BORED
GHOSTS
DEVELOPING

AAVE V3 LIQUID EMODES SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

SEPTEMBER 12, 2024

1 EXECUTIVE SUMMARY

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for BGD Labs's Aave V3 Liquid eModes Update.

BGD Labs (Bored Ghosts Developing Labs) is a team of technical contributors founded by three core members of the Aave community with extensive expertise in Aave and decentralized finance. Their mission is to provide core development support for the Aave ecosystem, ensuring the security and continuous innovation of the protocol while maintaining an open and collaborative approach. BGD Labs is dedicated exclusively to Aave, focusing on creating open-source solutions for the benefit of the Aave DAO and its community.

The Liquid eModes update in Aave v3 introduces enhanced flexibility and precision to the existing eMode functionality. This update allows assets to be eligible for multiple eModes, enabling more versatile configurations while maintaining the rule that only one eMode can be active at a time. Additionally, Liquid eModes provide granular controls for enabling or disabling assets as collateral or borrowable within specific eModes. Key changes include the removal of the unused eMode oracle, the introduction of bitmask-based asset configurations, and improvements to health factor validations and governance control, ensuring a more adaptable risk management framework.

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 a changes in total of 13 smart contracts, encompassing 2709 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with BGD Labs 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 BGD Labs to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by BGD Labs, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the Aave V3 Liquid eModes, as of audited commit `740d0a8163197a4948c959078258c12254207bef`, 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.

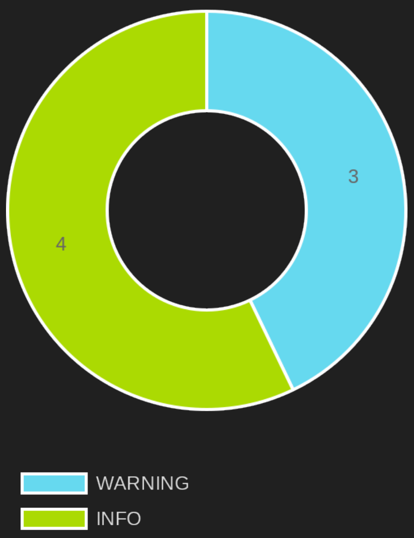
1.2 SUMMARY OF FINDINGS

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.

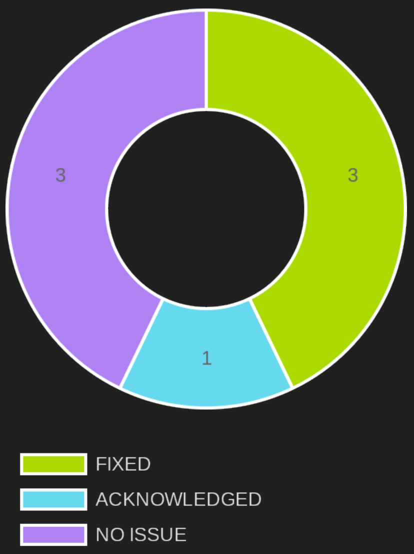
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Finding Report section for further reference.

All identified issues have been addressed, with BGD Labs fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**0**	0	0	0	0
WARNING	**3**	0	1	0	2
INFO	**4**	0	2	1	1
TOTAL	**7**	**0**	**3**	**1**	**3**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

1. EXECUTIVE SUMMARY 2

1.1. EXECUTIVE SUMMARY 3

1.2. SUMMARY OF FINDINGS 4

2. AUDIT OVERVIEW 5

2.1. DISCLAIMER 8

2.2. PROJECT BRIEF 9

2.3. PROJECT TIMELINE 10

2.4. AUDITED FILES 11

2.5. PROJECT OVERVIEW 12

2.6. FINDINGS BREAKDOWN BY FILE 13

2.7. CONCLUSION 14

3. FINDINGS REPORT 15

3.1. CRITICAL 16

3.2. MAJOR 17

3.3. WARNING 18

W-01 Possibility to modify the bitmap for non-existent emode categories in PoolConfigurator 18

