WISDOMTREE DIGITAL ERC20 REVOCABLE COMPLIANCE STANDARD SECURITY AUDIT REPORT: ANNEX

AUGUST 30, 2024

CONTENTS

1 AUDIT OVERVIEW

1.1 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.

1.2 PROJECT BRIEF

Title	Description
Client	WisdomTree Digital
Project name	Token Standards v3: ERC20 Revocable Compliance Standard
Category	Token Framework
Repository	https://bitbucket.org/wisdomtreeam/tokenstandardsv3
Documentation	https://bitbucket.org/wisdomtreeam/tokenstandardsv3/src/039218510ae2e46c815c7b31338ac1628f80c3b1/docs/
Initial Commit	`5ed116a1a5d6b071b9b7f85038389c81fe00eeb6`
Final Commit	`0053fed9d3c55e24fa16875211a077dd03b4d92a`
Platform	L1
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Nataly Demidova - nataly@oxor.io

1.3 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.
August 23, 2024	Preliminary report for Round 2 re-audit was submitted.
August 30, 2024	Final report for Round 2 re-audit, incorporating client’s verified fixes, was submitted.

1.4 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	407	45	186	**176**	11
2	src/common/access-control/IAccessControl.sol	161	14	131	**16**	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/StorageSlot.sol	150	17	64	**69**	13
8	src/common/libraries/Strings.sol	70	7	19	**44**	20
9	src/proxies/Beacon.sol	54	11	5	**38**	5
10	src/proxies/Proxy.sol	64	12	5	**47**	11
11	src/tokens/common/BaseERC20.sol	418	63	152	**203**	18
12	src/tokens/interfaces/erc20/IERC20.sol	85	9	60	**16**	0
13	src/tokens/interfaces/erc20/IERC20BatchBasic.sol	44	3	35	**6**	0
14	src/tokens/interfaces/erc20/IERC20BatchClawback.sol	26	1	17	**8**	0
15	src/tokens/interfaces/erc20/IERC20BatchFreeze.sol	27	2	20	**5**	0
16	src/tokens/interfaces/erc20/IERC20Burnable.sol	15	1	10	**4**	0
17	src/tokens/interfaces/erc20/IERC20Clawback.sol	20	1	11	**8**	0
18	src/tokens/interfaces/erc20/IERC20ClawbackEvents.sol	12	1	7	**4**	0
19	src/tokens/interfaces/erc20/IERC20Events.sol	18	2	11	**5**	0
20	src/tokens/interfaces/erc20/IERC20Freeze.sol	25	3	16	**6**	0
21	src/tokens/interfaces/erc20/IERC20FreezeEvents.sol	17	2	10	**5**	0
22	src/tokens/interfaces/erc20/IERC20Mintable.sol	19	2	12	**5**	0
23	src/tokens/interfaces/erc20/IERC20Pausable.sol	22	3	13	**6**	0
24	src/tokens/interfaces/erc20/IERC20PausableEvents.sol	17	2	10	**5**	0
25	src/tokens/interfaces/erc20/IERC20RevocableCompliance.sol	35	5	21	**9**	0
26	src/tokens/interfaces/erc20/IERC20Token.sol	41	3	4	**34**	0
27	src/tokens/interfaces/erc20/IERC20WithRoles.sol	32	2	17	**13**	0
28	src/tokens/interfaces/IBeacon.sol	6	1	1	**4**	0
29	src/tokens/interfaces/ICompliance.sol	26	1	17	**8**	0
30	src/tokens/interfaces/IERC165.sol	14	1	9	**4**	0
31	src/tokens/standards/ERC20BasicStandard.sol	83	9	21	**53**	23
32	src/tokens/standards/ERC20ControlledStandard.sol	338	37	108	**193**	15
33	src/tokens/standards/ERC20RevocableComplianceStandard.sol	252	24	70	**158**	18
34	src/tokens/standards/ERC20RevocableStandard.sol	75	8	22	**45**	27
	Total	**3051**	**338**	**1234**	**1479**	**16**

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.

1.5 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.

1.6 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.

