CLASH OF COINS SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

MARCH 12, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Clash of Coins's smart contracts.

The Clash Of Coins project implements a series of smart contracts designed to facilitate claim-based mechanisms for an on-chain gaming ecosystem. The core functionality revolves around enabling users to claim rewards based on predefined criteria and interact with the platform's token economy. The contracts are written in Solidity and are optimized for deployment on the Base blockchain. Key features include reward distribution logic, ownership management, and mechanisms to ensure the integrity and security of the claiming process.

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 project's security and functionality. The audit covered a total of 20 smart contracts, encompassing 3338 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Clash of Coins and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

Throughout the audit, a collaborative approach was maintained with Clash of Coins to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by Clash of Coins, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the smart contracts, as of audited commit a8cb7c624367a5381e97398a181ab1780c513abf, has met the security and functionality requirements established for this audit, based on the code and documentation provided, and operates as intended within the defined scope.

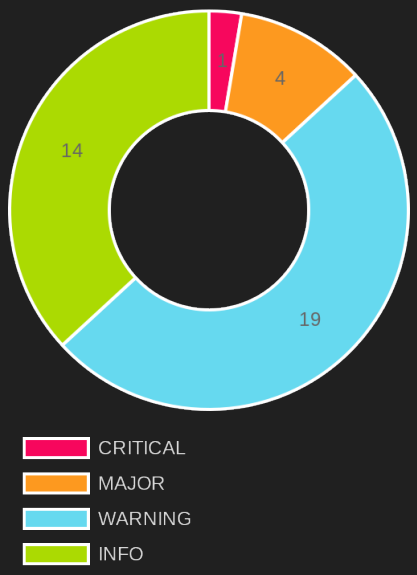
1.2 SUMMARY OF FINDINGS

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.

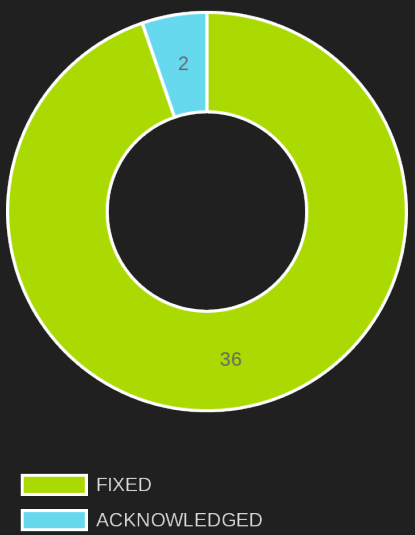
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

All identified issues have been addressed, with Clash of Coins fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**1**	0	1	0	0
MAJOR	**4**	0	4	0	0
WARNING	**19**	0	18	1	0
INFO	**14**	0	13	1	0
TOTAL	**38**	**0**	**36**	**2**	**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	Clash of Coins
Project name	Clash of Coins Smart Contracts
Category	Gaming
Website	https://clashofcoins.com/
Repository	https://github.com/onewayblock/clash-pre-audit-contracts
Documentation	https://github.com/onewayblock/clash-pre-audit-contracts/blob/main/README.md
Initial Commit	d8577a7c4f35c740d5863aeafa97702274543987
Final Commit	a8cb7c624367a5381e97398a181ab1780c513abf
Platform	L2
Network	Base
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Natali Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

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

Date	Event
Fabruary 14, 2025	Client approached Oxorio requesting an audit.
February 18, 2025	The audit team commenced work on the project.
February 19, 2025	Submission of the preliminary report #1.
February 24, 2025	Submission of the preliminary report #2.
February 26, 2025	Submission of the preliminary report #3.
March 4, 2025	Submission of the comprehensive report.
March 11, 2025	Client feedback on the report was received.
March 12, 2025	Submission of the final report incorporating client's verified fixes.

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	contracts/Claim.sol	168	29	36	**103**	25%
2	contracts/erc721a/ERC721A.sol	1579	173	716	**690**	30%
3	contracts/erc721a/IERC721A.sol	331	44	206	**81**	0%
4	contracts/HardCurrencyShop.sol	309	43	58	**208**	23%
5	contracts/interfaces/IClaim.sol	110	22	55	**33**	0%
6	contracts/interfaces/IHardCurrencyShop.sol	101	21	44	**36**	0%
7	contracts/interfaces/INFT.sol	164	30	75	**59**	0%
8	contracts/interfaces/INFTSale.sol	222	39	83	**100**	0%
9	contracts/interfaces/IReferralShare.sol	125	24	57	**44**	0%
10	contracts/interfaces/IUniswapHelper.sol	52	7	27	**18**	0%
11	contracts/interfaces/IVerification.sol	266	34	118	**114**	0%
12	contracts/interfaces/IVesting.sol	190	30	92	**68**	0%
13	contracts/NFT.sol	405	53	92	**260**	19%
14	contracts/NFTSale.sol	658	88	116	**454**	20%
15	contracts/OrdinaryNFTSale.sol	72	4	27	**41**	0%
16	contracts/ReferralShare.sol	320	49	72	**199**	23%
17	contracts/UniswapHelper.sol	239	35	53	**151**	10%
18	contracts/Verification.sol	511	61	90	**360**	13%
19	contracts/Vesting.sol	385	43	92	**250**	14%
20	contracts/WhitelistNFTSale.sol	120	13	38	**69**	9%
	Total		**6327**	**842**	**2147**	**3338**