W-02 A new reserve upon initialization may unintentionally be included in an emode category in EModeConfiguration 20

W-03 Missing validation for the existence of asset in PoolConfigurator 23

3.4. INFO 25

I-01 Redundant casting to uint8 in PoolConfigurator 25

I-02 Unused field eModeAssetCategory in GenericLogic 26

I-03 Emode category id cast to uint256 in Pool 27

I-04 Same error description for different errors within the setEModeCategory function in PoolConfigurator 28

4. APPENDIX 29

4.1. SECURITY ASSESSMENT METHODOLOGY 30

4.2. FINDINGS CLASSIFICATION REFERENCE 32

Severity Level Reference 32

Status Level Reference 32

4.3. ABOUT OXORIO 34

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	BGD Labs
Project name	Aave V3 Liquid eModes
Category	Lending
Website	https://aave.com/
Repository	https://github.com/bgd-labs/aave-v3-origin-oxorio/
Documentation	https://github.com/bgd-labs/aave-v3-origin-oxorio/pull/1
Initial Commit	`a4849111a0ce57e3af1ca5cd9a9b8c6a8cdad1e0`
Final Commit	`740d0a8163197a4948c959078258c12254207bef`
Platform	L1, L2
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Natalia Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

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

Date	Event
August 26, 2024	Client engaged Oxorio requesting an audit.
September 3, 2024	The audit team initiated work on the project.
September 3, 2024	A project kickoff call was conducted between the audit team and the client.
September 11, 2024	Submission of the comprehensive audit report.
September 12, 2024	Client's feedback on the report was received.
September 12, 2024	The audit team commenced work on a re-audit of the project.
September 12, 2024	Submission of the final audit report incorporating client’s verified fixes.

2.4 AUDITED FILES

The following table contains a list of the audited files. An audit of all changes in these files starting from commit `2b4315794a004ffc4ea44d283d92a66f18368439` was conducted. 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/contracts/helpers/AaveProtocolDataProvider.sol	295	37	34	**224**	4
2	src/contracts/interfaces/IPoolConfigurator.sol	551	63	360	**128**	0
3	src/contracts/interfaces/IPoolDataProvider.sol	250	22	137	**91**	0
4	src/contracts/protocol/libraries/configuration/EModeConfiguration.sol	83	5	30	**48**	21
5	src/contracts/protocol/libraries/configuration/ReserveConfiguration.sol	583	57	224	**302**	4
6	src/contracts/protocol/libraries/helpers/Errors.sol	103	1	6	**96**	0
7	src/contracts/protocol/libraries/logic/ConfiguratorLogic.sol	223	28	39	**156**	1
8	src/contracts/protocol/libraries/logic/EModeLogic.sol	70	6	18	**46**	13
9	src/contracts/protocol/libraries/logic/GenericLogic.sol	257	29	51	**177**	19
10	src/contracts/protocol/libraries/logic/LiquidationLogic.sol	460	47	90	**323**	8
11	src/contracts/protocol/libraries/logic/ValidationLogic.sol	642	65	133	**444**	16
12	src/contracts/protocol/libraries/types/DataTypes.sol	300	23	64	**213**	0
13	src/contracts/protocol/pool/PoolConfigurator.sol	621	91	69	**461**	8
	Total	**4438**	**474**	**1255**	**2709**	**8**

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 Liquid eModes update for Aave v3 enhances the flexibility and granularity of the existing eMode feature. Originally, eModes allowed users to group correlated assets with higher-risk configurations but limited assets to a single eMode. Liquid eModes now enable assets to belong to multiple eModes, providing more dynamic use cases. Users can still only activate one eMode at a time, but new features allow for more detailed configuration, such as enabling or disabling specific assets for collateral or borrowing within an eMode.

Key changes include the removal of the unused eMode oracle to save gas, and the introduction of bitmask configurations to specify which assets can be borrowed or used as collateral in each eMode. Assets must be borrowable or collateralizable both in and outside eMode. Additionally, the update ensures that health factors are validated during eMode switches and enhances the PoolConfigurator methods. Breaking changes include the removal of certain legacy methods and the introduction of new event types for tracking asset changes in eModes. Overall, these modifications improve flexibility and governance control over asset risk configurations.

2.6 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/contracts/protocol/pool/PoolConfigurator.sol	**4**	0	0	2	2
src/contracts/protocol/libraries/configuration/EModeConfiguration.sol	**1**	0	0	1	0
src/contracts/protocol/libraries/logic/GenericLogic.sol	**1**	0	0	0	1
src/contracts/protocol/pool/Pool.sol	**1**	0	0	0	1

2.7 CONCLUSION

A comprehensive audit was conducted on 13 smart contracts, initially revealing 3 warning issues, along with numerous informational notes. The audit identified issues related to the unintended inclusion of new reserves in eMode categories, missing asset validation in `PoolConfigurator`, and the possibility of modifying bitmaps for non-existent eMode categories, along with minor concerns like redundant castings and inconsistent error descriptions.

Following our initial audit, BGD Labs worked closely with our team to address the identified issues. The proposed changes focus on improving eMode category management, ensuring proper asset validation and configuration, and enhancing code clarity and efficiency. These recommendations aim to prevent unintended reserve inclusions, enforce accurate asset assignment in eModes, and provide clearer documentation. Through multiple rounds of interaction, all identified issues have been successfully addressed or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that the Aave V3 Liquid eModes, as of audited commit `740d0a8163197a4948c959078258c12254207bef`, 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.

3 FINDINGS REPORT

3.1 CRITICAL

No critical issues found.

3.2 MAJOR

No major issues found.

3.3 WARNING

W-01	Possibility to modify the bitmap for non-existent `emode` categories in `PoolConfigurator`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
PoolConfigurator.sol	contract `PoolConfigurator` > function `setAssetCollateralInEMode`	418
PoolConfigurator.sol	contract `PoolConfigurator` > function `setAssetBorrowableInEMode`	431

Description

In the mentioned locations, it is possible to add an `asset` to an `emode` category as "borrowable" or "collateral" by specifying the `categoryId`. While `categoryId` cannot be set to `0` due to a check inside the `configureEModeCategory` function, there are no checks to verify if the category with the specified `categoryId` actually exists.

This allows an `asset` to be added to a non-existent category, which could later become active.

Recommendation

We recommend considering this issue during the preparation of the eModes configuration update. Alternatively, validation can be added to not only check for non-zero values but also verify the existence of the `categoryId` to prevent populating the `_eModeCategories` mapping with non-existent categories, which could cause issues in the future.

Update

Client's response

According to the documentation:

> Note: The methods to alter configuration do not validate for an asset / eMode to exist. This is to stay consistent with the current methods on PoolConfigurator, as there are multiple layers of security/risk procedures on updates to not create any issues.

The reason why we don't enforce this is that usually these updates are called trough multiple layers of abstractions on top, so it's possible that the same transaction would add an asset:

- borrowable
- collateral
- creates the eMode

Enforcing existence would require us to enforce transaction order, which could create problems.
According to our assessment, setting a non-existent eMode or even a non-existent asset should never create any problem.

Users cannot enter an eMode that has `ltv == 0`, and users cannot supply/borrow non-existent assets.

W-02	A new reserve upon initialization may unintentionally be included in an `emode` category in `EModeConfiguration`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
EModeConfiguration.sol	contract `EModeConfiguration` > function `setCollateral`	20
EModeConfiguration.sol	contract `EModeConfiguration` > function `setBorrowable`	55

Description

In the mentioned locations, a bit is set where the bit index corresponds to the reserve id. This means that the reserve is included in the corresponding `emode` category. However, when a reserve is dropped, the corresponding bit is not cleared from the `emode` category bitmap.

Furthermore, when initializing a new reserve in the `executeInitReserve` function, it may be assigned the same id as that of the removed reserve:

```
for (uint16 i = 0; i < params.reservesCount; i++) {
  if (reservesList[i] == address(0)) {
    reservesData[params.asset].id = i;
    reservesList[i] = params.asset;
    return false;
  }
}
```

As a result, the new reserve will immediately be included in the `emode` category right after initialization.

Although the documentation explicitly states that `dropReserve` is unlikely to be ever called and does not account for the removal of flags from eModes, in this issue we outline the potential consequences of this decision.

Test case for the described issue:

```
// in the "PoolConfigurator.eMode.sol" file

import {ConfiguratorInputTypes} from '../../../../src/contracts/protocol/pool/
PoolConfigurator.sol';
import {TestVars} from '../../../utils/TestnetProcedures.sol';

// ...

function test_unchangedReserveIdInBitmap(TestVars memory t) public {
  // copy from test_setAssetCollateralInEMode()
  EModeCategoryInput memory input = _genCategoryOne();
  test_configureEmodeCategory();
  vm.expectEmit(address(contracts.poolConfiguratorProxy));
  emit AssetCollateralInEModeChanged(tokenList.usdx, input.id, true);
  vm.prank(poolAdmin);
  contracts.poolConfiguratorProxy.setAssetCollateralInEMode(tokenList.usdx, input.id, true);
  DataTypes.EModeCategory memory config = contracts.poolProxy.getEModeCategoryData(input.id);
  DataTypes.ReserveDataLegacy memory usdxReserveData = contracts.poolProxy.getReserveData(
    tokenList.usdx
  );
  assertEq(EModeConfiguration.isCollateralAsset(config.isCollateralBitmap,
usdxReserveData.id), true);

  // drop the USDX reserve
  vm.prank(poolAdmin);
  contracts.poolConfiguratorProxy.dropReserve(tokenList.usdx);

  // init the new reserve
  ConfiguratorInputTypes.InitReserveInput[] memory initInput = _generateInitConfig(
    t,
    report,
    poolAdmin,
    true
  );
  vm.prank(poolAdmin);
  contracts.poolConfiguratorProxy.initReserves(initInput);

  // check that the new reserve has the same id as dropped USDX and
  // is already in EMode right after initialization
  DataTypes.ReserveDataLegacy memory newReserveData = contracts.poolProxy.getReserveData(
    initInput[0].underlyingAsset
  );
  assertEq(usdxReserveData.id, newReserveData.id);
  assertEq(EModeConfiguration.isCollateralAsset(config.isCollateralBitmap,
newReserveData.id), true);
}
```

Recommendation

We recommend implementing the clearing of the corresponding bits from the `isCollateralBitmap` and `isBorrowableBitmap` in the eMode category configuration when removing an asset to avoid the unintended inclusion of a new asset in the eMode category.

Update

Client's response

According to the documentation:

> Note: dropReserve does not account for removal of flags from eModes.
> The rationale for this is the following:
> - even if a reserve is dropped and the flags persist, the system should behave perfectly fine given as for dropping the supply is enforced to be zero

This is expected behavior. Of course, when reusing that slot, one needs to act with caution.

W-03	Missing validation for the existence of `asset` in `PoolConfigurator`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
PoolConfigurator.sol	contract `PoolConfigurator` > function `setAssetCollateralInEMode`	425
PoolConfigurator.sol	contract `PoolConfigurator` > function `setAssetBorrowableInEMode`	438

Description

The functions `setAssetCollateralInEMode` and `setAssetBorrowableInEMode` do not validate whether the specified `asset` exists in the reserve. As a result, if a non-existent asset is passed, the system may inadvertently add an asset with `reserveData.index = 0` to the category in the bitmaps `isCollateralBitmap` or `isBorrowableBitmap`. On Ethereum Mainnet, this index corresponds to `WETH`, meaning `WETH` could unintentionally be added to an eMode category, potentially causing unexpected behavior or security issues.

In other functions of `PoolConfigurator`, asset existence is validated in the `_pool.setConfiguration` method, which was also previously present in the `setAssetEModeCategory` function but was removed as part of the audited codebase changes.

The events `AssetCollateralInEModeChanged` and `AssetBorrowableInEModeChanged` misleadingly suggest that the asset parameter was correctly assigned to the specified `categoryId`. This could result in unintended consequences if invalid assets are configured without proper validation.

Recommendation

We recommend implementing a validation check to ensure that the asset exists in the reserve before proceeding with the configuration changes. This can be achieved similarly to the `setConfiguration` function in the `Pool` contract:

```
require(_reserves[asset].id != 0 || _reservesList[0] == asset, Errors.ASSET_NOT_LISTED);
```

Update

Fixed in commit `27418752340b4b78729761aea72f29c09b9b265b`.

Client's response

In the docs we exactly state that that we don't validate this because there's system built around protecting against misusage. But we acknowledge that the validation in this case is an improvement without drawbacks.

3.4 INFO

I-01	Redundant casting to `uint8` in `PoolConfigurator`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
PoolConfigurator.sol	contract `PoolConfigurator` > function `setAssetCollateralInEMode`	427

Description

In the mentioned locations, the type `uint8` is assigned to variables that are already of type `uint8`, leading to reduced code readability and additional gas costs.

Recommendation

We recommend removing excessive casting of variables to the `uint8` type.

Update

Fixed in commit `477d9edd80adf50b13a1b476dcc543e4233ab9ad`.

I-02	Unused field `eModeAssetCategory` in `GenericLogic`
Severity	**INFO**
Status	● FIXED

Location

File	Location	Line
GenericLogic.sol	contract `GenericLogic` > struct `CalculateUserAccountDataVars`	43

Description

In the struct `CalculateUserAccountDataVars` of contract `GenericLogic`, the field `eModeAssetCategory` is not used anywhere in a current codebase.

Recommendation

We recommend removing the field `eModeAssetCategory` from the struct `CalculateUserAccountDataVars`.

Update

Fixed in commit `9a72720ad12ff38e3c957cbef3e1e1a05ffb2095`.

I-03	Emode category id cast to `uint256` in `Pool`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
Pool.sol	contract `Pool` > function `getUserEMode`	720

Description

In the `getUserEMode` function of the `Pool` contract, the return value for the emode category id is cast to type `uint256`:

```
function getUserEMode(address user) external view virtual override returns (uint256) {
    return _usersEModeCategory[user];
}
```

However, the category id is of type `uint8`.

Recommendation

We recommend changing the return type of the function to `uint8`, which corresponds to the type of the emode category id.

Update

Client's response

This can not be changed without causing issues on 3th party integrations.

I-04	Same error description for different errors within the `setEModeCategory` function in `PoolConfigurator`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
PoolConfigurator.sol	contract `PoolConfigurator` > function `setEModeCategory`	380
PoolConfigurator.sol	contract `PoolConfigurator` > function `setEModeCategory`	386
PoolConfigurator.sol	contract `PoolConfigurator` > function `setEModeCategory`	389
PoolConfigurator.sol	contract `PoolConfigurator` > function `setEModeCategory`	397

Description

In the mentioned locations, the same constant `Errors.INVALID_EMODE_CATEGORY_PARAMS` is used as a reference for describing different errors. This could mislead users, as the explanation will not be accurate.

Recommendation

We recommend using distinct error descriptions for different errors to simplify troubleshooting and help identify the specific cause of the issue.

Update

Client's response

We will not change this, as it's done in multiple places in the aave codebase and we don't consider the improvement big enough to propose the change.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

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

1. Project Architecture Review

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

2. Vulnerability Assessment

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

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

3. Security Model Evaluation

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

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

4. Cross-Verification by Multiple Auditors

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

5. Report Consolidation

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

6. Reaudit of Revised Submissions

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

7. Final Audit Report Publication

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

4.2 FINDINGS CLASSIFICATION REFERENCE

4.2.1 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.

4.2.2 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 and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.3 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

OXORIO