implementation contract as part of this function?

Oxorio's response

During the lifecycle of a contract, it might become necessary to add new storage variables or make other state changes. For instance, to initialize these new values, you may need to introduce a new function, such as `init2()`. This initialization function should be executed in the same transaction as the upgrade to ensure that the contract's state is properly set up without leaving any gaps that could be exploited.

The necessity of such an approach is highlighted by the existing practices in well-established libraries such as OpenZeppelin. They provide a function `upgradeAndCall` that facilitates the upgrade and initialization in a single transaction.

M-06	`REGISTRAR_ROLE` Controls All Function Calls Alongside `DEFAULT_ADMIN_ROLE` in `BaseERC20`
Severity	**MAJOR**
Status	• ACKNOWLEDGED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	122
BaseERC20.sol	contract `BaseERC20` > function `mint`	183
BaseERC20.sol	contract `BaseERC20` > function `burn`	193
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchMint`	35
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchBurn`	54
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `pause`	59
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `unpause`	67
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `mint`	146
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `burn`	164
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `freeze`	174
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `unfreeze`	184
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchMint`	218
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchBurn`	239
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchFreeze`	255
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchUnfreeze`	266
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `mint`	70
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `clawback`	89
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchMint`	128

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchClawback`	149
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `setCompliance`	169
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `removeCompliance`	182
ERC20RevocableStandard.sol	contract `ERC20RevocableStandard` > function `clawback`	30
ERC20RevocableStandard.sol	contract `ERC20RevocableStandard` > function `batchClawback`	44

Description

In the contract `BaseERC20` and all token standard contracts, the `REGISTRAR_ROLE` controls all standard functions, thereby having almost the same rights as the `DEFAULT_ADMIN_ROLE`, except for role assignment. Consequently, a user with these rights can control all important standard functions, and if the address is compromised, it may lead to malicious actions.

This current setup gives the `REGISTRAR_ROLE` too much power, similar to that of the `DEFAULT_ADMIN_ROLE`, which can lead to centralized control and increased risk if the address is compromised. Moreover, it limits the ability to implement independent security measures, such as pausing the contract, without granting excessive permissions to third-party systems. For instance, it is currently impossible to use a monitoring and proactive protection system that can pause the contract in case of negative impact without granting full control over the `REGISTRAR_ROLE` to such a third-party system.

Another example is in the function `setCompliance` of the contract `ERC20RevocableComplianceStandard`, where the `REGISTRAR_ROLE` can change the compliance address. Compliance checks are used in functions that can only be executed by the registrar role, such as `batchMint`. This creates a security risk where the registrar could change the compliance address to bypass whitelist checks, allowing transactions from blocked addresses.

The recommendation to separate roles is based on the principle of risk diversification. If one role is compromised, the risk is minimized compared to having a single role with a large number of capabilities being compromised. Each role should ideally have a limited scope of responsibilities, thereby reducing the impact of any potential breach.

Additionally, these roles can be assigned to addresses created from different HD wallets, further enhancing security by diversifying the keys and reducing the risk associated with any single point of failure.

Recommendation

We recommend assigning separate roles for each operation, such as `PAUSER_ROLE`, `UNPAUSER_ROLE`, `MINTER_ROLE`, `BURNER_ROLE`, `FREEZER_ROLE`, `UNFREEZER_ROLE`, etc.

Update

Client's response

See the documentation

M-07	`changeBeacon` Is Vulnerable To Function Selector Clashing
Severity	**MAJOR**
Status	• FIXED

Description

The `DEFAULT_ADMIN_ROLE` is assigned to the same address for both the implementation and the proxy. So there is a risk of function clashing, where a function in the implementation may have the same signature as a `changeBeacon` function in the Proxy. Suppose the implementation has a function `function_clash_XXX(address)` and is called by the `DEFAULT_ADMIN_ROLE`. If the first 4 bytes of the signature match the `changeBeacon(address)` function, the Proxy will mistakenly call `changeBeacon()` instead of `function_clash_XXX()`, causing the contract to malfunction. Additionally, the Proxy inherits the `AccessControl` contract and all its functions, which increases the likelihood of function clashing. To minimize such risks, the Proxy should have its own `DEFAULT_ADMIN_ROLE` address responsible solely for upgrades and should not have the ability to directly call functions in the implementation. You can read more about this vulnerability and its implications in this OpenZeppelin's Audit Report.