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

The Clash Of Coins project implements a series of smart contracts designed to facilitate claim-based mechanisms for an on-chain gaming ecosystem. The core functionality revolves around enabling users to claim rewards based on predefined criteria and interact with the platform's token economy. The contracts are written in Solidity and are optimized for deployment on the Base blockchain. Key features include reward distribution logic, ownership management, and mechanisms to ensure the integrity and security of the claiming process. The contracts also leverage access control patterns to maintain operational security.

The primary goal of the project is to establish a transparent and efficient claim and reward system for users participating in the Clash Of Coins game. Key functionalities include managing reward allocation, validating user eligibility, and integrating token-based interactions within the ecosystem. Notable architectural features include role-based access controls and mechanisms to mitigate common smart contract vulnerabilities, ensuring the system's robustness and reliability in a decentralized environment.

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	All roles are defined and implemented correctly. Access control has been verified.	**Excellent**
Arithmetic	Verified library functions are used for computations.	**Good**
Complexity	Contract logic is fairly complex but well-structured.	**Good**
Data Validation	Input validation was missing in some functions, but fixed during reaudit	**Excellent**
Decentralization	The project does not incorporate a decentralized approach to management, and therefore, the metric is not applicable in this context.	**Not Applicable**
Documentation	The project documentation covers all components, is up-to-date, and is centralized in a single source.	**Excellent**
External Dependencies	The project does not interact with any external smart contracts in its logic.	**Not Applicable**
Error Handling	The project demonstrates robust exception handling throughout the codebase. Custom errors with clear naming and descriptions are used.	**Excellent**
Logging and Monitoring	The project exhibits excellent logging capabilities, recording all important events within the system.	**Excellent**
Low-Level Calls	Low-level calls like ether transfers are used cautiously.	**Excellent**
Testing and Verification	Comprehensive testing has been done, but it does not cover all possible scenarios.	**Good**

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
contracts/HardCurrencyShop.sol	**11**	0	1	7	3
contracts/ReferralShare.sol	**7**	0	1	4	2
contracts/Verification.sol	**5**	0	0	3	2
contracts/NFTSale.sol	**4**	0	0	4	0
contracts/Vesting.sol	**4**	0	0	1	3
contracts/HardCurrencyShop.sol	**2**	0	0	2	0
contracts/NFT.sol	**2**	0	0	1	1
contracts/NFTSale.sol	**2**	0	0	1	1
contracts/Claim.sol	**1**	0	0	0	1
contracts/NFT.sol	**1**	0	0	1	0
contracts/UniswapHelper.sol	**1**	0	1	0	0
contracts/UniswapHelper.sol	**1**	0	0	0	1
contracts/Verification.sol	**1**	0	0	1	0
contracts/Verification.sol	**1**	0	0	1	0
contracts/WhitelistNFTSale.sol	**1**	0	1	0	0
contracts/WhitelistNFTSale.sol	**1**	0	0	0	1
contracts/WhitelistNFTSale.sol	**1**	1	0	0	0
contracts/erc721a/ERC721A.sol	**1**	0	0	0	1
contracts/interfaces/IClaim.sol	**1**	0	0	0	1
contracts/interfaces/INFTSale.sol	**1**	0	0	0	1

2.8 CONCLUSION

20 Smart contracts have been audited, and 1 critical and 4 major issues were found. Also a lot of recommendations were marked as a warning and informational. Some changes were proposed to follow best practices, reduce the potential attack surface, simplify code maintenance and increase its readability. The severe attack vectors and possible broken features identified.

As stated in each particular issue, each critical, major, and warning issue identified has been correctly fixed or acknowledged by the client, so contracts are assumed as secure to use according to our security criteria. Final commit identifier with all fixes: `a8cb7c624367a5381e97398a181ab1780c513abf`. This version is recommended to deploy to testnet for further system testing.