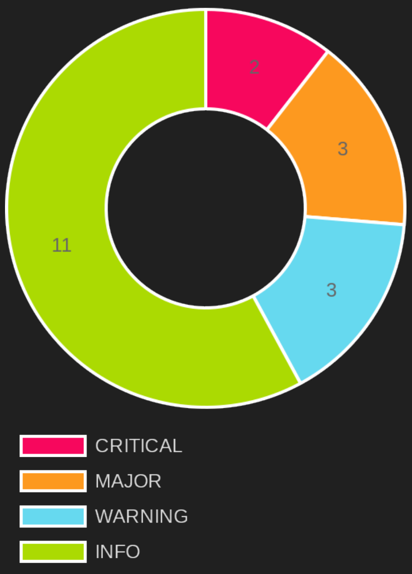
Category	Assessment	Result
Access Control	The project employs a solid role-based access control system. The identified issues related to role management, such as `C-04`, `C-05`, `M-08` and `M-09`, have been successfully resolved.	**Good**
Arithmetic	The project implements standard token arithmetic and 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	The identified complexity issues, particularly in `M-10`, have been successfully addressed, resulting in a more streamlined and optimal codebase. The resolution of these issues has improved the management of code complexity, reducing potential maintenance challenges, minimizing gas usage, and eliminating noticeable code duplication.	**Fair**
Data Validation	The identified issues with data validation have been successfully resolved. This improvement has solidified the validation logic, ensuring better data integrity and reliability across the project.	**Good**
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 previously identified gaps and inaccuracies in the documentation regarding rights and roles have been addressed, with improvements made following M-09, enhancing clarity and comprehensiveness. The Solidity smart contract documentation is now well-maintained, providing clear, up-to-date insights into the code's functionality. NatSpec comments are adequately used, and inline comments effectively clarify complex logic. Comprehensive diagrams illustrate the system architecture and execution flows, and user roles and privileges are thoroughly documented, contributing to the overall robustness of the documentation.	**Excellent**
External Dependencies	The identified flaws in the implementation of the external Compliance Oracle has been successfully resolved. The correction of this dependency has enhanced the reliability and proper functioning of critical operations within the project.	**Good**
Error Handling	The issues related to error handling have been successfully resolved. The project's use of `require` statements now demonstrates solid reliability, providing clearer and more descriptive error messages across most scenarios.	**Good**
Logging and Monitoring	The event logging-related issues has been successfully resolved, strengthening the project's logging and monitoring capabilities. The project now includes solid event logging mechanisms that effectively track system operations, with all state-changing functions emitting events and custom errors signaling specific reasons for reverts.	**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. Several critical components, including proxy handling, are insufficiently tested. Enhancing the full test coverage and incorporating comprehensive test scenarios is recommended.	**Fair**

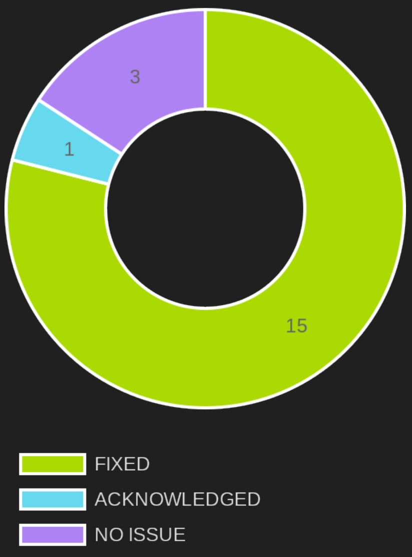
1.7 SUMMARY OF FINDINGS

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. All identified issues have been addressed, with client fixing them or formally acknowledging their status. Detailed descriptions of each finding can be found in the Findings Report section.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**2**	0	2	0	0
MAJOR	**3**	0	3	0	0
WARNING	**3**	0	2	0	1
INFO	**11**	0	8	1	2
TOTAL	**19**	**0**	**15**	**1**	**3**



Issue distribution by severity



Issue distribution by status

1.8 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/common/access-control/AccessControl.sol	**9**	2	3	1	3
src/tokens/common/BaseERC20.sol	**6**	0	0	1	5
src/tokens/standards/ERC20RevocableComplianceStandard.sol	**5**	0	0	1	4
src/proxies/Beacon.sol	**2**	0	0	0	2
src/proxies/Proxy.sol	**1**	0	0	0	1

1.9 CONCLUSION

A comprehensive audit was conducted on 34 smart contracts, initially revealing 2 critical and 3 major issues, along with numerous warnings and informational notes. The audit identified vulnerabilities in role management, inconsistencies in administrative permission enforcement, and opportunities for code optimization and documentation enhancement.