Recommendation

We recommend implementing a fix which restricts the Proxy's functions only to its owner. This approach forwards every message from other users to the implementation contract. Assigning a separate `DEFAULT_ADMIN_ROLE` for the Proxy to manage upgrades, while the implementation should have a different owner, will further ensure that each role operates within its intended scope, preventing potential function clashing.

Update

Fixed in commits `e82a20a035a574ff1419fa647870e4102fe0500e`, `cbf2bca0f3ef78d5869fa5c78fe564fe322933be`, `f76c485879d7be0ee30880631d11c71a96678343`.

Client's response

Implemented fix by moving this function to the implementation.

3.3 WARNING

W-01	`_initializationOwnerAddress` Not Used in Function in `BaseERC20`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	92

Description

In the `initializeWithRoles` function of the `BaseERC20` contract, the parameter `initOwner` is assigned to the variable `_initializationOwnerAddress` but is then not used.

This can lead to confusion if, in a future implementation, this parameter is intended to be used. The existence of a value for this parameter may not be obvious as it is an internal variable.

Recommendation

We recommend removing this parameter from the code and the `initializeWithRoles` function.

Update

Fixed in commits `1b0799cce626d2e945653fb5df25d0f16886ee64`, `055dd4baee0ea595d96db33b2c1bfdd6b186df10`, `22d2a16c9604c7ef66e29b03f4ff34b9e42cd243`, `9c26c49aba3603fb07d86a8bef33b182ffa66fae`, `8d922fadcc877978116671d0ae4c5e865cf96343`, `38b99fd31372ed5fe273e5dba3ac34305d7b55a2`

Client's response

This has highlighted an incorrect usage of the `_initializationOwnerAddress` that should be set at the proxy and the `initializeWithRoles` should be protected by this role.

W-02	Delegate Admin Cannot Directly Remove Their Delegate's Delegate in `AccessControl`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_recursiveRemoveDelegate`	308

Description

In the `_recursiveRemoveDelegate` function of the `AccessControl` contract, when removing the `DELEGATE_ADMIN_ROLE`, the remover must either be the default admin or the direct delegator of the role being removed.

Consider the following hierarchy: an admin (`delegateAdmin1`) has a delegate (`delegateAdmin2`), who in turn has their own delegate (`delegateAdmin3`):

```
delegateAdmin1 -> delegateAdmin2 -> delegateAdmin3
```

This means that `delegateAdmin1` cannot directly remove `delegateAdmin3`. If `delegateAdmin3` is compromised, `delegateAdmin1` must remove `delegateAdmin2`, which will then recursively remove `delegateAdmin3`.

Recommendation

We recommend considering adding the ability for an admin to remove a specific delegate from the "tree" of delegates without affecting others.

Update

Fixed in commit `ece880391fa06a7543f97e73929f3a95cac11e3e`

Client's response

Implemented auditor's recommendation, the `_recursiveRemoveDelegate` function now checks if the caller is either the direct delegator or a `DEFAULT_ADMIN_ROLE`, allowing a higher-level admin to remove any delegate in the tree directly.

W-03	No Clear Error Message for Insufficient Funds in `BaseERC20`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `_transfer`	284
BaseERC20.sol	contract `BaseERC20` > function `_burn`	312

Description

In the `_transfer` and `_burn` functions of the `BaseERC20` contract, it is possible to pass a `value` greater than `_balances[from]`.

This results in an underflow without returning a readable error message to indicate the issue. When such an underflow occurs, the transaction fails silently, making it difficult for users and developers to diagnose and understand the failure. This lack of clear error messaging can lead to confusion, misinterpretation of the issue, and additional time spent on debugging.

Recommendation

We recommend checking the `value` against the balance and returning an error message that clearly explains the cause of the problem. This will improve the transparency and usability of the contract, making it easier for users and developers to understand and resolve issues related to insufficient funds.

Update

Fixed in commit `57d8f2a503b30f89aee9906883cf4a9c05c4034b`

Client's response

Implemented auditor's recommendation by adding checks for the `value` against the caller's `_balance` and returning an error message.