To further help the project reach a production-ready state, we highly advise additional rounds of security reviews after every change in contracts.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

C-01	Possible NFT purchase without whitelist verification in `WhitelistNFTSale`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
WhitelistNFTSale.sol	contract `WhitelistNFTSale` > function `buyNFT`	81
WhitelistNFTSale.sol	contract `WhitelistNFTSale` > function `buyNFTFromCrossmint`	112

Description

In the mentioned locations, the contract `WhitelistNFTSale` calls the functions `buyNFT` and `buyNFTFromCrossmint`, which in turn invoke the corresponding functions of the base contract `NFTSale` via `super.*`. However, these functions are not overridden in the `WhitelistNFTSale` contract as they have different signatures:

```
function buyNFT(
    uint256 _saleId,
    uint256 _quantity,
    address _paymentToken,
    uint256 _expectedTokenAmount,
    uint256 _slippageTolerance,
    bytes32[] calldata _merkleProof
) public payable;

function buyNFT(
    uint256 _saleId,
    uint256 _quantity,
    address _paymentToken,
    uint256 _expectedTokenAmount,
    uint256 _slippageTolerance
) public payable;
```

```
function buyNFTFromCrossmint(
    uint256 _saleId,
    address _receiver,
    uint256 _quantity,
    bytes32[] calldata _merkleProof
) public;

function buyNFTFromCrossmint(
    uint256 _saleId,
    address _receiver,
    uint256 _quantity
) public;
```

Thus, in the `WhitelistNFTSale` contract, it is possible to call both the function with whitelist verification and the function with the core purchase logic directly from the base contract.

This renders the whitelist verification functions ineffective.

Recommendation

We recommend refactoring the `NFTSale` contract and its child contracts to ensure that purchase functions in inherited contracts cannot be called directly. One possible approach is to define `NFTSale` as an abstract contract and move the purchase logic into `internal` functions.

Update

Oxorio's response

Fixed at a8cb7c624367a5381e97398a181ab1780c513abf.

3.2 MAJOR

M-01	Incorrect `msg.sender` whitelist check in `WhitelistNFTSale`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
WhitelistNFTSale.sol	contract `WhitelistNFTSale` > function `buyNFT`	80
WhitelistNFTSale.sol	contract `WhitelistNFTSale` > function `buyNFTFromCrossmint`	111

Description

In the mentioned locations, `msg.sender` is checked for whitelist inclusion. However, in the case of the following functions:

- In `buyNFT`, the actual user address may be present in the calldata if `msg.sender == trustedForwarder`.
- In `buyNFTFromCrossmint`, the whitelist check is meaningless if `msg.sender != crossmintAddress`.

It is likely that, in the case of the `buyNFT` function, the whitelist check should be performed on the result of the `_msgSender` function instead of `msg.sender` to verify the actual user address.

Recommendation

We recommend modifying the `buyNFTFromCrossmint` function to check whether the `_receiver` parameter is included in the whitelist, as it is unnecessary to validate `msg.sender`, given that only `crossmintAddress` can call this function.

Update

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

M-02	There is no verification that the `_token` can be zero address in `UniswapHelper`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
UniswapHelper.sol	contract `UniswapHelper` > function `checkPrice`	195

Description

In the `checkPrice` function of the `UniswapHelper` contract, there is no verification that the `_token` can be zero address, this could lead to a situation where, if Ether is used and `_token` equals `address(0)`, the pool will not be found, resulting in a revert error.

Recommendation

We recommend adding a check to ensure that if the `_token` _is `address(0)`, the `WETH` address is used instead.

Update

Oxorio's response

Fixed at `80c438b29e351c6a06ea40c7dde64be9742a9897`.

M-03	Potential manipulation of `totalTokenAmount` value in `HardCurrencyShop`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `purchase`	87

Description

In the `purchase` function of the `HardCurrencyShop` contract, there is a possibility to manipulate `totalTokenAmount` values since `_expectedTokenAmount` and `_slippageTolerance` parameters are not validated. Using techniques like flash loans, one can manipulate the price to maximize the effect, paying less `totalTokenAmount` while gaining more benefit from `_USDAmount`. In essence, the slippage protection is not working as intended.

Recommendation

We recommend relying on oracle values to address this issue. In Uniswap V3, each pool has a built-in oracle that can provide reliable price data.

Additionally, it should be clarified whether the `_expectedTokenAmount` and `_slippageTolerance` values are provided by the backend. If these values are indeed backend-supplied, implementing backend signature verification would be a prudent approach to ensure the integrity and security of the provided data.

Update

Clash of Coins' Response

- We will use only tokens which have USDC pool pair (like ETH, USDT, own token).
- We will block transaction if expected token amount is less than real amount.
- Added additional checks.

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

M-04	A compromised address from the whitelist can withdraw all funds in `ReferralShare`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
	ReferralShare.sol	contract `ReferralShare` > function `recordDeposit`

Description

In the `recordDeposit` function of the `ReferralShare` contract, tokens are added to the balance without actually being transferred to the contract. Additionally, there are no restrictions on the amount by which the balance can be increased.