Following our initial audit, WisdomTree Digital worked closely with our team to address the identified issues. The proposed changes focused on reinforcing role management integrity, ensuring accurate administrative permission enforcement, and enhancing code efficiency and documentation clarity to strengthen the overall security and reliability of the smart contracts. Through multiple rounds of interaction, all identified issues have been addressed or formally acknowledged.

As a result, the token standard has passed our audit. Our auditors have verified that the ERC20 Revocable Compliance Standard, as of audited commit `0053fed9d3c55e24fa16875211a077dd03b4d92a`, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

2 FINDINGS REPORT

2.1 CRITICAL

C-04	`DEFAULT_ADMIN_ROLE` can be assigned to more addresses than `MAX_ADMINS` in `AccessControl`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `revokeDefaultAdminRole`	243

Description

In the function `revokeDefaultAdminRole` of contract `AccessControl`, there is no check to ensure that the provided address `account` has the `DEFAULT_ADMIN_ROLE`.

This leads to the possibility of calling `revokeDefaultAdminRole` with an address `account` that does not have the `DEFAULT_ADMIN_ROLE`, which would incorrectly decrement the `_adminCount` counter. Consequently, there could be more admins with the `DEFAULT_ADMIN_ROLE` than the `_adminCount` counter indicates. This allows assigning roles to more addresses than the specified `MAX_ADMINS`.

Recommendation

We recommend adding a check to ensure that the address passed to the `revokeDefaultAdminRole` function has the `DEFAULT_ADMIN_ROLE`:

```
require(
    hasRole(DEFAULT_ADMIN_ROLE, account),
    "grantDefaultAdminRole: Incorrect Account Role"
);
```

Update

Fixed in commit `d2353d56ec0bb4d252222f5e4ebe632a14a0a7a7`

Client's response

Implemented auditor's recommendation by adding the appropriate `require`

C-05	Risk of admin control loss due to missing decrement for `lastDelegatedAdmin` in `AccessControl`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_grantRole`	328

Description

The `_grantRole` function in the `AccessControl` contract allows assigning the `DEFAULT_ADMIN_ROLE` to the same `account` multiple times, causing the `_adminCount` to increase without actually increasing the number of admins. This discrepancy enables the last remaining admin to call the `revokeDefaultAdmin` function, effectively renouncing their `DEFAULT_ADMIN_ROLE` and leaving the protocol without any admins:

```
require(_adminCount > 1, "revokeDefaultAdminRole: Cannot have less than one admin");
```

If the protocol is left without an admin holding the `DEFAULT_ADMIN_ROLE`, no new admins can be assigned. This could lead to a complete loss of administrative control over the protocol.

Recommendation

We recommend incrementing the admin counter only when adding new admins who do not already hold the `DEFAULT_ADMIN_ROLE`.

Update

Fixed in commit `c36b7588bcb1a6bd47a1b1c5ddce9c94ac191d99`

Client's response

Implemented auditor's recommendation by adding an appropriate require for incrementing the admin counter

2.2 MAJOR

M-08	Inability to reassign delegate roles due to unmanaged `lastDelegatedAdmin` counter in `AccessControl`
Severity	**MAJOR**
Status	● FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `grantDelegateAdminRole`	182

Description

The `grantDelegateAdminRole` function in the `AccessControl` contract increments the `lastDelegatedAdmin` counter when assigning the `DELEGATED_ADMIN_ROLE` to an address. However, this counter is not decremented when the role is revoked. As a result, the counter could eventually reach its limit, `MAX_DELEGATES`, and prevent new delegates from being added, even if previous delegates have been removed.

Recommendation

We recommend implementing a mechanism to decrement the `lastDelegatedAdmin` counter in the `revokeDelegateAdminRole` function whenever a delegate role is revoked.

Update

Fixed in commits `598cbcda125b9fe22f524751b03a38a8f931b4c9`.

Client's response

Implemented auditor's recommendation by adding a decrement to the lastDelegatedAdmin when a delegate is revoked using revokeDelegateAdminRole

M-09	Retention of Delegate Roles After `DELEGATED_ADMIN_ROLE` Removal in `AccessControl`
Severity	**MAJOR**
Status	● FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `revokeDelegateAdminRole`	220

Description

In the function `revokeDelegateAdminRole` of contract `AccessControl`, the removal of the `DELEGATED_ADMIN_ROLE` can be done either by the admin who introduced the delegate or any admin with the `DEFAULT_ADMIN_ROLE`.

However, if an admin is removed from the `DEFAULT_ADMIN_ROLE`, their delegates retain their `DELEGATED_ADMIN_ROLE`. In contrast, if a delegate is removed from the `DELEGATED_ADMIN_ROLE`, all their subordinate delegates also lose their roles.

Additionally, there is a discrepancy between the code logic and the documentation. From the code, it appears that:

- Not only an admin with the `DEFAULT_ADMIN_ROLE` can add and remove delegates, but other delegates can also do so.
- An address that loses the `DEFAULT_ADMIN_ROLE` retains the ability to remove delegates (their own).

However, the documentation for `AccessControl` states:

```
The DELEGATED_ADMIN_ROLE role can only be assigned (delegated) or revoked by
DEFAULT_ADMIN_ROLE.
```

Recommendation

We recommend considering the removal of the `DELEGATED_ADMIN_ROLE` from all delegates dependent on the default admin when the admin loses their `DEFAULT_ADMIN_ROLE`.

Additionally, update the documentation on working with delegates to avoid misunderstandings and discrepancies with the code.

Update

Fixed in commits `44054c0b1a163dccb3cba9ccb1066b57d0e46c34`, `1b61b5ec9f0061104b7b8bc815c6eac8cc8c7ba4`

Client's response

Implemented auditor's recommendation by ensuring that before revoking the DEFAULT_ADMIN_ROLE, the function iterates through the delegates array of the admin and calls _recursiveRemoveDelegate on each delegate. This ensures that all delegates and their sub-delegates are removed when the admin loses their DEFAULT_ADMIN_ROLE. After removing all delegates, the function proceeds to revoke the DEFAULT_ADMIN_ROLE from the specified admin.

M-10	Ineffective Recursion in Delegate Removal Leaves Parent and Descendant Links Unchanged in AccessControl
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `_recursiveRemoveDelegate`	359

Description

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.

1) 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.

W-11	Missing compliance check for `burn` and `batchBurn` in `ERC20RevocableComplianceStandard`
Severity	**WARNING**
Status	• NO ISSUE

Location

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

Description

In the `ERC20RevocableComplianceStandard` contract, compliance checks are added for all token operations. However, the `burn` and `batchBurn` functions are not overridden in the contract, allowing tokens to be burned without compliance checks.

This means that an attacker with stolen tokens, who has been restricted from transferring funds in compliance, can still burn them.

Recommendation

We recommend adding a compliance check to the `burn` and `batchBurn` functions to ensure that compliance rules can prevent unauthorized token burning.

Update

Client's response

This is as intended

W-12	A single address can have both `DEFAULT_ADMIN_ROLE` and `DELEGATED_ADMIN_ROLE` simultaneously in `AccessControl`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `grantDefaultAdminRole`	149

Description

In the function `grantDefaultAdminRole` of the `AccessControl` contract, there is no check to ensure that the specified address does not already have the `DEFAULT_ADMIN_ROLE` or `DELEGATED_ADMIN_ROLE`. However, such a check exists in the `grantDelegateAdminRole` function:

```
require(!hasRole(DELEGATED_ADMIN_ROLE, account), "_grantDelegateAdminRole: account already
has this role");
require(
    !hasRole(DEFAULT_ADMIN_ROLE, account),
    "_grantDelegateAdminRole: DEFAULT_ADMIN_ROLE accounts cannot be assigned
DELEGATED_ADMIN_ROLE"
);
```

As a result, a user can first obtain the `DELEGATED_ADMIN_ROLE` and then receive the `DEFAULT_ADMIN_ROLE`, holding both roles simultaneously. This situation can lead to conflicts, such as when removing delegates.

For example, if Alice with the `DELEGATED_ADMIN_ROLE` delegates this role to Bob, and Bob subsequently receives the `DEFAULT_ADMIN_ROLE` through the `grantDefaultAdminRole` function and adds their own delegate, Carol, then Alice can remove Carol's delegate role by calling `grantDelegateAdminRole(B)`, as default admin Bob is still considered a delegate of Alice:

```
delegateAdmin_Alice -> delegateAndDefaultAdmin_Bob -> delegateAdmin_Carol
```

Recommendation

We recommend adding a check to the `grantDefaultAdminRole` function to ensure that the specified address does not have delegate or default admin rights, as is done in the `grantDelegateAdminRole` function.

Update

Fixed in commit `89ed3c629fcaa8e8bb5903b9b7afbbf57ac5c944`.

Client's response

Implemented auditor's recommendation.

W-13	totalSupply is not decreased in the _transfer function when to is zero in BaseERC20
Severity	**WARNING**
Status	● FIXED

Location

File	Location	Line
BaseERC20.sol	contract BaseERC20 > function _transfer	354

Description

In the `_transfer` function of the `BaseERC20` contract, the value of `totalSupply` is not decreased when `to == 0`. At the same time, `totalSupply` is increased when `from == 0`:

```
if (from == address(0)) {
    uint256 currentTotalSupply = StorageSlot.getUint256Slot(_TOTAL_SUPPLY_SLOT).value;
    StorageSlot.getUint256Slot(_TOTAL_SUPPLY_SLOT).value = currentTotalSupply + value;
} else {
    StorageSlot.getUint256Slot(keccak256(abi.encode(_BALANCES_SLOT, from))).value =
fromBalance - value;
}

uint256 toBalance = StorageSlot.getUint256Slot(keccak256(abi.encode(_BALANCES_SLOT,
to))).value;
StorageSlot.getUint256Slot(keccak256(abi.encode(_BALANCES_SLOT, to))).value = toBalance +
value;
```

This discrepancy may lead to mismatches between `totalSupply` and user balances if the function is called with `to == 0`. In the current version of the code, it is not possible to pass `to == 0`. However, in future versions, the internal function `_transfer` might be reused in a way that could introduce this possibility.

Additionally, the inconsistent behavior for `from == 0` and `to == 0` creates a logical inconsistency in handling `totalSupply`.

Recommendation

We recommend adding logic to decrease `totalSupply` when `to == 0`, similar to the logic used when `from == 0`.

It is also worth noting that the logic for `from == 0` and `to == 0` can be reused in the `_mint` and `_burn` functions, respectively, to avoid code duplication.

Update

Fixed in commit `4dde06131dfe6dcea85a45acc2b0cd613047dd0e`.

Client's response

Implemented auditor's recommendation

2.4 INFO

I-18	`StorageSlot` library is not used for storing the `_complianceAddress` variable in `ERC20RevocableComplianceStandard`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard`	17

Description

In the `ERC20RevocableComplianceStandard` contract, the `_complianceAddress` variable is set in the contract state. At the same time, the base contract uses the `StorageSlot` library for storing variables.

Recommendation

We recommend using the `StorageSlot` library for storing the `_complianceAddress` variable to maintain consistency with the rest of the protocol.

Update

Fixed in commit `f25a9370537a2bc18af7fe859796b0395d001069`

Client's response

Implemented auditor's recommendation

I-19	Insufficient documentation for functions modifying `_complianceAddress` in `ERC20RevocableComplianceStandard`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `setCompliance`	172
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `removeCompliance`	185

Description

In the mentioned locations, the functions that modify `_complianceAddress` are accessible only to admins with the `DEFAULT_ADMIN_ROLE`. However, these admins can delegate their rights by creating admins with the `DELEGATED_ADMIN_ROLE`.

The documentation does not clarify who is authorized to invoke the functions for changing `_complianceAddress`—whether it is the default admin, the delegated admin, or both—making it unclear if the described access rights in the code are accurate.

Recommendation

We recommend updating the documentation to clarify who has the authority to modify `_complianceAddress`. If delegated admins should have this right, the access permissions for the `setCompliance` and `removeCompliance` functions should be adjusted accordingly.

Update

Client's response

We will update the documentation accordingly

I-20	Redundant inheritance of the `Context` contract when inheriting `AccessControl` in `BaseERC20`
Severity	**INFO**
Status	● FIXED

Location

File	Location	Line
Beacon.sol	-	15
BaseERC20.sol	contract `BaseERC20`	15

Description

In the mentioned locations, the contracts inherit both `AccessControl` and `Context`. However, the `Context` contract is already inherited within `AccessControl`.

Recommendation

We recommend not inheriting the `Context` contract in these cases to maintain a clean codebase.

Update

Fixed in commit `57ffcc6793b20990721a8ecacd835098db723914`

Client's response

Implemented auditor's recommendation

I-21	Unused `Context` contract functions in `AccessControl`, `BaseERC20`, `Beacon`, `Proxy`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
AccessControl.sol	contract `AccessControl`	46
Beacon.sol	contract `Beacon`	15
Proxy.sol	contract `Proxy`	12
BaseERC20.sol	contract `BaseERC20`	15

Description

In the mentioned locations, the `Context` contract is inherited, which defines two internal functions: `_msgSender` and `_msgData`. However, neither of these functions are used in the code.

Recommendation

We recommend considering the removal of the `Context` contract from the protocol to maintain a clean codebase.

Update

Fixed in commit `57ffcc6793b20990721a8ecacd835098db723914`

Client's response

Implemented auditor's recommendation

I-22	Redundant overloading of the `_checkRole` function in `AccessControl`
Severity	**INFO**
Status	● NO ISSUE

Location

File	Location	Line
AccessControl.sol	contract `AccessControl`	372

Description

In the `AccessControl` contract, there are two functions `_checkRole` with the signatures `_checkRole(bytes32 role)` and `_checkRole(bytes32 role, address account)`. The first function is used only to call the second:

```
function _checkRole(bytes32 role) internal view virtual {
    _checkRole(role, msg.sender);
}
```

Recommendation

We recommend removing the `_checkRole(bytes32 role)` function to maintain a clean codebase and calling `_checkRole(bytes32 role, address account)` directly.

Update

Client's response

This is intentional as both _checkRole's and their parameters have use-cases

I-23	Array length check after operations in `ERC20RevocableComplianceStandard`
Severity	**INFO**
Status	● FIXED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchTransfer`	108
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchMint`	134

Description

In the mentioned locations, the array length is assigned to a separate variable, and a check is performed to ensure the input arrays have matching lengths:

```
address[] memory from = new address[](_toList.length);
for (uint256 i = 0; i < _toList.length; ++i) {
    from[i] = msg.sender;
}
require(_areWhitelisted(from, _toList, _amounts), "batchTransfer: One or more addresses not whitelisted");

uint256 len_ = _toList.length;
require(len_ == _amounts.length, "batchTransfer: Inconsistent input array lengths");
```

However, in the specified locations, these checks occur after the arrays are validated for compliance using the `_areWhitelisted` function.

Recommendation

We recommend moving the above lines to the beginning of the functions, before the compliance array checks, to optimize gas usage and improve code organization.

Update

Fixed in commit `930ebbf263bb6333ce6534b91c87625ce0944502`

Client's response

Implemented auditor's recommendation

I-24	Redundant zero value assignment to an empty array in `ERC20RevocableComplianceStandard`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
ERC20RevocableComplianceStandard....	contract `ERC20RevocableComplianceStandard` > function `batchMint`	130

Description

In the `batchMint` function of the `ERC20RevocableComplianceStandard` contract, an empty array `from` is initialized. Then, the elements of the array are assigned zero values:

```
address[] memory from = new address[](_toList.length);
for (uint256 i = 0; i < _toList.length; ++i) {
    from[i] = address(0);
}
```

However, the empty `from` array already contains zero values, making the zero-assignment loop redundant.

Recommendation

We recommend removing the code that fills the array with zero addresses to optimize gas usage and maintain code clarity.

Update

Fixed in commit `3afc71c70ecddec00de1a99296a3a613e7e58f5e`

Client's response

Implemented auditor's recommendation

I-25	Arbitrary `decimals` size can be set in `BaseERC20`
Severity	**INFO**
Status	● FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `initializeWithRoles`	116

Description

In the `initializeWithRoles` function of the `BaseERC20` contract, the `decimals` value is set, which must be greater than `0`. However, there is no check for the maximum value of `decimals`.

This could lead to an overflow when performing operations with very large `decimals`, such as multiplication.

Recommendation

We recommend limiting the `decimals` value to a maximum that is reasonable within the protocol.

Update

Fixed in commit `ac5ec89bf7415eb34da85d6eea8e5dea535f0310`

Client's response

Implemented auditor's recommendation

I-26	No overflow check for `totalSupply` in `BaseERC20`
Severity	**INFO**
Status	● FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `_transfer`	349
BaseERC20.sol	contract `BaseERC20` > function `_mint`	372

Description

In the mentioned locations, `totalSupply` is increased by the `value`. However, there is no overflow check for this operation.

This could result in an uninformative error message if an attempt is made to increase `totalSupply` beyond its maximum value.

Recommendation

We recommend adding a condition to check for overflow in `totalSupply`. For example, it could look like this:

```
require(type(uint256).max - value >= currentTotalSupply, "_function: totalSupply overflow");
StorageSlot.getUint256Slot(_TOTAL_SUPPLY_SLOT).value = currentTotalSupply + value;
```

Update

Fixed in commit `7f1a65e9ced62c1acb93242e922e9cb8c5de54a8`

Client's response

Implemented auditor's recommendation

I-27	`BeaconChanged` event emitted twice in `BaseERC20`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BaseERC20.sol	contract `BaseERC20` > function `upgradeBeaconToAndCall`	161
BaseERC20.sol	contract `BaseERC20` > function `_setBeacon`	179

Description

In the `upgradeBeaconToAndCall` function of the `BaseERC20` contract, the `BeaconChanged` event is emitted twice: once in the `upgradeBeaconToAndCall` function and again in the `_setBeacon` function.

```
emit BeaconChanged(previousBeacon, newBeacon);
```

and then in the `_setBeacon` function:

```
emit BeaconChanged(address(0), newBeacon);
```

This duplication can lead to confusion when using monitoring systems, as the last log entry will be `emit BeaconChanged(address(0), newBeacon)`, indicating `previousBeacon=0`.

Recommendation

We recommend emitting the event only once with valid `previousBeacon` and `newBeacon` values to ensure clarity and accuracy in logs.

Update

Fixed in commit `5aa757b93060a3a73a7c7dc046b95884caa7b5dd`

Client's response

Implemented auditor's recommendation

I-28	`DELEGATED_ADMIN_ROLE` cannot call `batchGrantDelegateAdminRole` in `AccessControl`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
AccessControl.sol	contract `AccessControl` > function `grantDelegateAdminRole`	168
AccessControl.sol	contract `AccessControl` > function `batchGrantDelegateAdminRole`	198

Description

At the mentioned locations, the `DELEGATED_ADMIN_ROLE` role is assigned. However, the `grantDelegateAdminRole` function is available to an admin with `DELEGATED_ADMIN_ROLE`, while the `batchGrantDelegateAdminRole` function is only available to admins with `DEFAULT_ADMIN_ROLE`.

Recommendation

We recommend considering adding the ability for `DELEGATED_ADMIN_ROLE` to call the `batchGrantDelegateAdminRole` function to ensure overall consistency in the code logic.

Update

Client's response

This is as intended

3 APPENDIX

3.1 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.

3.2 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.

3.2.1 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.

3.3 FINDINGS CLASSIFICATION REFERENCE

3.3.1 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.

3.3.2 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.

3.4 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

OXORIO