W-04	Inefficient Loop Over `delegatedAdmins` in `AccessControl`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_recursiveHelper`	331
AccessControl.sol	contract `AccessControl` > function `_grantDelegateAdminRole`	357
AccessControl.sol	contract `AccessControl` > function `_isTheDelegator`	373

Description

In the mentioned locations, a loop is used over the `delegatedAdmins` array to find an element with the required conditions. This approach is inefficient because it involves iterating over the entire array, which can be particularly costly in terms of gas consumption if the array is large. Moreover, some functions call this loop multiple times, compounding the inefficiency and leading to significantly higher gas costs.

The problem is exacerbated when these functions are called frequently or when the array size grows, as the gas cost increases linearly with the number of elements in the array. This can result in unexpected and prohibitively high gas expenses for users interacting with the contract.

Recommendation

We recommend reviewing this structure and using mappings of structures, such as `mapping(address => Delegate)`, to store the `delegatedAdmins`. Additionally maintain a cursor for the free slot, which can also be limited by `MAX_DELEGATES` to prevent excessive growth of the data structure.

Update

Fixed in commit `ece880391fa06a7543f97e73929f3a95cac11e3e`

Client's response

Implemented auditors recommendation, the `Delegate` struct now includes an array of delegates. The loop over the array has been optimized, and the structure ensures that unnecessary iterations are minimized.

W-05	Limited Functionality of `DELEGATED_ADMIN_ROLE` in `AccessControl`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl`	239

Description

In the contract `AccessControl`, the `DELEGATED_ADMIN_ROLE` has a complex hierarchical structure but is limited to only assigning roles. It lacks the ability to manage other critical contract functions, unlike the `DEFAULT_ADMIN_ROLE`. This limitation can lead to inefficiencies and potential security issues, as the `DELEGATED_ADMIN_ROLE` cannot perform comprehensive administrative tasks, requiring the `DEFAULT_ADMIN_ROLE` to handle both role assignment and contract management. This centralizes power and can become a single point of failure if the `DEFAULT_ADMIN_ROLE` is compromised.

Recommendation

We recommend reviewing the functions of `DELEGATED_ADMIN_ROLE` to expand its capabilities or eliminating the hierarchical structure of `DELEGATED_ADMIN_ROLE` and managing role assignments directly through the `grantRole` function.

Update

Client's response

See the documentation

W-06	Inefficient Loop for Setting Multiple Delegates in `AccessControl`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_grantDelegateAdminRole`	357
AccessControl.sol	contract `AccessControl` > function `batchGrantDelegateAdminRole`	188

Description

In the function `batchGrantDelegateAdminRole` of the contract `AccessControl`, the `DELEGATED_ADMIN_ROLE` is set in a loop that calls `_grantDelegateAdminRole`. This involves iterating through all `delegatedAdmins` from the start to find a free slot, which is inefficient and costly in terms of gas. Each iteration over the `delegatedAdmins` array to find an available slot increases gas consumption, potentially leading to unexpected expenses, especially as the array grows and depending on the value of the constant `MAX_DELEGATES`. At high values of `MAX_DELEGATES`, this could potentially lead to transaction reverts due to the excessive gas required to process these loop.

Recommendation

We recommend reviewing this structure and considering the introduction of a separate parameter `lastDelegatesIndex`, which stores the index of the last free slot for the delegate's index. This would optimize the slot-finding process and reduce gas costs.

Update

Fixed in commit `ece880391fa06a7543f97e73929f3a95cac11e3e`

Client's response

The `batchGrantDelegateAdminRole` function now checks the total number of delegates against `MAX_DELEGATES` before starting the loop, ensuring that it won't exceed the limit. The function has been refactored to use `grantDelegateAdminRole`, reducing redundant code and iterations.

W-07	Direct Initialization Call in `Proxy` Constructor
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Proxy.sol	contract `Proxy` > `constructor`	41

Description

In the `constructor` of the `Proxy` contract, there is a direct call to the initialization function of the implementation contract via `initializeWithRoles`. This practice is considered suboptimal because it makes the `Proxy` contract dependent on a specific implementation.

```
(bool success, ) = implementationAddress.delegatecall(
    abi.encodeWithSignature(

"initializeWithRoles(address,string,string,uint8,uint256,address,address,address,address)",
        initOwner,
        name_,
        symbol_,
        decimals_,
        totalSupply_,
        tokensRecipient_,
        owner,
        issuer,
        registrar
    )
);
```

Such an approach ties the `Proxy` contract to a specific implementation and its initialization logic, which can lead to maintenance issues and reduced flexibility. Currently, the Proxy contract is not universal, as it requires this specific initialization function to be present in the implementation. If an implementation contract without the `initializeWithRoles` function is deployed, the deployment will fail.