If one of the addresses in the whitelist is compromised, it can set the maximum possible balance for "its" referral code for each of the tokens stored in the contract. Subsequently, using this referral code, all tokens held in the contract can be withdrawn.

Recommendation

We recommend addressing the vulnerabilities in the `recordDeposit` function of the `ReferralShare` contract by ensuring that tokens are actually transferred to the contract when balances are updated.

Update

Clash of Coins' Response

- Added code fixes
- About compromised addresses. For now, we're using OpenZeppelin Defender Relayer for all backend transactions and we don't have access to the private key of this relayer wallet, so it's impossible to compromise it.

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

3.3 WARNING

W-01	Repeated NFT purchase without condition verification in `NFTSale`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFTSale.sol	contract `NFTSale` > function `buyNFTFromCrossmint`	395

Description

In the function `buyNFTFromCrossmint` of the contract `NFTSale`, the `nonReentrant` modifier is not used, unlike in the similar function `buyNFT`.

If we assume that Crossmint is a contract that calls `buyNFTFromCrossmint` on behalf of users, the following exploit scenario is possible:

- A malicious contract calls `buyNFT`, passes all checks, and reaches the `_handlePayment` function.
- Within `_handlePayment`, during the external call execution, the malicious contract intercepts execution and calls the Crossmint contract, which then purchases an NFT via `buyNFTFromCrossmint`.
- The malicious contract could purchase all remaining `sale.quantity`, setting `sale.soldQuantity` equal to `sale.quantity`.
- The malicious contract then returns execution control back to `buyNFT` and finalizes the original NFT purchase.
- Upon finalization, it increases `sale.soldQuantity` beyond `sale.quantity`. This is possible because all checks have already been passed before the external call in `_handlePayment`.

Thus, the attacker executes two purchases while passing the purchase conditions only once.

Recommendation

We recommend adding the `nonReentrant` modifier to the `buyNFTFromCrossmint` function and adhering to the `Checks-Effects-Interactions` pattern in the NFT purchase logic to mitigate potential reentrancy attacks.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

W-02	Unsafe ether transfer in `HardCurrencyShop`, `ReferralShare`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	114
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	115
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	191
ReferralShare.sol	contract `ReferralShare` > function `withdrawBalances`	107

Description

In the mentioned locations, using the `transfer` method for sending ether is deprecated and vulnerable. It's recommended to use the `call` method instead. More details can be found here.

Recommendation

We recommend avoiding the `transfer` method for sending Ether in the mentioned locations as it is deprecated and potentially vulnerable. Use the `call` method instead, as it is safer and more flexible. Learn more here.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-03	Potential reentrancy in `HardCurrencyShop`, `ReferralShare`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	114
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	115
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_handlePayment`	191
ReferralShare.sol	contract `ReferralShare` > function `withdrawBalances`	107

Description

In the mentioned locations, reentrancy is possible since there are ether operations.

Recommendation

We recommend using OpenZeppelin's `nonReentrant` modifier for user-facing functions to mitigate the risk of reentrancy, which is possible in the mentioned locations due to Ether operations.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-04	Possibility to send ether alongside tokens in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `purchase`	70

Description

In the `purchase` function of `HardCurrencyShop` contract, it's possible to send ether while specifying a token as the payment method. In this case, the ether will be lost and stuck in the contract.

Recommendation

We recommend adding a validation check in the `purchase` function of the `HardCurrencyShop` contract to ensure that ether cannot be sent when a token is specified as the payment method. This will prevent ether from being lost and stuck in the contract.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-05	Unable to reuse function calls with signature in `Verification`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `Verification` > function `_verifySignature`	501

Description

In the `Verification` contract, the `_messageHash` becomes invalid after use

```
    function _verifySignature(
        bytes32 _messageHash,
        bytes memory _signature
    ) private returns (bool) {
        if(usedHashes[_messageHash]) {
            revert InvalidMessageHash();
        }

        usedHashes[_messageHash] = true;

        bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(
            _messageHash
        );
        return ethSignedMessageHash.recover(_signature) == backendSigner;
    }
```

but the hash itself doesn't contain unique parameters like a nonce.
This means that functions requiring signature validation can only be called once, as subsequent calls with the same parameters will be rejected.
For example,

```
        IVerification(verification).verifySignaturePublic(
            keccak256(abi.encode(address(this), 'withdrawBalances', sender, _referralCode,
block.chainid)),
```

```
            _signature
        );

        msgHash =  keccak256(abi.encode(address(this), 'withdrawBalances', sender,
_referralCode, block.chainid))
```

Recommendation

We recommend adding a nonce field to the messageHash, which would be a sequential transaction number, and implementing `nonces[msg.sender]++` increment in the `_verifySignature` function after verification. This way, the current signature becomes invalid after use, and new signatures will differ by their sequential number.

Update

Oxorio's response

Fixed at `d17f9c98ec50bbbcd977dd714102d32b8474f37c`.

W-06	`userVerifications` parameters cannot be changed in `Verification`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Verification.sol	contract `Verification` > function `setBaseKyc`	111
Verification.sol	contract `Verification` > function `setAdvancedKyc`	144
Verification.sol	contract `Verification` > function `setBaseAMLScore`	177
Verification.sol	contract `Verification` > function `setAdvancedAMLScore`	210

Description

In the specified locations, once a value is set in `userVerifications`, it cannot be changed. This leads to the following issues:

- For `baseKyc/advancedKyc`, passing a `bool` parameter becomes meaningless, as only `true` can be set. Calling the setter with `false` has no effect other than emitting an event.
- For `baseAMLScore/advancedAMLScore`, the inability to modify values in `userVerifications` means that a user can maintain a good score even if it deteriorates significantly. The protocol will be unable to revoke the user's access to higher limits.
- In all four setters, an event with the name `*Updated` is emitted, even though the value is only set once and cannot be changed.

Recommendation

We recommend updating the implementation to allow modifications to the `userVerifications` values as needed. This can be achieved by introducing the following changes:

- For `baseKyc/advancedKyc`, ensure the `bool` parameter reflects the intended state by allowing values to be toggled between `true` and `false`, so a proper mechanism exists for revoking KYC statuses when necessary.

- For `baseAMLScore/advancedAMLScore`, implement functionality to update the user's score dynamically. This ensures that a deteriorating score can trigger changes in permissions or access levels, maintaining the protocol's flexibility and security.
- Modify the logic for emitting `*Updated` events so that such events are only emitted when an actual change occurs in the relevant value, preventing unnecessary or misleading event logs that may disrupt tracking or monitoring processes.

Update

Clash of Coins' Response

- Removed check for set status for these fields.
- For KYC we will have possibility to revoke verification for specific cases.
- For AML will be possible to recheck it and change in the contract.

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-07	If `_msgSender` is a contract, it should be able to receive Ether in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_msgSender`	240

Description

In the `_msgSender` function of the `HardCurrencyShop` contract, the returned address may belong to a contract. Therefore, it is necessary to check whether the contract can receive Ether before transferring funds and to return a readable error message if the transfer fails.

Recommendation

We recommend verifying that the returned address in the `_msgSender` function does not belong to a contract with unexpected behavior. Before transferring Ether, ensure the contract can receive funds by simulating a low-level call. Additionally, implement a clear and descriptive error message to handle failed transfers effectively.

Update

Oxorio's response

Fixed at `d17f9c98ec50bbbcd977dd714102d32b8474f37c`.

W-08	Missing `_disableInitializers` call in `HardCurrencyShop`, `ReferralShare`, `Verification`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop`	38
ReferralShare.sol	contract `ReferralShare`	34
Verification.sol	contract `Verification`	69

Description

In the specified locations, the `_disableInitializers` function is not called in the constructor. Given the presence of the `Proxies.sol` file, it is likely that the contracts are used through a proxy.

Recommendation

We recommend calling `_disableInitializers` in the constructor to prevent the contract from being reinitialized and to ensure the constructor is executed only once during deployment.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-09	Lack of validations in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `initialize`	61
ReferralShare.sol	contract `ReferralShare` > function `initialize`	54
ReferralShare.sol	contract `ReferralShare` > function `initialize`	57
Verification.sol	contract `Verification` > function `addAllowedContract`	329

Description

In the `initialize` function of the `HardCurrencyShop` contract, there is no validation to check for duplicate addresses in the array or an empty `_paymentTokens` list.
In the `initialize` function of the `ReferralShare` contract, duplicate addresses in the `_supportedTokens` array are not checked, nor are zero addresses in the `_whitelistedContracts` array.
In the `addAllowedContract` function of the `Verification` contract, there is no validation to check for a zero address in the provided contract.

Recommendation

We recommend adding validation checks to ensure no duplicate or zero addresses are allowed in arrays and to prevent empty lists during initialization or function calls.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-10	Missing check for a zero address in `supportedTokens` for handling Ether in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `initialize`	61

Description

In the `initialize` function of the `HardCurrencyShop` contract, supported tokens for purchases are set. However, there is no validation to check for the presence of a zero address in the list, which is used to support native tokens. For example, setting a zero token as a native token is used in the `ReferralShare` contract.

Recommendation

We recommend adding a validation check in the `initialize` function of the `HardCurrencyShop` contract to ensure that no unintended zero address is present in the list of supported tokens for purchases. This is important because a zero address is often used to represent native tokens, as seen in the `ReferralShare` contract. By explicitly validating the token list, you can avoid accidental inclusion of a zero address, which might lead to unexpected behavior or vulnerabilities. If the intention is to allow a zero address to represent native tokens, this behavior should be documented clearly, and the contract's logic should explicitly handle such cases to ensure consistent and secure operation.

Update

Clash of Coins' Response

Sale can be ONLY in USDC for example, so ETH payment is not required, so removed adding logic also from Referral share contract.

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

W-11	Use of `_mint` instead of `_safeMint` in `NFT`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFT.sol	contract `NFT` > function `mint`	148
NFT.sol	contract `NFT` > function `mintWithSameMetadata`	169

Description

In the mentioned locations, the internal function `_mint` is used for minting NFT tokens. However, to prevent minting tokens for a contract that does not support `ERC721A`, the function `_safeMint` should be used.

Recommendation

We recommend replacing the use of `_mint` with `_safeMint` in the mentioned locations to ensure that tokens are not minted to contracts that do not support `ERC721A`, aligning with safer token transfer standards.

Update

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

W-12	Possible purchases without validation if `baseAmlLimit > _amount > 0` in `Verification`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Verification.sol	contract `Verification` > function `validateSpending`	333

Description

In the function `validateSpending` of the contract `Verification`, if `dailyAmount < baseAmlLimit`, no actions are performed. This means that if the NFT price is lower than `baseAmlLimit`, it is possible to purchase an NFT without validation in the function `validateSpending`.

Additionally, the contracts `NFTSale` and `OrdinaryNFTSale` do not check whether buyers are whitelisted (unlike `WhitelistNFTSale`). Therefore, if the daily limit is reached in the function `validateSpending`, or the `totalLimitPerUser` is exceeded, it is possible to continue purchasing NFTs from another arbitrary address.

A similar issue is also present when making purchases in the contract `HardCurrencyShop`.

Recommendation

We recommend modifying the function `validateSpending` in the `Verification` contract to ensure validation is performed regardless of whether `dailyAmount < baseAmlLimit`. Additionally, the contracts `NFTSale`, `OrdinaryNFTSale`, and `HardCurrencyShop` should enforce whitelist checks and shared spending limits across addresses to prevent circumvention of `dailyLimit` and `totalLimitPerUser`.

Update

Clash of Coins' Response

- We allow to buy less than dailyAmount without any aml verifications, so it's not an issue
- Ordinary NFT Sale is made for selling NFT to everyone without any restrictions (exclude KYC/AML verifications), and we have Whitelist NFT Sale where will be listed positions only available for buying from whitelisted addresses.

W-13	Unable to delist when `soldQuantity == 0` in `NFTSale`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFTSale.sol	contract `NFTSale` > function `delistNFTFromSale`	221

Description

In the function `delistNFTFromSale` of contract `NFTSale`, there is a check requiring at least one token sale to proceed with delisting. This results in an erroneously listed `NFTSale` being impossible to remove from the listing.

Additionally, the user will receive the error `SaleDoesNotExist`, which is incorrect since the `NFTSale` does exist.

Recommendation

We recommend updating the logic to check whether `quantity == 0` instead of `soldQuantity == 0`. This adjustment ensures the validation is performed based on the intended input parameter rather than the tracking variable, thereby reducing potential logical errors and improving code correctness.

Update

Clash of Coins' Response

- Added code fixes
- For now we will remove only if quantity > 0 (sale exists) and no NFT's were sold.

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

W-14	Vesting start can be far in the past in `Vesting`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Vesting.sol	contract `Vesting` > function `createVestingSchedule`	118

Description

In the function `createVestingSchedule` of contract `Vesting`, there is no validation for `_start`.

As a result, it is possible to set the vesting start far in the past, causing the vesting to be already completed. This does not make sense, as the beneficiary would be able to withdraw all funds from the contract immediately after creating the vesting schedule.

Recommendation

We recommend adding validation for `_start` in the `createVestingSchedule` function of the `Vesting` contract to ensure the start date is not set in the past, preventing scenarios where the vesting schedule is already completed upon creation.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

W-15	`msg.sender` as `_msgSender()` in `NFTSale`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFTSale.sol	contract `NFTSale` > function `buyNFTFromCrossmint`	396
NFTSale.sol	contract `NFTSale` > function `buyNFTFromCrossmint`	421

Description

In the mentioned locations instead of using `msg.sender`, `_msgSender()` should be used to allow transactions to also be sent through a trusted forwarder.

Recommendation

We recommend changing `msg.sender` to `_msgSender()`.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

W-16	`_receiver` can be zero address in `NFTSale`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFTSale.sol	contract `NFTSale` > function `buyNFTFromCrossmint`	433

Description

In the function `buyNFTFromCrossmint` of contract `NFTSale`. There is no check that `_receiver` is not a zero address.

Recommendation

We recommend adding check that `_receiver` is not zero.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

W-17	`msg.sender` becomes the owner of the contract in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `initialize`	59
NFTSale.sol	contract `NFTSale` > function `__NFTSale_init_unchained`	99

Description

In the mention locations `msg.sender` or deployer becomes the owner of the contract.

Recommendation

We recommend adding separate variable `_owner` to `intitialize` function to set the owner's address.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

W-18	`_usdAmount` can be zero in `HardCurrencyShop`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `purchase`	81

Description

In the function `purchase` of contract `HardCurrencyShop`. There is no check that `_usdAmount` is not zero and can cause ddos attacks with zero values.

Recommendation

We recommend adding that `_usdAmount` is not zero address.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

W-19	Possibility to send ETH to the contract in `NFT`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NFT.sol	contract `NFT` > function `approve`	372

Description

In the function `approve` of the contract `NFT`, the function from the base contract `ERC721A` is overridden along with the `payable` keyword. However, receiving ETH is not relevant for the `NFT` contract. As a result, a user can call the `approve` function and send ETH, which will become locked in the contract since there are no withdrawal mechanisms for ETH.

Recommendation

We recommend adding a check to ensure `msg.value` is zero when calling the `approve` function to prevent ETH from being sent to the contract.

Update

Oxorio's response

Fixed at `a8cb7c624367a5381e97398a181ab1780c513abf`.

3.4 INFO

I-01	Ambiguity of Crossmint support in the interface `INFTSale.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
INFTSale.sol	interface `INFTSale`	211

Description

In the `INFTSale` interface, there is no function intended for interaction with Crossmint, yet a function for setting the Crossmint address on the contract is present.

Recommendation

We recommend adding the function `buyNFTFromCrossmint` to the `INFTSale` interface or removing the `changeCrossmintAddress` function from the interface to resolve the ambiguity.

I-02	Unused function in `Verification`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Verification.sol	contract `Verification` > function `setTrustedForwarder`	566

Description

The `setTrustedForwarder` function in the `Verification` contract is not used, nor is the `trustedForwarder` variable.

Recommendation

We recommend removing the `setTrustedForwarder` function and the unused `trustedForwarder` variable from the `Verification` contract to reduce unnecessary code and potential attack surface, improving the overall security and maintainability of the contract.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

I-03	Array length determination should be assigned to a separate memory variable in `HardCurrencyShop`, `ReferralShare`, `UniswapHelper`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_isTokenSupported`	204
HardCurrencyShop.sol	contract `HardCurrencyShop` > function `_removePaymentToken`	218
ReferralShare.sol	contract `ReferralShare` > function `withdrawBalances`	98
UniswapHelper.sol	contract `UniswapHelper` > function `getFeeTier`	98

Description

In the specified locations, the array length is determined within the loop for each iteration while the array resides in storage. For gas optimization, this determination should be assigned to a separate memory variable.

Recommendation

We recommend assigning the array length to a separate memory variable outside the loop to optimize gas usage when the array resides in storage.

Update

Oxorio's response

Fixed at `bb6ae7cd910dd5e0e42bb4dc1f8c2bd431d95221`.

I-04	`Initializable` is already inherited within `OwnableUpgradeable` in `HardCurrencyShop`, `ReferralShare`, `Verification`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	contract `HardCurrencyShop`	20
ReferralShare.sol	contract `ReferralShare`	16
Verification.sol	contract `Verification`	16

Description

In the specified locations, explicit inheritance from the `Initializable` contract is unnecessary, as its logic is already inherited through `OwnableUpgradeable`.

Recommendation

We recommend removing the explicit inheritance from the `Initializable` contract, as its logic is already inherited through `OwnableUpgradeable`.

Update

Oxorio's response

Fixed at `d17f9c98ec50bbbcd977dd714102d32b8474f37c`.

I-05	Unused imports in `HardCurrencyShop.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
HardCurrencyShop.sol	-	10
HardCurrencyShop.sol	-	11

Description

In the mentioned locations, unused imports are present.

Recommendation

We recommend removing there unused imports.

Update

Oxorio's response

Fixed at `d17f9c98ec50bbbcd977dd714102d32b8474f37c`.

I-06	Royalty reset due to rounding during division in `NFT`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NFT.sol	contract `NFT` > function `royaltyInfo`	280

Description

In the function `royaltyInfo` of contract `NFT`, `royaltyAmount` may become `0` if the product of `_salePrice` and `royaltyBasisPoints` is less than `10000` due to rounding during division:

```
royaltyAmount = (_salePrice * royaltyBasisPoints) / 10000;
```

It is important to consider that some tokens have a small `decimals` value. For example, the GUSD token has `decimals = 2`. In this case, when purchasing for `$1` with a `0.5%` royalty rate, the royalty will be rounded down to zero.

Recommendation

We recommend revising the `royaltyInfo` function in the `NFT` contract to account for potential rounding issues when calculating `royaltyAmount`, as it may become `0` if the product of `_salePrice` and `royaltyBasisPoints` is less than `10000`. This is particularly important for tokens with small `decimals` values (e.g., GUSD with `decimals = 2`), where small purchase amounts (e.g., `$1` with a `0.5%` royalty rate) could result in the royalty being rounded down to zero.

Update

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

I-07	Unused imports in `NFTSale.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NFTSale.sol	-	7
NFTSale.sol	-	15

Description

In the mentioned locations, contracts are imported but not used in the code.

Recommendation

We recommend removing there unused imports.

Update

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

I-08	Unused `_fee` parameter in `WhitelistNFTSale`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
WhitelistNFTSale.sol	contract `WhitelistNFTSale` > function `buyNFT`	74

Description

In the function `buyNFT` of contract `WhitelistNFTSale`, the `_fee` parameter is not used.

Recommendation

We recommend removing this parameter.

Update

Oxorio's response

Fixed at `8f865a4053320b4eee456979b977a22a2a22a477`.

I-09	Overflow when exceeding the `quantity` limit in `ERC721A.sol`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ERC721A.sol	-	18

Description

In `ERC721A.sol#L18`, there is an assumption that a single owner cannot hold more than `2**64-1` tokens:

```
 * Assumptions:
 *
 * - An owner cannot have more than 2**64 - 1 (max value of uint64) of supply.
```

At the same time, when minting tokens in the `NFT` contract, the `quantity` parameter of type `uint256` is passed to the `_safeMint` function without validation.

As a result, an overflow may occur during minting because `quantity` is stored in the allocated `64` bits of the `_packedAddressData` variable, and the operation is performed within an `unchecked` block:

```
_packedAddressData[to] += quantity * ((1 << _BITPOS_NUMBER_MINTED) | 1);
```

Recommendation

We recommend adding an additional check in the minting functions `mint` and `mintWithSameMetadata` of the `NFT` contract to ensure that the passed `quantity` does not exceed the limit of `2**64 - 1`.

I-10	Contract accepts and does not use ether in `Vesting`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Vesting.sol	contract `Vesting`	62

Description

The contract `Vesting` can accept ether, but it is not used in any contract logic.

Recommendation

We recommend removing this logic.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

I-11	Variable `unlockTime` can be `immutable` in `Claim`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Claim.sol	contract `Claim`	31

Description

In the contract `Claim`, the variable `unlockTime` is set in the constructor and cannot be changed later. It should be declared as `immutable`.

Recommendation

We recommend declaring the variable `unlockTime` as `immutable`.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

I-12	Error `InvalidTokensToStake` is not used in `IClaim.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
IClaim.sol	interface `IClaim`	14

Description

In the interface `IClaim`, the custom error `InvalidTokensToStake` is declared but not used anywhere in the contracts.

Recommendation

We recommend removing this error.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

I-13	Redundant zero checks in `Vesting`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Vesting.sol	contract `Vesting` > function `createVestingSchedule`	99
Vesting.sol	contract `Vesting` > function `createVestingSchedule`	102
Vesting.sol	contract `Vesting` > function `createVestingSchedule`	105

Description

In the mentioned locations, `unsigned` variables are checked for being less than zero, which is redundant.

```
if (_duration <= 0) {
// ...
if (_amount <= 0) {
// ...
if (_slicePeriodSeconds < 1) {
// ...
```

Recommendation

We recommend simplifying the checks to equality/inequality with zero for improved clarity and efficiency.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

I-14	Codebase simplification in `Vesting`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Vesting.sol	contract `Vesting` > function `createVestingSchedule`	130

Description

In the mentioned locations, the code can be simplified for clarity:

```
holdersVestingCount[_beneficiary]++;
// ...
vestingSchedule.released += vestedAmount;
// ...
vestingSchedulesTotalAmount -= vestedAmount;
```

Recommendation

We recommend refactoring this code.

Update

Oxorio's response

Fixed at `bef76ee0fa3c5e44c7409ae6f37409d4efd89bb5`.

4. APPENDIX

4.1 DISCLAIMER

At the request of client, Oxorio consents to the public release of this audit report. The information contained in this audit report is provided "as is," without any representations or warranties whatsoever. Oxorio disclaims any responsibility for damages that may arise from or in relation to this audit report. Oxorio retains copyright of this report.

The audit makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

4.2 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.

4.3 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.

4.3.1 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.

4.4 FINDINGS CLASSIFICATION REFERENCE

4.4.1 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.

4.4.2 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.

4.5 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, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, 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