To avoid such constraints and ensure seamless integration with various implementation contracts, it is better to pass the function call as a parameter. This approach will enhance

the flexibility and universality of the Proxy contract, allowing it to work with different implementations without requiring specific initialization functions.

Recommendation

We recommend replacing the direct call with a call through a `data` variable, similar to the implementation in OpenZeppelin's ERC1967Utils. This would decouple the proxy from a specific implementation and improve the flexibility and maintainability of the contract.

Update

Fixed in commit `3a52615fdd5c8acb94de873acf529aa0ec181493`

Client's response

Implemented auditors recommendation

W-08	Unjustified Increase in Contract Bytecode
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Beacon.sol	-	-
Proxy.sol	-	-

Description

The `Proxy` and `Beacon` contracts inherit the `AccessControl` contract, which includes logic used exclusively in token standards. However, the `Proxy` and `Beacon` contracts only utilize `DEFAULT_ADMIN_ROLE` from `AccessControl`, leading to an unnecessary increase in contract size and higher deployment costs.

Recommendation

We recommend creating a separate contract that includes only the admin functionality, reducing the bytecode size and deployment cost of the `Proxy` and `Beacon` contracts.

W-09	Require in cycle in `ERC20BasicStandard`, `ERC20ControlledStandard`, `ERC20RevocableComplianceStandard`, `ERC20RevocableStandard`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchTransfer`	23
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchMint`	42
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchBurn`	62
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchTransfer`	205
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchMint`	224
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchBurn`	246
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchFreeze`	257
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchUnfreeze`	268
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchTransfer`	113
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchMint`	135
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchClawback`	155
ERC20RevocableStandard.sol	contract `ERC20RevocableStandard` > function `batchClawback`	49

Description

In the mentioned locations, `require` statements are used within loops. If any parameter within the loop fails to meet the condition, the transaction will revert, making it impossible to determine which specific parameter was invalid.

Recommendation

We recommend removing the `require` statements and replacing them with `if` statements to ensure that the operation is only applied to addresses in the batch that meet the conditions, and emitting an event for invalid addresses indicating which parameters were invalid, or providing a clearer error message in the `require` statements.

W-10	Missing Check for Non-Zero `amount` in `ERC20BasicStandard`, `ERC20ControlledStandard`, `ERC20RevocableComplianceStandard`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
ERC20BasicStandard.sol	contract `ERC20BasicStandard` > function `batchTransfer`	24
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchTransfer`	205
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchTransfer`	115

Description

In the mentioned locations, there is no check to ensure that `amounts[i]` is non-zero. Allowing zero-value transfers can result in spam transactions that unnecessarily increase the transaction volume on the network without transferring any real value. This can degrade the network's performance. Additionally, these zero-value transactions generate additional log entries, creating noise in logging systems and making it difficult to identify and analyze significant events. Processing these transactions also consumes computational resources and storage space, which could be better utilized for meaningful transactions.

Recommendation

We recommend adding a check to ensure that `amounts[i]` is non-zero in the batch transfer functions.

Update

Fixed in commit `65975a50c5c57205326728da72e34448ad679c8e`

Client's response

Implemented auditor's recommendation.

3.4 INFO

I-01	`ISSUER_ROLE` is Unused in `BaseERC20`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	119

Description

In the function `initializeWithRoles` of the contract `BaseERC20`, an address and role `ISSUER_ROLE` are set, but this role is not used in the contract except in the function `shareRegistrarRole`. This redundancy can lead to confusion and potential security risks, as it is unclear what permissions and actions are intended for this role. Moreover, maintaining unused roles adds unnecessary complexity to the contract.

Recommendation

We recommend reviewing the potential uses of this role or removing it if it is unnecessary.

Update

Client's response

See the documentation

I-02	Missing Interfaces in `supportsInterface` in `BaseERC20`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `supportsInterface`	331

Description

In the `supportsInterface` function of the `BaseERC20` contract, the interfaces `IERC20Burnable`, `IERC20Mintable` and `IERC20Events` are not included, even though the contract utilizes these interfaces. This omission can complicate the integration of the contract with external systems, particularly if they query for interface support using EIP-165.

Recommendation

We recommend adding these interfaces to the `supportsInterface` function to ensure that the contract correctly reports its support for `IERC20Burnable`, `IERC20Mintable`, and `IERC20Events`.

Update

Fixed in commit `8e7ef632f3a0e707445825d5e907deb7171296e9`

Client's response

Implemented auditors recommendation by adding missing interfaces to `supportsInterface` and additionally added this to all token standards wth new interfaces.

I-03	Inconsistent Use of `msg.sender` and `_msgSender()`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `shareRegistrarRole`	146
AccessControl.sol	contract `AccessControl` > function `grantDelegateAdminRole`	166
AccessControl.sol	contract `AccessControl` > function `batchGrantDelegateAdminRole`	184
AccessControl.sol	contract `AccessControl` > function `revokeDelegateAdminRole`	204
AccessControl.sol	contract `AccessControl` > function `revokeDelegateAdminRole`	208
AccessControl.sol	contract `AccessControl` > function `renounceRole`	258
AccessControl.sol	contract `AccessControl` > function `_grantRole`	302
AccessControl.sol	contract `AccessControl` > function `_grantDelegateAdminRole`	359
AccessControl.sol	contract `AccessControl` > function `_isTheDelegator`	375
Beacon.sol	contract `Beacon` > modifier `onlyAdmin`	23
Beacon.sol	contract `Beacon` > `constructor`	29
BaseERC20.sol	contract `BaseERC20` > function `approve`	143
BaseERC20.sol	contract `BaseERC20` > function `approve`	144
BaseERC20.sol	contract `BaseERC20` > function `approve`	145
BaseERC20.sol	contract `BaseERC20` > function `transfer`	158
BaseERC20.sol	contract `BaseERC20` > function `_transferFrom`	265
BaseERC20.sol	contract `BaseERC20` > function `_transferFrom`	266
BaseERC20.sol	contract `BaseERC20` > function `_transferFrom`	268
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `approve`	86

File	Location	Line
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `approve`	92
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `approve`	94
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `approve`	96
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `transfer`	112
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `transfer`	117
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `transferFrom`	131
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchTransfer`	198
ERC20ControlledStandard.sol	contract `ERC20ControlledStandard` > function `batchTransfer`	206
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `transfer`	31
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `transfer`	39
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `transferFrom`	53
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchTransfer`	105
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchTransfer`	116

Description

In the mentioned locations, both `_msgSender()` and `msg.sender` are used interchangeably. While `_msgSender()` returns `msg.sender`, using them inconsistently can lead to confusion and reduce the readability of the code.

Recommendation

We recommend standardizing the use of `msg.sender` across the codebase for consistency and improved code readability.

Update

Fixed in commit `927bdc8f43b1a8fa17d77382049bc887bcdbfa32`

Client's response

Implemented auditor's recommendation and aligned `msg.sender` as the standard across the codebase.

I-04	Redundant `Beacon` Contract in Implementation Upgrade Process in `Proxy`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Proxy.sol	contract `Proxy` > function `_implementation`	63

Description

In the `_implementation` function of the `Proxy` contract, the implementation for `Proxy` is retrieved from another contract, `Beacon`. In other words, the `Beacon` contract acts as an intermediary between the `Proxy` and its implementation.

This design means that the admin of the `Proxy` cannot directly change its implementation; they can only change the address of the `Beacon` contract. The `Beacon` contract itself has no other purpose than to store and update the implementation address.

This approach introduces several issues and inconveniences:

1. Additional Complexity: The presence of an intermediary contract adds unnecessary complexity to the upgrade process.
2. Indirect Control: Admins lack direct control over the implementation, which could complicate emergency updates or quick fixes.
3. Potential Delays: The extra step of updating the `Beacon` contract may introduce delays in deployment and management.

Recommendation

We recommend considering the removal of the intermediary `Beacon` contract in the implementation setup process. This would allow the admin to directly manage the implementation of the `Proxy`, simplifying the upgrade process and providing more direct control over contract updates.

Update

Client's response

this serves a business logic purpose to ensure multiple tokens (Proxy's) are always referencing the same implementation contract.

i.e we have a given token standard implementation that is used by 10 tokens. We want to ensure that they are all always utilizing the same version. When we perform an upgrade, the new implementation contact is upgraded at the beacon so all token's are referencing the implementation contact by way of retrieving this from the beacon. This means that we dont need to upgrade each token to the now implementation individually and avoids a risk that some are operating on outdated logic for a period of time ensuring constancy across tokens

The ability to upgrade at the proxy level exists (by changing the beacon) so that we can also change the token standard a given token follows. This allows us to maintain flexibility to respond to changing client segments and jurisdictional specific regulatory changes.

I-05	Insufficient Validation of Parameters in `BaseERC20` and `Beacon`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Beacon.sol	contract `Beacon` > `constructor`	25
Beacon.sol	contract `Beacon` > function `upgradeTo`	40
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	78

Description

In the mentioned locations, input parameters are insufficiently validated. For example, implementation addresses can be passed as zero.

Some addresses in the parameters are not checked to ensure they are not zero.

The implementation address is not validated to confirm it is a contract, nor is it checked whether the implementation includes the mandatory `initializeWithRoles` function.

This lack of validation can lead to several issues:

1. Zero Address Vulnerability: If a zero address is used, it can result in the contract pointing to an invalid implementation, causing the contract to malfunction or become unusable.
2. Non-Contract Implementation: If the implementation address is not a contract, the system can encounter unexpected behavior or runtime errors, leading to potential security breaches.
3. Missing Required Functions: If the implementation does not include the required `initializeWithRoles` function, the contract will fail to initialize properly, which could result in uninitialized variables and unintended contract behavior.

Recommendation

We recommend thoroughly checking the input parameters to ensure that the provided values are valid. Specifically:

- Verify that addresses are not zero.
- Ensure that the implementation address is indeed a contract.

- Check that the implementation contains the required `initializeWithRoles` function.

Update

Fixed in commits `664497e15e956a16a98a6b47904e45bc63ca947c` and `51eea9ec595bdd84e0631bee6ecff2383d238d77`

Client's response

Implemented auditor's recommendation by verifying `owner`, `issuer`, and `registrar` is not zero, since these addresses are critical for role assignment. The implementation address is also now verified as being contract using a low-level check. Finally, since `initializeWithRoles` is a state-changing function, a check was implemented in the Beacon by leveraging `ERC165` to validate that the implementation contract supports an interface with the `initializeWithRoles` function defined.

I-06	Redundant Constant in OpenZeppelin Library in `Strings`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Strings.sol	contract `Strings`	12

Description

In the `Strings` contract, two identical constants with different names are defined:

```
bytes16 private constant HEX_DIGITS = "0123456789abcdef";
bytes16 private constant _SYMBOLS = "0123456789abcdef";
```

In the original source from OpenZeppelin, only one constant is used.

Recommendation

We recommend aligning the library code with the original source to maintain codebase cleanliness and minimize discrepancies with the original implementation.

Update

Fixed in commit `d61efe59e5461f3f2c78f7036af8dc45cfd26586`

Client's response

Implemented auditors recommendation by aligning `Strings` library with the original OpenZeppelin source as we do not use HEX_DIGITS within the scope of this project.

I-07	Variables Can Be Made `immutable` in `BaseERC20`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	101

Description

In the `initializeWithRoles` function of the `BaseERC20` contract, several variables are set that are not modified thereafter. These variables can be made `immutable`:

- `_name`
- `_symbol`
- `_decimals`
- `_initializationOwnerAddress`
- `_isInitialized`

Recommendation

We recommend making these variables `immutable` to improve code clarity and potentially optimize gas usage.

Update

Client's response

We acknowledge the auditors recommendation.

I-08	Misleading Error Message in `BaseERC20`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	96

Description

In the `initializeWithRoles` function of the `BaseERC20` contract, there is a check to ensure that the length of `symbol_` is between 1 and 12 inclusive. However, the error message is misleading:

```
require(
    (bytes(symbol_).length > 0) && (bytes(symbol_).length < 13),
    "Symbol length should always be between 1 & 13"
);
```

Recommendation

We recommend updating the error message to accurately reflect the condition to avoid misleading users:

```
require(
    (bytes(symbol_).length > 0) && (bytes(symbol_).length < 13),
    "Symbol length should always be between 1 & 12"
);
```

Update

Fixed in commit `ff403e7c8ead2e83355f1ce762ae348043e87a9c`

Client's response

Implemented auditors recommended fix.

I-09	Inefficient Variable Usage in `AccessControl`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_grantDelegateAdminRole`	342

Description

In the function `_grantDelegateAdminRole` of contract `AccessControl`, there is a loop that searches for a null address in the array. Upon finding a null address, the loop exits using `break`.

This approach necessitates the use of two unnecessary variables `_done` and `_ret`, leading to more complex code and higher gas costs.

Recommendation

We recommend considering returning the result from the function using `return` as soon as a null address is found. This way, you can avoid creating unnecessary variables and maintain the clarity and simplicity of the codebase.

Update

Fixed in commit `ece880391fa06a7543f97e73929f3a95cac11e3e`

Client's response

Implemented auditor's recommendation by directly adding a delegate to the mapping when a free slot is found, removing the need for these variables. The function now directly returns when a delegate is added.

I-10	Use `++i` to save gas
Severity	**INFO**
Status	• FIXED

Description

In all contracts across the codebase `i++` is used in loops. However `++i` costs less gas compared to `i++` or `i += 1` for unsigned integer, as pre-increment is cheaper (about 5 gas per iteration). This statement is true even with the optimizer enabled.

Recommendation

We recommend using pre-increment `++i` instead of post-increment `i++`.

Update

Fixed in commit `565014615bd05b8b3850f25d8d09a021d4f8c9ff`

Client's response

Implemented auditor's recommendation across all loops throughout the codebase.

I-11	Non-Optimal Array Length Determination in `RoleAgencyLib`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
RoleAgencyLib.sol	contract `RoleAgencyLib` > function `removeRoleContext`	13

Description

In the function `removeRoleContext` of contract `RoleAgencyLib`, multiple calls to `.length` occur during the function execution and within the `for` loop. This results in excessive gas consumption when working with `storage`.

Recommendation

We recommend determining the array size `roleContexts.length` once and assigning it to a memory variable to use within the loop, reducing gas consumption.

Update

Fixed in commit `0eaec1ced3f9b748f1afb82c25aaa5fea506a22b`

Client's response

This is fixed by the removal of the `RoleAgencyLib.sol` contract in the fix for I-13.

I-12 Potential Reuse of `_mint` Function Call in `BaseERC20`

Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	108-111

Description

In the function `initializeWithRoles` of contract `BaseERC20`, the following code could potentially be reused by calling the `_mint` function:

```
_totalSupply = totalSupply_;
_balances[tokensRecipient_] = totalSupply_;

emit Transfer(address(0x00), tokensRecipient_, totalSupply_);
```

Recommendation

We recommend replacing the above code with a call to the `_mint` function to maintain codebase cleanliness, reduce contract bytecode size, and avoid code duplication.

Update

Fixed in commit `afd95e01db1ae4e725306136df6b2de45338584a`

Client's response

Implemented auditor's recommendation by replacing the above code with the `_mint` function.

I-13	Unused Files in the Project
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
RoleAgencyLib.sol	contract `RoleAgencyLib`	4
IERC20WhitelistContextControlled.sol	interface `IERC20WhitelistContextControlled`	7
IERC20WhitelistContextRevocable.sol	interface `IERC20WhitelistContextRevocable`	7
IContext.sol	interface `IContext`	13
IContextFactory.sol	interface `IContextFactory`	14
IController.sol	interface `IController`	11
IRoleAgency.sol	interface `IRoleAgency`	11

Description

The specified locations contain libraries and interfaces that are not used in the project.

Recommendation

We recommend considering the removal of unused files to maintain a clean codebase. Removing unnecessary files helps reduce clutter, improve maintainability, and minimize potential confusion for developers.

Update

Fixed in commit `0eaec1ced3f9b748f1afb82c25aaa5fea506a22b`

Client's response

Implemented auditors recommendation and removed unused files that are currently not implemented in the current scope of the project.

I-14	Unused Libraries Included in `BaseERC20`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20`	26
BaseERC20.sol	contract `BaseERC20`	27
BaseERC20.sol	contract `BaseERC20`	28
BaseERC20.sol	contract `BaseERC20`	29

Description

In the mentioned locations, libraries are included in the contract but are not used.

Recommendation

We recommend considering the removal of these included libraries from the contract to maintain a clean codebase. Removing unused libraries helps reduce clutter, improve maintainability, and minimize potential confusion for developers.

Update

Fixed in commits `0762b02b5d633a7a7d14341d2f79d02db15f0268` and `8f74467a1921878881b8052af3cdf2a30680c849`

Client's response

Implemented auditors recommendation by removing the unused libraries from the specified contract.

I-15 Potential Inheritance of Existing Interfaces in `IERC20WhitelistContextControlled.sol`, `IERC20WhitelistContextRevocable.sol`

Severity **INFO**

Status • FIXED

Location

File	Location	Line
IERC20WhitelistContextControlled.sol	interface `IERC20WhitelistContextControlled`	7
IERC20WhitelistContextRevocable.sol	interface `IERC20WhitelistContextRevocable`	7

Description

In the mentioned locations, functions are described that are already defined in separate interfaces in the directory `tokens/interfaces/erc20`, such as `IERC20RevocableCompliance`, `IERC20BatchClawback`, `IERC20Clawback`.

Recommendation

We recommend inheriting `IERC20WhitelistContextControlled` and `IERC20WhitelistContextRevocable` from interfaces describing the same functions for optimal code reuse, or considering the possibility of completely removing these interfaces if they are not used in the project.

Update

Fixed in commit `a4b85a030c13db6c40f1426dcf6fb98e65cf243d`

Client's response

Implemented auditors recommendation by removing mentioned interfaces as they are currently not used within the current scope of the project.

I-16	Insufficient Code Reuse in Inheritance for `ERC20RevocableComplianceStandard`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard`	11

Description

In the contract `ERC20RevocableComplianceStandard`, inherited functions from the `ERC20ControlledStandard` and `ERC20RevocableStandard` contracts are being overridden. The only addition by the `ERC20RevocableComplianceStandard` contract is a whitelist check, while the rest of the code is copied without reusing the inherited functions.

For example, let's compare the `batchMint` function in the two contracts `ERC20RevocableComplianceStandard` and `ERC20ControlledStandard`. They differ by only one line for the whitelist check:

```
// ERC20RevocableComplianceStandard contract

function batchMint(address[] calldata _toList, uint256[] calldata _amounts)
    external
    virtual
    override
    notPaused
    onlyRole(REGISTRAR_ROLE)
{
    require(_areWhitelisted(_toList), "batchMint: One or more addresses not whitelisted");
    uint len_ = _toList.length;
    require(len_ == _amounts.length, "batchMint: Inconsistent input array lengths");

    for (uint256 i = 0; i < len_; i++) {
        require(_toList[i] != address(0x00), "batchMint: Invalid address");
        require(_amounts[i] > 0x00, "batchMint: Invalid value");
        require(!_isFrozen(_toList[i]), "batchMint: Frozen");
        _mint(_toList[i], _amounts[i]);
    }
```

```
}


// ERC20ControlledStandard contract

function batchMint(address[] calldata _toList, uint256[] calldata _amounts)
    external
    virtual
    override
    notPaused
    onlyRole(REGISTRAR_ROLE)
{
    uint len_ = _toList.length;
    require(len_ == _amounts.length, "batchMint: Inconsistent input array lengths");

    for (uint256 i = 0; i < len_; i++) {
        require(_toList[i] != address(0x00), "batchMint: Invalid address");
        require(_amounts[i] > 0x00, "batchMint: Invalid value");
        require(!_isFrozen(_toList[i]), "batchMint: Frozen");
        _mint(_toList[i], _amounts[i]);
    }
}
```

Recommendation

We recommend considering the possibility of reusing identical code when inheriting to maintain code cleanliness, optimize codebase maintenance, and reduce the contract size.

I-17	Simultaneous Use of `uint` and `uint256` Types
Severity	**INFO**
Status	• FIXED

Description

In many contracts across the project, both `uint` and `uint256` are used simultaneously for defining variable types. For example:

```
uint len_ = _toList.length;
require(len_ == _amounts.length, "batchMint: Inconsistent input array lengths");

for (uint256 i = 0; i < len_; i++) {
```

Recommendation

We recommend using a single type for variable definitions to maintain a consistent style across the project. This practice improves code readability and maintainability by reducing confusion and potential type conversion issues.

Update

Fixed in commit `035b91f1764c39dcefc2ef191aa266b3b8912a85`

Client's response

Implemented auditor's recommendation and aligned the codebase to use `uint256`.

4. APPENDIX

OXORIO

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, Oxorio conducted key security audits for notable DeFi projects, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING