LTV protocol

LTV SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

NOVEMBER 4, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for LTV Protocol's LTV.

The LTV Protocol is a revolutionary Curatorless Leveraged Tokenized Vault that maintains a constant target Loan-To-Value (LTV) ratio without requiring a central curator. Built on the foundation of two interconnected EIP-4626 vaults, it enables users to deposit and withdraw funds while receiving tokenized shares representing their leveraged positions. The protocol's core innovation lies in its auction-based stimulus system that incentivizes users to participate in rebalancing actions through rewards or fees. This mechanism ensures alignment with the target LTV while providing basic MEV protection against frontrunning.

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 244 smart contracts, encompassing 9087 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with LTV Protocol 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 LTV Protocol to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by LTV Protocol, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the LTV Smart contracts, as of audited commit c8a080d1a79d03877ab8b1605ee061010afd5dba, have met the security and functionality requirements established for this audit, based on the code and documentation provided, and operate as intended within the defined scope.

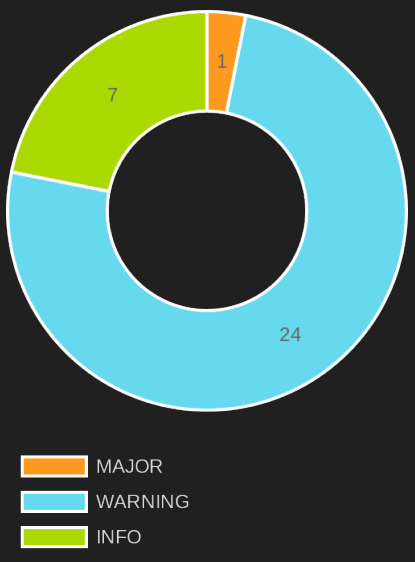
1.2 SUMMARY OF FINDINGS

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.

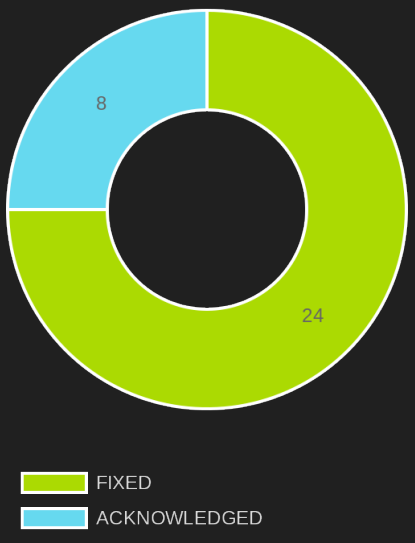
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

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

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**1**	0	1	0	0
WARNING	**24**	0	17	7	0
INFO	**7**	0	6	1	0
TOTAL	**32**	**0**	**24**	**8**	**0**



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 10

2.2. PROJECT BRIEF 11

2.3. PROJECT TIMELINE 12

2.4. AUDITED FILES 14

2.5. PROJECT OVERVIEW 21

2.6. CODEBASE QUALITY ASSESSMENT 22

2.7. FINDINGS BREAKDOWN BY FILE 24

2.8. CONCLUSION 26

3. AUDIT OBJECTIVES 28

3.1. ARCHITECTURE AND STORAGE SAFETY 30

Audit Hypotheses 30

Audit Strategies 30

Key Findings Validated 30

3.2. ACCESS CONTROL AND PERMISSIONS 32

Audit Hypotheses 32

Audit Strategies 32

Key Findings Validated 33

3.3. INTEGRATION WITH EXTERNAL PROTOCOLS 34

Audit Hypotheses 34

Audit Strategies 34

Key Findings Validated 35

3.4. TOKEN HANDLING AND APPROVALS 36
Audit Hypotheses 36
Audit Strategies 36
Key Findings Validated 37
3.5. DATA VALIDATION AND PARAMETER UPDATES 38
Audit Hypotheses 38
Audit Strategies 38
Key Findings Validated 39
3.6. REENTRANCY AND ATTACK VECTORS 40
Audit Hypotheses 40
Audit Strategies 40
Key Findings Validated 41
3.7. MATHEMATICAL CORRECTNESS 42
Audit Hypotheses 42
Audit Strategies 42
Key Findings Validated 43
3.8. EDGE CASES AND RISK SCENARIOS 44
Audit Hypotheses 44
Audit Strategies 44
Key Findings Validated 45
3.9. OUT OF SCOPE 46
4. FINDINGS REPORT 47
4.1. CRITICAL 48
4.2. MAJOR 49
M-01 Deposit may result in near-zero shares minted 49
4.3. WARNING 53
W-01 Division by Zero Risk in AdministrationSetters 53
W-02 Missing Zero Address Validation in Constructors in AaveV3Connector, MorphoConnector .. 54
W-03 Missing Validation for eMode Category Existence in AaveV3Connector 55

W-04 Missing Verification of Support for Borrow and Collateral Tokens in AaveV3OracleConnector, MorphoOracleConnector 56
W-05 Missing Data Validation in MorphoConnector 58
W-06 Non-Upgradeable Contract in MorphoConnector, AaveV3Connector, VaultBalanceAsLendingConnector,AaveV3OracleConnector, MorphoOracleConnector, ConstantSlippageConnector 60
W-07 Weak Address Validation in ModulesProvider 62
W-08 No Signature Expiry in WhitelistRegistry 63
W-09 Storage Incompatibility Risk in AdministrationWrite 65
W-10 Unsafe Delegatecall in InitializeWrite 66
W-11 Unclear Deleveraging State in OnlyEmergencyDeleverager 67
W-12 Unrestricted approve Function in Approve 68
W-13 Missing Zero Address Check in Approve 69
W-14 Missing Balance Checks in Transfer, TransferFrom, and Burn 70
W-15 Zero Address Receiver in Mint, MintCollateral, Redeem, RedeemCollateral, Withdraw, and WithdrawCollateral 72
W-16 Unsafe Delegatecall to Zero Address in AdministrationSetters 74
W-17 Unbounded LTV Parameters in AdministrationSetters 76
W-18 No Pause Mechanism in Initialize 78
W-19 Unfair fee collection on price increase in ApplyMaxGrowthFee 79
W-20 Readonly reentrancy during the execution of lowLevel functions in ExecuteLowLevelRebalance 81
W-21 Missing Protections Against Price Manipulation in Oracle Functions in AaveV3OracleConnector, MorphoOracleConnector 83
W-22 Slippage value validation in ConstantSlippageConnector 85
W-23 Inability to deposit when borrow is negative in Vault, VaultCollateral 86
W-24 Missing handling of attempts to liquidate more collateral than available in OnlyEmergencyDeleverager 88
4.4. INFO 90
I-01 Incorrect uint256 Type for eMode Parameter in AaveV3Connector 90
I-02 Unsafe Use of abi.encodePacked for Hashing in WhitelistRegistry 91

I-03 Missing Minimum Value Enforcement for Auction Duration in Initialize 92
I-04 No sweep of excessive tokens in LTV 93
I-05 Redundant parameter type in function _getBool in BoolReader 94
I-06 feeCollector cannot burn shares when withdrawals are disabled in AdministrationSetters ... 95
I-07 Misleading comments and typos in TotalAssets, TotalSupply, MaxGrowthFee, DeltaSharesAndDeltaRealBorrow, DeltaSharesAndDeltaRealCollateral 97
5. APPENDIX 99
5.1. SECURITY ASSESSMENT METHODOLOGY 100
5.2. CODEBASE QUALITY ASSESSMENT REFERENCE 102
Rating Criteria 103
5.3. FINDINGS CLASSIFICATION REFERENCE 104
Severity Level Reference 104
Status Level Reference 104
5.4. ABOUT OXORIO 106

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	LTV Protocol
Project name	LTV
Category	Leveraged Tokenized Vault
Website	https://ltv.finance
Repository	https://github.com/ltvprotocol/ltv_v0
Documentation	LTV_Curatorless_Leveraged_Tokenized_Vault_with_a_Constant_Target_Loan-To-Value_Ratio.pdf
Initial Commit	5b9a17e514e8dd4c8d6f4f85daa86900b2240115
Final Commit	c8a080d1a79d03877ab8b1605ee061010afd5dba
Platform	L1
Network	Ethereum
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Elena Kozmiryuk - elena@oxor.io

2.3 PROJECT TIMELINE

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

Date	Event
May 16, 2025	Client approached for consultation on project architecture.
May 21, 2025	Introductory call with tech lead.
May 23, 2025	Tech call with auditors to familiarize with code and architectural decisions, formalization of the task.
May 29, 2025	Delivered architecture review to client with recommendations on code base security and preparation for audit.
June 6, 2025	Tech call with auditors, explanation of architecture review, "good samaritan" case, Q&A.
June 18, 2025	Client requested audit.
July 9, 2025	Client revised the timeline for code preparation for audit and approved audit deadlines.
September 16, 2025	Client provided up-to-date project documentation before audit start.
September 22, 2025	Kick-off call to start according to the planned timeline.
September 22, 2025	Start of audit according to the planned timeline, kick-off call.
September 25, 2025	Tech call, final implementation notes, client answers auditors' questions.
September 29, 2025	Tech call, code freeze.
October 2, 2025	Tech call.
October 8, 2025	Submission of the preliminary audit report.
October 9, 2025	Tech call with client on preliminary report.
October 17, 2025	Submission of the initial audit report.
October 28, 2025	The audit team commenced work on a re-audit of the project.

Date	Event
November 4, 2025	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. 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/connectors/lending_connectors/AaveV3Connector.sol	81	13	29	**39**	0%
2	src/connectors/lending_connectors/interfaces/IAaveV3Pool.sol	37	7	1	**29**	0%
3	src/connectors/lending_connectors/interfaces/IMorphoBlue.sol	67	9	1	**57**	0%
4	src/connectors/lending_connectors/MorphoConnector.sol	132	27	39	**66**	9%
5	src/connectors/lending_connectors/VaultBalanceAsLendingConnector.sol	54	12	12	**30**	0%
6	src/connectors/oracle_connectors/AaveV3OracleConnector.sol	46	8	17	**21**	0%
7	src/connectors/oracle_connectors/interfaces/IAaveV3Oracle.sol	6	1	1	**4**	0%
8	src/connectors/oracle_connectors/interfaces/IMorphoOracle.sol	6	1	1	**4**	0%
9	src/connectors/oracle_connectors/MorphoOracleConnector.sol	45	8	16	**21**	0%
10	src/connectors/README.md	50	18	0	**32**	0%
11	src/connectors/slippage_connectors/ConstantSlippageConnector.sol	37	6	15	**16**	0%
12	src/constants/Constants.sol	22	3	5	**14**	0%
13	src/constants/README.md	5	2	0	**3**	0%
14	src/elements/LTV.sol	57	2	22	**33**	0%
15	src/elements/modules/AdministrationModule.sol	18	2	5	**11**	0%
16	src/elements/modules/AuctionModule.sol	15	2	5	**8**	0%
17	src/elements/modules/BorrowVaultModule.sol	19	2	5	**12**	0%
18	src/elements/modules/CollateralVaultModule.sol	26	2	5	**19**	0%
19	src/elements/modules/ERC20Module.sol	17	2	5	**10**	0%
20	src/elements/modules/InitializeModule.sol	14	2	5	**7**	0%
21	src/elements/modules/LowLevelRebalanceModule.sol	21	2	5	**14**	0%
22	src/elements/ModulesProvider.sol	106	14	36	**56**	0%
23	src/elements/README.md	44	13	0	**31**	0%
24	src/elements/WhitelistRegistry.sol	78	12	24	**42**	2%
25	src/errors/IAdministrationErrors.sol	218	26	148	**44**	0%
26	src/errors/IAuctionErrors.sol	56	4	39	**13**	0%
27	src/errors/IERC20Errors.sol	27	1	21	**5**	0%
28	src/errors/ILowLevelRebalanceErrors.sol	46	4	35	**7**	0%
29	src/errors/IVaultErrors.sol	115	12	83	**20**	0%
30	src/errors/README.md	100	27	0	**73**	0%
31	src/events/IAdministrationEvents.sol	185	19	138	**28**	0%
32	src/events/IAuctionEvent.sol	22	1	17	**4**	0%
33	src/events/IERC20Events.sol	31	2	24	**5**	0%
34	src/events/IERC4626Events.sol	59	1	43	**15**	0%
35	src/events/ILowLevelRebalanceEvent.sol	25	1	18	**6**	0%
36	src/events/README.md	105	23	0	**82**	0%

	File	Lines	Blanks	Comments	Code	Complexity
37	src/facades/README.md	28	7	0	**21**	0%
38	src/facades/reads/AdministrationRead.sol	47	6	21	**20**	0%
39	src/facades/reads/AuctionRead.sol	25	3	12	**10**	0%
40	src/facades/reads/BorrowVaultRead.sol	109	14	43	**52**	0%
41	src/facades/reads/CollateralVaultRead.sol	105	14	42	**49**	0%
42	src/facades/reads/ERC20Read.sol	20	4	9	**7**	0%
43	src/facades/reads/LowLevelRebalanceRead.sol	98	11	30	**57**	0%
44	src/facades/writes/AdministrationWrite.sol	188	25	74	**89**	1%
45	src/facades/writes/AuctionWrite.sol	27	3	13	**11**	0%
46	src/facades/writes/BorrowVaultWrite.sol	41	5	19	**17**	0%
47	src/facades/writes/CollateralVaultWrite.sol	41	5	19	**17**	0%
48	src/facades/writes/CommonWrite.sol	34	5	11	**18**	39%
49	src/facades/writes/ERC20Write.sol	34	4	16	**14**	0%
50	src/facades/writes/InitializeWrite.sol	29	3	12	**14**	0%
51	src/facades/writes/LowLevelRebalanceWrite.sol	54	6	22	**26**	0%
52	src/interfaces/connectors/ILendingConnector.sol	37	1	26	**10**	0%
53	src/interfaces/connectors/IOracleConnector.sol	23	3	14	**6**	0%
54	src/interfaces/connectors/ISlippageConnector.sol	22	2	14	**6**	0%
55	src/interfaces/ILTV.sol	1019	145	688	**186**	0%
56	src/interfaces/IModules.sol	45	2	26	**17**	0%
57	src/interfaces/IWhitelistRegistry.sol	13	1	8	**4**	0%
58	src/interfaces/README.md	43	12	0	**31**	0%
59	src/interfaces/reads/IAdministrationModule.sol	35	6	20	**9**	0%
60	src/interfaces/reads/IAuctionModule.sol	30	3	15	**12**	0%
61	src/interfaces/reads/IBorrowVaultModule.sol	108	14	42	**52**	0%
62	src/interfaces/reads/ICollateralVaultModule.sol	112	14	42	**56**	0%
63	src/interfaces/reads/IERC20Module.sol	15	1	10	**4**	0%
64	src/interfaces/reads/ILowLevelRebalanceModule.sol	82	9	30	**43**	0%
65	src/interfaces/writes/IInitializeModule.sol	15	2	8	**5**	0%
66	src/math/abstracts/AuctionStateToData.sol	28	2	8	**18**	0%
67	src/math/abstracts/BoolReader.sol	47	6	22	**19**	5%
68	src/math/abstracts/MaxGrowthFee.sol	69	6	18	**45**	4%
69	src/math/abstracts/state_to_data/max/MaxDepositMintCollateralStateToData.sol	43	2	9	**32**	0%
70	src/math/abstracts/state_to_data/max/MaxDepositMintStateToData.sol	48	5	9	**34**	0%
71	src/math/abstracts/state_to_data/max/MaxWithdrawRedeemCollateralStateToData.sol	44	2	9	**33**	0%
72	src/math/abstracts/state_to_data/max/MaxWithdrawRedeemStateToData.sol	47	4	9	**34**	0%
73	src/math/abstracts/state_to_data/MaxGrowthFeeStateToConvertCollateralData.sol	62	6	9	**47**	2%
74	src/math/abstracts/state_to_data/preview/PreviewDepositStateToPreviewDepositData.sol	135	12	17	**106**	1%
75	src/math/abstracts/state_to_data/preview/PreviewDepositVaultStateToCollateralData.sol	148	13	17	**118**	1%

	File	Lines	Blanks	Comments	Code	Complexity
76	src/math/abstracts/state_to_data/preview/PreviewLowLevelRebalanceStateToData.sol	122	11	11	**100**	1%
77	src/math/abstracts/state_to_data/preview/PreviewWithdrawStateToPreviewWithdrawData.sol	125	12	16	**97**	1%
78	src/math/abstracts/state_to_data/preview/PreviewWithdrawVaultStateToCollateralData.sol	137	12	16	**109**	1%
79	src/math/abstracts/state_to_data/TotalAssetsCollateralStateToData.sol	80	5	8	**67**	0%
80	src/math/abstracts/state_to_data/TotalAssetsStateToData.sol	66	4	8	**54**	0%
81	src/math/abstracts/Vault.sol	44	6	13	**25**	8%
82	src/math/abstracts/VaultCollateral.sol	52	6	13	**33**	6%
83	src/math/libraries/AuctionMath.sol	459	43	132	**284**	11%
84	src/math/libraries/CasesOperator.sol	23	2	5	**16**	44%
85	src/math/libraries/CommonBorrowCollateral.sol	248	29	99	**120**	12%
86	src/math/libraries/CommonMath.sol	158	14	61	**83**	4%
87	src/math/libraries/delta_future_borrow/DeltaSharesAndDeltaRealBorrow.sol	588	69	180	**339**	13%
88	src/math/libraries/delta_future_collateral/DeltaRealBorrowAndDeltaRealCollateral.sol	674	94	210	**370**	12%
89	src/math/libraries/delta_future_collateral/DeltaSharesAndDeltaRealCollateral.sol	593	71	181	**341**	13%
90	src/math/libraries/DepositWithdraw.sol	91	13	11	**67**	3%
91	src/math/libraries/LowLevelRebalanceMath.sol	263	23	74	**166**	4%
92	src/math/libraries/MintRedeem.sol	120	16	13	**91**	3%
93	src/math/libraries/MulDiv.sol	126	25	21	**80**	32%
94	src/math/libraries/NextStep.sol	156	11	43	**102**	15%
95	src/math/README.md	98	33	0	**65**	0%
96	src/modifiers/AdministrationModifiers.sol	60	9	25	**26**	23%
97	src/modifiers/FunctionStopperModifier.sol	28	3	12	**13**	0%
98	src/modifiers/README.md	7	2	0	**5**	0%
99	src/modifiers/WhitelistModifier.sol	31	3	12	**16**	6%
100	src/public/administration/read/GetLendingConnector.sol	20	2	10	**8**	0%
101	src/public/administration/read/GetVaultBoolState.sol	38	5	17	**16**	0%
102	src/public/administration/write/OnlyEmergencyDeleverager.sol	99	17	11	**71**	11%
103	src/public/administration/write/OnlyGovernor.sol	106	11	35	**60**	0%
104	src/public/administration/write/OnlyGuardian.sol	34	4	14	**16**	0%
105	src/public/administration/write/OnlyOwner.sol	68	7	23	**38**	0%
106	src/public/auction/read/PreviewExecuteAuctionBorrow.sol	37	3	11	**23**	0%
107	src/public/auction/read/PreviewExecuteAuctionCollateral.sol	37	3	11	**23**	0%
108	src/public/auction/write/ExecuteAuctionBorrow.sol	29	3	8	**18**	0%
109	src/public/auction/write/ExecuteAuctionCollateral.sol	33	3	8	**22**	0%
110	src/public/erc20/read/TotalSupply.sol	20	3	10	**7**	0%
111	src/public/erc20/write/Approve.sol	35	3	10	**22**	0%
112	src/public/erc20/write/Transfer.sol	44	3	9	**32**	6%
113	src/public/erc20/write/TransferFrom.sol	47	4	10	**33**	6%
114	src/public/low_level/read/max/MaxLowLevelRebalanceBorrow.sol	42	4	11	**27**	4%

	File	Lines	Blanks	Comments	Code	Complexity
115	src/public/low_level/read/max/MaxLowLevelRebalanceCollateral.sol	46	4	15	**27**	4%
116	src/public/low_level/read/max/MaxLowLevelRebalanceShares.sol	121	10	13	**98**	1%
117	src/public/low_level/read/preview/PreviewLowLevelRebalanceBorrow.sol	71	7	17	**47**	6%
118	src/public/low_level/read/preview/PreviewLowLevelRebalanceCollateral.sol	73	7	17	**49**	6%
119	src/public/low_level/read/preview/PreviewLowLevelRebalanceShares.sol	41	4	11	**26**	4%
120	src/public/low_level/write/execute/ExecuteLowLevelRebalanceBorrow.sol	97	11	17	**69**	3%
121	src/public/low_level/write/execute/ExecuteLowLevelRebalanceCollateral.sol	101	10	17	**74**	3%
122	src/public/low_level/write/execute/ExecuteLowLevelRebalanceShares.sol	103	10	10	**83**	2%
123	src/public/README.md	33	12	0	**21**	0%
124	src/public/vault/read/borrow/convert/ConvertToAssets.sol	26	4	8	**14**	7%
125	src/public/vault/read/borrow/convert/ConvertToShares.sol	29	4	11	**14**	7%
126	src/public/vault/read/borrow/max/MaxDeposit.sol	53	7	13	**33**	6%
127	src/public/vault/read/borrow/max/MaxMint.sol	54	7	13	**34**	6%
128	src/public/vault/read/borrow/max/MaxRedeem.sol	65	11	13	**41**	12%
129	src/public/vault/read/borrow/max/MaxWithdraw.sol	106	10	49	**47**	11%
130	src/public/vault/read/borrow/preview/PreviewDeposit.sol	91	9	21	**61**	3%
131	src/public/vault/read/borrow/preview/PreviewMint.sol	83	9	20	**54**	4%
132	src/public/vault/read/borrow/preview/PreviewRedeem.sol	87	9	20	**58**	3%
133	src/public/vault/read/borrow/preview/PreviewWithdraw.sol	89	8	20	**61**	3%
134	src/public/vault/read/borrow/TotalAssets.sol	42	5	17	**20**	5%
135	src/public/vault/read/collateral/convert/ConvertToAssetsCollateral.sol	37	4	11	**22**	5%
136	src/public/vault/read/collateral/convert/ConvertToSharesCollateral.sol	40	4	11	**25**	12%
137	src/public/vault/read/collateral/max/MaxDepositCollateral.sol	58	8	13	**37**	8%
138	src/public/vault/read/collateral/max/MaxMintCollateral.sol	57	8	13	**36**	8%
139	src/public/vault/read/collateral/max/MaxRedeemCollateral.sol	67	12	13	**42**	12%
140	src/public/vault/read/collateral/max/MaxWithdrawCollateral.sol	108	11	48	**49**	10%
141	src/public/vault/read/collateral/preview/PreviewDepositCollateral.sol	91	9	21	**61**	3%
142	src/public/vault/read/collateral/preview/PreviewMintCollateral.sol	87	9	20	**58**	3%
143	src/public/vault/read/collateral/preview/PreviewRedeemCollateral.sol	88	9	21	**58**	3%
144	src/public/vault/read/collateral/preview/PreviewWithdrawCollateral.sol	91	9	21	**61**	3%
145	src/public/vault/read/collateral/TotalAssetsCollateral.sol	48	5	15	**28**	4%
146	src/public/vault/write/borrow/execute/Deposit.sol	121	15	10	**96**	4%
147	src/public/vault/write/borrow/execute/Mint.sol	119	15	9	**95**	4%
148	src/public/vault/write/borrow/execute/Redeem.sol	129	16	9	**104**	5%
149	src/public/vault/write/borrow/execute/Withdraw.sol	128	15	9	**104**	5%
150	src/public/vault/write/collateral/execute/DepositCollateral.sol	126	14	9	**103**	4%
151	src/public/vault/write/collateral/execute/MintCollateral.sol	126	14	9	**103**	4%
152	src/public/vault/write/collateral/execute/RedeemCollateral.sol	130	15	9	**106**	5%

	File	Lines	Blanks	Comments	Code	Complexity
153	src/public/vault/write/collateral/execute/WithdrawCollateral.sol	130	15	9	**106**	5%
154	src/README.md	21	1	0	**20**	0%
155	src/state_reader/administration/GetLendingConnectorStateReader.sol	22	2	8	**12**	0%
156	src/state_reader/common/GetAuctionStateReader.sol	26	2	9	**15**	0%
157	src/state_reader/common/GetRealBorrowAssetsReader.sol	23	2	9	**12**	8%
158	src/state_reader/common/GetRealCollateralAndRealBorrowAssetsReader.sol	34	2	10	**22**	5%
159	src/state_reader/common/GetRealCollateralAssetsReader.sol	24	2	9	**13**	8%
160	src/state_reader/common/MaxGrowthFeeStateReader.sol	29	2	9	**18**	0%
161	src/state_reader/low_level/ExecuteLowLevelRebalanceStateReader.sol	22	2	9	**11**	0%
162	src/state_reader/low_level/MaxLowLevelRebalanceBorrowStateReader.sol	28	2	10	**16**	0%
163	src/state_reader/low_level/MaxLowLevelRebalanceCollateralStateReader.sol	32	2	10	**20**	0%
164	src/state_reader/low_level/MaxLowLevelRebalanceSharesStateReader.sol	28	2	10	**16**	0%
165	src/state_reader/low_level/PreviewLowLevelRebalanceStateReader.sol	32	2	9	**21**	0%
166	src/state_reader/README.md	3	1	0	**2**	0%
167	src/state_reader/vault/MaxDepositMintBorrowVaultStateReader.sol	24	2	9	**13**	0%
168	src/state_reader/vault/MaxDepositMintCollateralVaultStateReader.sol	24	2	9	**13**	0%
169	src/state_reader/vault/MaxWithdrawRedeemBorrowVaultStateReader.sol	28	2	9	**17**	0%
170	src/state_reader/vault/MaxWithdrawRedeemCollateralVaultStateReader.sol	29	2	9	**18**	0%
171	src/state_reader/vault/PreviewDepositVaultStateReader.sol	33	2	9	**22**	0%
172	src/state_reader/vault/PreviewWithdrawVaultStateReader.sol	29	2	9	**18**	0%
173	src/state_reader/vault/TotalAssetsStateReader.sol	36	2	10	**24**	0%
174	src/state_transition/AdministrationSetters.sol	279	29	79	**171**	8%
175	src/state_transition/ApplyMaxGrowthFee.sol	28	3	9	**16**	12%
176	src/state_transition/AuctionApplyDeltaState.sol	96	12	27	**57**	16%
177	src/state_transition/BoolWriter.sol	28	3	14	**11**	18%
178	src/state_transition/ERC20.sol	79	8	27	**44**	7%
179	src/state_transition/ExecuteLowLevelRebalance.sol	100	11	34	**55**	15%
180	src/state_transition/Initialize.sol	62	14	11	**37**	0%
181	src/state_transition/Lending.sol	55	6	18	**31**	0%
182	src/state_transition/MintProtocolRewards.sol	32	4	10	**18**	17%
183	src/state_transition/README.md	3	1	0	**2**	0%
184	src/state_transition/TransferFromProtocol.sol	24	4	7	**13**	8%
185	src/state_transition/VaultStateTransition.sol	59	7	12	**40**	30%
186	src/states/LTVState.sol	72	13	6	**53**	0%
187	src/states/README.md	3	1	0	**2**	0%
188	src/structs/connectors/MorphoConnectorStorage.sol	13	1	6	**6**	0%
189	src/structs/data/auction/AuctionData.sol	15	1	5	**9**	0%

	File	Lines	Blanks	Comments	Code	Complexity
190	src/structs/data/common/MaxGrowthFeeData.sol	14	1	5	**8**	0%
191	src/structs/data/low_level/DeltaRealCollateralFromDeltaSharesData.sol	18	1	5	**12**	0%
192	src/structs/data/low_level/LowLevelRebalanceData.sol	26	1	5	**20**	0%
193	src/structs/data/low_level/MaxLowLevelRebalanceSharesData.sol	16	1	5	**10**	0%
194	src/structs/data/vault/common/Cases.sol	17	1	5	**11**	0%
195	src/structs/data/vault/common/DepositWithdrawData.sol	23	1	5	**17**	0%
196	src/structs/data/vault/common/MergeAuctionData.sol	19	1	5	**13**	0%
197	src/structs/data/vault/common/MintProtocolRewardsData.sol	15	1	5	**9**	0%
198	src/structs/data/vault/common/MintRedeemData.sol	23	1	5	**17**	0%
199	src/structs/data/vault/convert/ConvertCollateralData.sol	11	1	5	**5**	0%
200	src/structs/data/vault/delta_real_borrow/DeltaSharesAndDeltaRealBorrowData.sol	23	2	5	**16**	0%
201	src/structs/data/vault/delta_real_borrow/DeltaSharesAndDeltaRealBorrowDividendData.sol	23	2	5	**16**	0%
202	src/structs/data/vault/delta_real_borrow/DeltaSharesAndDeltaRealBorrowDividerData.sol	19	2	5	**12**	0%
203	src/structs/data/vault/delta_real_collateral/DeltaRealBorrowAndDeltaRealCollateralData.sol	26	2	5	**19**	0%
204	src/structs/data/vault/delta_real_collateral/DeltaRealBorrowAndDeltaRealCollateralDividendData.sol	26	2	5	**19**	0%
205	src/structs/data/vault/delta_real_collateral/DeltaRealBorrowAndDeltaRealCollateralDividerData.sol	23	2	5	**16**	0%
206	src/structs/data/vault/delta_real_collateral/DeltaSharesAndDeltaRealCollateralData.sol	23	2	5	**16**	0%
207	src/structs/data/vault/delta_real_collateral/DeltaSharesAndDeltaRealCollateralDividendData.sol	23	2	5	**16**	0%
208	src/structs/data/vault/delta_real_collateral/DeltaSharesAndDeltaRealCollateralDividerData.sol	19	2	5	**12**	0%
209	src/structs/data/vault/max/MaxDepositMintBorrowVaultData.sol	17	2	5	**10**	0%
210	src/structs/data/vault/max/MaxDepositMintCollateralVaultData.sol	17	2	5	**10**	0%
211	src/structs/data/vault/max/MaxWithdrawRedeemBorrowVaultData.sol	17	2	5	**10**	0%
212	src/structs/data/vault/max/MaxWithdrawRedeemCollateralVaultData.sol	17	2	5	**10**	0%
213	src/structs/data/vault/preview/PreviewCollateralVaultData.sol	26	1	5	**20**	0%
214	src/structs/data/vault/preview/PreviewDepositBorrowVaultData.sol	26	1	5	**20**	0%
215	src/structs/data/vault/preview/PreviewWithdrawBorrowVaultData.sol	25	1	5	**19**	0%
216	src/structs/data/vault/total_assets/TotalAssetsCollateralData.sol	14	1	5	**8**	0%
217	src/structs/data/vault/total_assets/TotalAssetsData.sol	13	1	5	**7**	0%
218	src/structs/README.md	3	1	0	**2**	0%
219	src/structs/state/auction/AuctionState.sol	15	1	5	**9**	0%
220	src/structs/state/common/LendingConnectorState.sol	12	1	5	**6**	0%
221	src/structs/state/common/MaxGrowthFeeState.sol	18	2	5	**11**	0%
222	src/structs/state/common/ModulesState.sol	24	2	5	**17**	0%
223	src/structs/state/initialize/StateInitData.sol	42	2	5	**35**	0%
224	src/structs/state/low_level/execute/ExecuteLowLevelRebalanceState.sol	13	2	5	**6**	0%

	File	Lines	Blanks	Comments	Code	Complexity
225	src/structs/state/low_level/max/MaxLowLevelRebalanceBorrowStateData.sol	15	1	5	**9**	0%
226	src/structs/state/low_level/max/MaxLowLevelRebalanceCollateralStateData.sol	15	1	5	**9**	0%
227	src/structs/state/low_level/max/MaxLowLevelRebalanceSharesState.sol	16	2	5	**9**	0%
228	src/structs/state/low_level/preview/PreviewLowLevelRebalanceState.sol	19	2	5	**12**	0%
229	src/structs/state/vault/max/MaxDepositMintBorrowVaultState.sol	15	2	5	**8**	0%
230	src/structs/state/vault/max/MaxDepositMintCollateralVaultState.sol	15	2	5	**8**	0%
231	src/structs/state/vault/max/MaxWithdrawRedeemBorrowVaultState.sol	15	2	5	**8**	0%
232	src/structs/state/vault/max/MaxWithdrawRedeemCollateralVaultState.sol	15	2	5	**8**	0%
233	src/structs/state/vault/preview/PreviewDepositVaultState.sol	21	2	5	**14**	0%
234	src/structs/state/vault/preview/PreviewVaultState.sol	18	2	5	**11**	0%
235	src/structs/state/vault/preview/PreviewWithdrawVaultState.sol	19	2	5	**12**	0%
236	src/structs/state/vault/total_assets/CommonTotalAssetsState.sol	18	1	6	**11**	0%
237	src/structs/state/vault/total_assets/TotalAssetsState.sol	14	2	5	**7**	0%
238	src/structs/state_transition/DeltaAuctionState.sol	17	1	5	**11**	0%
239	src/structs/state_transition/DeltaFuture.sol	17	1	5	**11**	0%
240	src/structs/state_transition/NextState.sol	16	1	6	**9**	0%
241	src/structs/state_transition/NextStateData.sol	16	2	5	**9**	0%
242	src/structs/state_transition/NextStepData.sol	23	1	5	**17**	0%
243	src/utils/README.md	3	1	0	**2**	0%
244	src/utils/RevertWithDataIfNeeded.sol	19	1	8	**10**	20%
	Total	**15859**	**1874**	**4898**	**9087**	**5%**

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 LTV Protocol is a revolutionary Curatorless Leveraged Tokenized Vault that maintains a constant target Loan-To-Value (LTV) ratio without requiring a central curator. Built on the foundation of two interconnected EIP-4626 vaults, it enables users to deposit and withdraw funds while receiving tokenized shares representing their leveraged positions. The protocol's core innovation lies in its auction-based stimulus system that incentivizes users to participate in rebalancing actions through rewards or fees. This mechanism ensures alignment with the target LTV while providing basic MEV protection against frontrunning.

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	Comprehensive role-based controls with whitelisting and modifiers	**Excellent**
Arithmetic	Division by zero risks in fee calculations and unbounded negative borrow handling identified [W-05]; requires safe casting and bounds checks. The issue was resolved.	**Excellent**
Complexity	Moderate complexity with delegatecalls and state transitions.	**Good**
Data Validation	Missing min/max validations in slippage connectors and leverage setters [W-04, W-05]; add protocol-specific bounds to prevent misconfigurations. All issues were resolved.	**Excellent**
Decentralization	Curatorless design with limited admin roles enhances decentralization; emergency functions minimized.	**Excellent**
Documentation	Comprehensive inline comments, interfaces, and READMEs provide full coverage of the codebase; however, the dense mathematical derivations and modular architecture (e.g., Diamond/Facade patterns) make it heavy to understand, requiring significant time to fully grasp the protocol's intricacies and interactions.	**Good**
External Dependencies	There are integrations with Morpho and Aave. They are implemented in accordance with the requirements, and all validations are in place.	**Excellent**
Error Handling	Custom errors throughout with descriptive reverts; generic cases addressed.	**Excellent**
Logging and Monitoring	Granular events for auctions, rebalances, and fees enable full monitoring.	**Excellent**

Category	Assessment	Result
Low-Level Calls	Delegatecalls in facade/Diamond pattern for modular upgrades, with validated storage slots; however, unsafe delegatecalls to zero addresses in ModulesProvider and LowLevelRebalanceModule risk execution failures or unintended behavior due to uninitialized slots. All issues were resolved.	**Excellent**
Testing and Verification	Extensive Foundry testing with a large number of tests covering core functionalities and edge cases.	**Excellent**

2.7 FINDINGS BREAKDOWN BY FILE

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

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
src/connectors/lending_connectors/AaveV3Connector.sol	**4**	0	0	3	1
src/state_transition/AdministrationSetters.sol	**4**	0	0	3	1
src/connectors/lending_connectors/MorphoConnector.sol	**3**	0	0	3	0
src/connectors/oracle_connectors/AaveV3OracleConnector.sol	**3**	0	0	3	0
src/connectors/oracle_connectors/MorphoOracleConnector.sol	**3**	0	0	3	0
src/connectors/slippage_connectors/ConstantSlippageConnector.sol	**2**	0	0	2	0
src/elements/WhitelistRegistry.sol	**2**	0	0	1	1
src/public/administration/write/OnlyEmergencyDeleverager.sol	**2**	0	0	2	0
src/public/erc20/write/Approve.sol	**2**	0	0	2	0
src/state_transition/Initialize.sol	**2**	0	0	1	1
src/connectors/lending_connectors/VaultBalanceAsLendingConnector.sol	**1**	0	0	1	0
src/elements/LTV.sol	**1**	0	0	0	1
src/elements/ModulesProvider.sol	**1**	0	0	1	0
src/facades/writes/AdministrationWrite.sol	**1**	0	0	1	0
src/facades/writes/InitializeWrite.sol	**1**	0	0	1	0
src/math/abstracts/BoolReader.sol	**1**	0	0	0	1
src/math/abstracts/MaxGrowthFee.sol	**1**	0	0	0	1
src/math/abstracts/Vault.sol	**1**	0	0	1	0
src/math/abstracts/VaultCollateral.sol	**1**	0	0	1	0
src/math/libraries/delta_future_borrow/DeltaSharesAndDeltaRealBorrow.sol	**1**	0	0	0	1
src/math/libraries/delta_future_collateral/DeltaSharesAndDeltaRealCollateral.sol	**1**	0	0	0	1
src/public/erc20/read/TotalSupply.sol	**1**	0	0	0	1
src/public/erc20/write/Transfer.sol	**1**	0	0	1	0
src/public/erc20/write/TransferFrom.sol	**1**	0	0	1	0
src/public/vault/read/borrow/TotalAssets.sol	**1**	0	0	0	1
src/public/vault/write/borrow/execute/Mint.sol	**1**	0	0	1	0
src/public/vault/write/borrow/execute/Redeem.sol	**1**	0	0	1	0
src/public/vault/write/borrow/execute/Withdraw.sol	**1**	0	0	1	0
src/public/vault/write/collateral/execute/MintCollateral.sol	**1**	0	0	1	0
src/public/vault/write/collateral/execute/RedeemCollateral.sol	**1**	0	0	1	0
src/public/vault/write/collateral/execute/WithdrawCollateral.sol	**1**	0	0	1	0
src/state_transition/ApplyMaxGrowthFee.sol	**1**	0	0	1	0
src/state_transition/ERC20.sol	**1**	0	0	1	0

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
src/state_transition/ExecuteLowLevelRebalance.sol	**1**	0	0	1	0

2.8 CONCLUSION

A comprehensive audit was conducted on the LTV Protocol smart contracts, revealing **1 major issue** and **23 warning-level findings**, along with **7 informational notes**. The audit highlighted various potential vulnerabilities, with significant findings related to frontrunning attacks, validation gaps in connector initialization, oracle price manipulation risks, unsafe delegatecall patterns, fee mechanism fairness, and readonly reentrancy vectors.

Key Findings Summary:

The most major discovery was a frontrunning vulnerability in the deposit mechanism (`M-1`), where malicious actors could exploit third-party lending protocol features to force users into near-zero share mints, effectively draining deposited assets. This was addressed by the client through the implementation of helper contracts (`SafeERC4626` and `SafeERC4626Collateral`) to wrap vault interactions securely.

Additional high-priority issues included:

- Missing validation in connector initialization functions, allowing deployment with invalid or incompatible configurations
- Lack of safeguards against oracle price manipulation in `AaveV3OracleConnector` and `MorphoOracleConnector`
- Unsafe delegatecall operations that could mask failures when called on zero addresses
- Potential readonly reentrancy during `executeLowLevelRebalance` operations
- Unfair fee distribution mechanics in the `MaxGrowthFee` implementation

Recommendations Summary:

The proposed changes are aimed at:

1. Strengthening input validation across all connector initialization functions to prevent misconfiguration and ensure compatibility with underlying protocols (Aave, Morpho)
2. Implementing oracle resilience mechanisms, such as TWAP checks and price deviation bounds, to mitigate manipulation risks (acknowledged by client with plans for future connector versions)
3. Enhancing delegatecall safety through explicit zero-address checks and standardized error handling
4. Adding pausability mechanisms to enable safe protocol updates and emergency responses
5. Improving ERC20 operation safety with comprehensive zero-address checks and explicit balance validations
6. Refactoring fee calculation logic to prevent sudden dilution and ensure fair distribution across all users

7. Correcting typographical errors and misleading comments to maintain codebase clarity

These recommendations are based on adherence to industry best practices for DeFi protocols, ensuring enhanced security, operational reliability, and user protection. We strongly advise addressing the identified issues - particularly the acknowledged oracle manipulation risks and fee fairness concerns - to mitigate potential economic exploits, improve code quality, and ensure the contracts meet the highest security standards.

Additional Recommendations:

Moreover, we advise increasing the test coverage of the codebase, particularly for edge cases involving:

- Negative borrow/collateral scenarios and their impact on deposit/withdrawal operations
- Extreme LTV configurations and their interaction with lending protocol limits
- Oracle failure modes and fallback behaviors
- Reentrancy scenarios across different execution paths

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 LTV Vault Craft, as of audited commit [c8a080d1a79d03877ab8b1605ee061010afd5dba], 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 AUDIT OBJECTIVES

The audit focused on identifying vulnerabilities, ensuring compliance with ERC-4626 and EIP-2535 Diamond standards, and verifying the correctness of the implementation for the LTV Protocol's curatorless leveraged tokenized vault. The examination employed a multi-layered approach combining static analysis, dynamic testing, mathematical verification, and attack simulation to assess the protocol's resilience under both normal and adversarial conditions.

3.1 ARCHITECTURE AND STORAGE SAFETY

Objective: Ensure the integrity of the Facade architecture and Diamond pattern implementation to prevent storage collisions and state corruption across modular components.

3.1.1 Audit Hypotheses

- Storage slot collisions could occur between modules due to improper namespace isolation or overlapping slot assignments in the Diamond storage pattern
- Delegatecall operations in `ModulesProvider` might inadvertently corrupt caller contract state if module implementations use incompatible storage layouts
- Upgrades to lending/oracle/slippage connectors could introduce breaking changes to storage structure, causing data misinterpretation in downstream operations

3.1.2 Audit Strategies

- **Storage Layout Mapping:** Systematically documented all storage slots used across modules (`BorrowVaultModule`, `CollateralVaultModule`, `AuctionModule`, etc.) to identify potential overlaps or namespace violations
- **Delegatecall Flow Analysis:** Traced execution paths through `_delegate()` in facades to verify that context preservation maintains correct storage access, particularly for stateful operations like `executeLowLevelRebalance`
- **Upgrade Simulation Testing:** Created test scenarios mimicking connector upgrades (e.g., Morpho → Aave migration) to detect storage misalignment or orphaned state variables
- **Diamond Pattern Compliance:** Verified adherence to EIP-2535 facet management rules, ensuring module addresses are validated before registration and that selector conflicts are prevented

3.1.3 Key Findings Validated

- Confirmed that unsafe delegatecalls to zero addresses in `AdministrationSetters` (`_setLendingConnector`, `_setOracleConnector`) could mask initialization failures (`WARNING` severity)

- Identified storage incompatibility risks in `setModules()` where new module providers could be set without layout validation (`WARNING` severity)

3.2 ACCESS CONTROL AND PERMISSIONS

Objective: Validate the robustness of role-based access controls and ensure that privilege escalation or unauthorized state modifications are impossible.

3.2.1 Audit Hypotheses

- Privilege boundaries between `owner`, `governor`, `guardian`, and `emergencyDeleverager` roles might overlap, allowing lower-privileged roles to execute critical functions
- Whitelist enforcement could be bypassed through indirect paths (e.g., `transferFrom` with pre-approved allowances, or direct lending protocol interactions)
- Emergency functions like `deleverageAndWithdraw` might be callable multiple times without proper state locking, enabling repeated draining of vault assets

3.2.2 Audit Strategies

Role Permission Matrix: Constructed a comprehensive matrix mapping each role to authorized functions, then attempted cross-role function calls in test environments to detect permission leaks

Modifier Chain Analysis: Examined all access control modifiers (`onlyOwner`, `onlyGovernor`, `onlyGuardian`, `onlyEmergencyDeleverager`) for logical flaws or missing checks

Whitelist Bypass Testing: Attempted to circumvent `isReceiverWhitelisted` checks through:

- Pre-approved `transferFrom` calls from whitelisted → non-whitelisted addresses
- Direct `approve` calls followed by third-party transfers
- Exploit scenarios where `msg.sender` whitelist status is checked but not `recipient` in functions like `approve` (`WARNING` severity confirmed)

State Lock Verification: Tested emergency deleveraging to confirm `IS_VAULT_DELEVERAGED_BIT` prevents re-execution and validated whether the bit could be unset (unclear logic identified as `WARNING`)

3.2.3 Key Findings Validated

- `approve()` in `Approve.sol` lacked whitelist validation for `msg.sender`, allowing non-whitelisted users to grant spending rights when whitelisting is active (ACKNOWLEDGED by client as intended behavior)
- `deleverageAndWithdraw` did not clearly document whether `IS_VAULT_DELEVERAGED_BIT` could be toggled for repeated emergency operations (`WARNING` severity, later clarified by client)

3.3 INTEGRATION WITH EXTERNAL PROTOCOLS

Objective: Ensure seamless and secure interaction with Aave V3 and Morpho Blue lending markets, including correct handling of edge cases like liquidations, interest accrual, and parameter updates.

3.3.1 Audit Hypotheses

- Morpho market parameters (`oracle`, `irm`, `lltv`, `marketId`) could be misconfigured during initialization, leading to failed borrows/supplies or incorrect position tracking
- Oracle failures (stale prices, zero returns, or manipulation) might not trigger appropriate fallback mechanisms, causing incorrect LTV calculations
- Lending protocol liquidations could occur outside the vault's control, leaving `realBorrow` and `realCollateral` desynchronized from on-chain state

3.3.2 Audit Strategies

Connector Parameter Validation: For `MorphoConnector.initializeLendingConnectorData()`, cross-referenced input parameters against Morpho's `idToMarketParams` to detect:

- Non-existent `marketId` (zero-address loan/collateral tokens)
- Mismatched `oracle`, `irm`, or `lltv` values
- Incompatible token pairs for the specified market

Oracle Manipulation Scenarios: Simulated extreme price changes and flash loan attacks to test:

- `AaveV3OracleConnector.getPriceCollateralOracle()` and `getPriceBorrowOracle()` response to sudden 50%+ price swings
- Absence of TWAP/TWAT checks allowing single-block price manipulation (`WARNING` severity, `ACKNOWLEDGED` by client with plans for future TWAP-enabled connectors)

Liquidation Event Handling: Triggered Morpho/Aave liquidations externally while vault positions were active to verify:

- Correct `realBorrow`/`realCollateral` updates post-liquidation
- Impact on `totalAssets` calculations and share price stability

- Behavior when `realBorrow` drops to zero but `futureBorrow` remains negative (edge case affecting deposit availability, `INFO` severity)

eMode Compatibility Testing: For `AaveV3Connector`, validated that `emode` categories exist in Aave and support the configured token pairs (`WARNING` severity for missing validation)

3.3.3 Key Findings Validated

- `MorphoConnector.initializeLendingConnectorData()` lacked validation against `idToMarketParams`, allowing invalid market configurations (`WARNING` severity)
- Oracle connectors (`AaveV3OracleConnector`, `MorphoOracleConnector`) did not verify token support in oracles during construction, risking runtime failures (`WARNING` severity)
- No TWAP/price deviation checks exposed the protocol to oracle manipulation attacks, with client acknowledging the risk for initial wstETH/ETH and sUSDe/USDT vaults (`WARNING` severity, `ACKNOWLEDGEDv)

3.4 TOKEN HANDLING AND APPROVALS

Objective: Verify secure ERC20 token operations, including approvals, transfers, and balance accounting, to prevent unauthorized spending or asset loss.

3.4.1 Audit Hypotheses

- Zero-address assignments in `approve`, `transfer`, or `transferFrom` could lead to unintended token burns or locked funds
- Missing balance checks before subtraction operations might produce generic panic errors instead of user-friendly reverts
- Force-approve mechanisms could override user-set allowances without explicit consent

3.4.2 Audit Strategies

Zero-Address Exploit Testing: Attempted to call token functions with `address(0)` as `spender` / `recipient`:

- `approve(address(0), amount)` → Confirmed missing validation (`WARNING` severity)
- `transfer(address(0), amount)` and `transferFrom(sender, address(0), amount)` → Verified checks exist in collateral but not borrow vault functions (`WARNING` severity for `mint`, `redeem`, `withdraw`)

Balance Underflow Testing: Forced insufficient balance scenarios in `Transfer.transfer()`, `TransferFrom.transferFrom()`, and `ERC20._burn()` to confirm:

- Generic panic errors (0x11) instead of descriptive reverts (`WARNING` severity)
- Lack of explicit `require(balanceOf[sender] >= amount)` checks

Approval Race Condition Analysis: Tested standard approve/transferFrom race conditions and verified that the codebase did not mitigate this via `increaseAllowance` / `decreaseAllowance` patterns (out of scope per ERC20 standard compliance)

3.4.3 Key Findings Validated

- `Approve.approve()` allowed `spender = address(0)`, enabling nonsensical approvals (`WARNING` severity)
- `Transfer.transfer()`, `TransferFrom.transferFrom()`, and `ERC20._burn()` lacked explicit balance checks, producing unhelpful error messages (`WARNING` severity)
- `mint`, `mintCollateral`, `redeem`, `redeemCollateral`, `withdraw`, `withdrawCollateral` permitted `receiver = address(0)`, risking unintentional asset burns (`WARNING` severity)

3.5 DATA VALIDATION AND PARAMETER UPDATES

Objective: Ensure all input parameters are validated to prevent misconfigurations that could destabilize the protocol or enable exploits.

3.5.1 Audit Hypotheses

- Extreme LTV ratios (e.g., `targetLTV > 0.99`) could cause excessive leverage, amplifying losses during liquidations
- Division-by-zero risks exist in fee calculations if `maxGrowthFeeDivider`, `maxDeleverageFeeDivider`, or similar parameters are set to zero
- Missing checks in constructors could allow deployment with EOA addresses as modules or connectors

3.5.2 Audit Strategies

Boundary Value Testing: Tested extreme parameter values:

- `targetLTV = 0.99999` → Confirmed leverage approaches infinity (`leverage = 1/(1-targetLTV)`), enabling minimal price drops to trigger liquidations (`WARNING` severity, `ACKNOWLEDGED` by client as administrator responsibility)
- `maxGrowthFeeDivider = 0` → Verified division-by-zero risk in `ApplyMaxGrowthFee` (`WARNING` severity, `FIXED` via require check)

Constructor Validation Audits: Reviewed all constructors for missing checks:

- `ModulesProvider` allowed duplicate or zero addresses in module arrays (`WARNING` severity, `FIXED`)
- `AaveV3Connector`, `MorphoConnector`, oracle connectors lacked zero-address validation (`WARNING` severity, `FIXED`)

Slippage Configuration Testing: For `ConstantSlippageConnector`, tested:

- `borrowSlippage` or `collateralSlippage > 1` (e.g., `slippage = 1.5` → auction fee exceeds auction amount)
- Mismatched precision between borrow/collateral slippage values (`INFO` severity)

3.5.3 Key Findings Validated

- `_setMaxGrowthFee()` permitted `divider = 0`, causing division-by-zero in fee calculations (`WARNING` severity, `FIXED`)
- `ModulesProvider` constructor lacked address uniqueness/validity checks (`WARNING` severity, `FIXED`)
- `ConstantSlippageConnector` did not validate slippage bounds (e.g., `<= 1` and `>= minimum`), allowing nonsensical configurations (`WARNING` severity, `FIXED`)

3.6 REENTRANCY AND ATTACK VECTORS

Objective: Identify and mitigate reentrancy vulnerabilities and complex attack vectors involving cross-function state manipulation.

3.6.1 Audit Hypotheses

- Readonly reentrancy could occur in `executeLowLevelRebalance` if `borrowToken` is malicious and calls back into view functions (`totalAssets`, `convertToAssets`) mid-execution, exposing inflated share prices to external protocols
- "Good Samaritan" attacks could exploit third-party supply/repayment features in lending protocols to frontrun user deposits, forcing near-zero share mints (`MAJOR` severity vulnerability)
- Dust attacks via rounding errors could accumulate over time, enabling precision manipulation or share inflation

3.6.2 Audit Strategies

Readonly Reentrancy Simulation: Crafted a malicious `borrowToken` contract that:

- Triggers reentrancy during `transferBorrowToken()` in `executeLowLevelRebalance`
- Calls `convertToAssets(shares)` while `totalAssets` reflects updated lending protocol balances but `totalShares` is stale
- Confirmed inflated share prices could mislead external protocols relying on vault view functions (`WARNING` severity, `FIXED` via reentrancy guard adjustments)

Frontrunning Deposit Attacks: Implemented the "Good Samaritan" scenario:

- Attacker monitors mempool for user's `deposit(100_000 borrowTokens)` transaction
- Frontruns with large `supply(1_625_000 collateralTokens)` directly to Aave/Morpho on vault's behalf
- User's deposit executes with drastically lowered LTV, forcing `futurePaymentCollateral = deltaFutureCollateral * collateralSlippage` to approach deposited amount
- User receives ~1,455 shares instead of expected 100,000 (98.5% loss) as confirmed in provided test (`MAJOR` severity, `FIXED` via `SafeERC4626` helper contracts)

Dust Accumulation Testing: Analyzed rounding errors in:

- `mulDivDown` vs. `mulDivUp` usage across auction math and share conversions
- Impact of residual wei in `realBorrow` / `realCollateral` on high-leverage positions (leverage=100x amplifies 1 wei to 100 wei error)
- Long-term dust buildup across thousands of deposits/withdrawals

3.6.3 Key Findings Validated

- Readonly reentrancy in `executeLowLevelRebalance` confirmed via malicious `borrowToken` callback, exposing inflated `totalAssets` to external view function callers (`WARNING` severity, `FIXED`)
- "Good Samaritan" frontrunning attack validated, enabling near-total deposit value extraction (`MAJOR` severity, `FIXED` via off-chain helper contracts)
- Dust errors at high leverage could amplify rounding inaccuracies, though no direct exploit path was confirmed (noted in `WARNING` for extreme LTV configurations)

3.7 MATHEMATICAL CORRECTNESS

Objective: Verify precision, rounding consistency, and edge case handling in all mathematical operations, particularly in auction mechanics and share/asset conversions.

3.7.1 Audit Hypotheses

- Auction progress calculations could produce division-by-zero errors if `auctionDuration = 0`
- Inconsistent rounding directions (Down vs. Up) in `mulDiv` operations might favor attackers or lead to protocol insolvency over time
- High leverage (e.g., 100x) could amplify rounding errors, causing significant deviations in `totalAssets` calculations

3.7.2 Audit Strategies

Auction Math Verification: For `AuctionMath` library:

- Tested `calculateAuctionProgress()` with `auctionDuration = 0` → Confirmed division-by-zero risk (`INFO` severity, `FIXED` via minimum duration check)
- Validated reward/fee calculations at auction extremes (progress = 0%, 50%, 100%) to ensure monotonic behavior
- Simulated scenarios where `startAuction > auctionDuration` to detect invalid progress values

Rounding Direction Analysis: Catalogued all `mulDivDown` and `mulDivUp` calls:

- Asset-to-share conversions: Confirmed `mulDivDown` for deposits (favors protocol) and `mulDivUp` for withdrawals (favors users)
- Fee calculations: Verified `mulDivUp` usage to prevent fee underpayment
- LTV computations: Checked that conservative rounding (Up for borrow, Down for collateral) prevents false safety signals

High-Leverage Precision Testing: Created vaults with `targetLTV = 0.99` (leverage = 100x):

- Deposited 1 wei collateral and borrowed maximum amount
- Tracked cumulative rounding errors across 1000 deposit/withdraw cycles

- Confirmed errors remain bounded but noted amplification at extreme leverage (`ACKNOWLEDGED` in `WARNING` for unbounded LTV parameters)

Edge Case Fuzzing: Applied property-based testing with Echidna/Foundry to invariants:

- `totalAssets >= 0` across all operations
- `totalSupply` increases monotonically with deposits (excluding fees)
- `realBorrow + futureBorrow + futureRewardBorrow >= 0` (violated in acknowledged edge case where deposits revert)

3.7.3 Key Findings Validated

- `auctionDuration = 0` caused division-by-zero in auction progress calculations (`INFO` severity, `FIXED`)
- Rounding directions were generally correct, but dust errors at 100x leverage could accumulate (noted in unbounded LTV `WARNING`)
- Negative `borrow` edge case (when `futureBorrow < 0` exceeds `realBorrow`) blocked deposits with unclear error (`INFO` severity, `ACKNOWLEDGED` as connector design invariant)

3.8 EDGE CASES AND RISK SCENARIOS

Objective: Stress-test the protocol under extreme market conditions, protocol failures, and adversarial scenarios to evaluate resilience.

3.8.1 Audit Hypotheses

- Sharp collateral price drops (e.g., -30% in one block) could cause liquidations before auction mechanisms activate, leaving bad debt
- Morpho/Aave hacks or pauses might freeze vault assets, preventing withdrawals or rebalancing
- Negative future values (e.g., large auction creating `futureBorrow < 0`) could persist if auction fails to execute, destabilizing the vault

3.8.2 Audit Strategies

Price Crash Simulation: Mocked oracle to return collateral prices dropping 30%, 50%, 70% in single blocks:

- Monitored vault LTV crossing `maxSafeLTV` → `minProfitLTV` → liquidation threshold
- Verified auction creation triggers and bot incentive calculations
- Confirmed that without `TWAP` checks, single-block manipulation could trigger false liquidations (`ACKNOWLEDGED` in oracle manipulation `WARNING`)

Protocol Failure Scenarios: Simulated:

- Morpho market pause: Confirmed `supply` / `borrow` / `withdraw` calls revert, blocking all vault operations (expected behavior, no mitigation needed)
- Aave liquidity crisis: Tested withdrawals when `realCollateral > available liquidity` → Partial withdrawal success, but full withdrawal reverts (noted as external dependency risk)

Negative Future Value Persistence: Created auctions with `futureBorrow = -1000` and `futureCollateral = -2000`:

- Let auction expire without execution
- Verified protocol state allows new deposits/withdrawals despite negative futures (behavior undefined, flagged as potential invariant violation)

Dust Attack at Scale: Executed 10,000 micro-deposits (1 wei each) to test:

- Gas cost efficiency of `_mintToUser` at scale
- Cumulative rounding error impact on `totalSupply` vs. `totalAssets` ratio
- Potential griefing vector via bloated state (no critical issue found, gas limits prevent practical attack)

3.8.3 Key Findings Validated

- Oracle manipulation via single-block price swings confirmed as viable attack vector without TWAP (`WARNING` severity, `ACKNOWLEDGED`)
- Negative borrow edge case prevented deposits when `realBorrow = 0` and `futureBorrow < 0` (`INFO` severity, acknowledged as requiring external action to resolve)
- Fee mechanism unfairness: Temporarily setting `maxGrowthFee = 0` freezes `lastSeenTokenPrice`, causing sudden large fee upon re-enabling (`WARNING` severity, `ACKNOWLEDGED` as intended design)

3.9 OUT OF SCOPE

The following areas were explicitly excluded from the audit to maintain focus on core smart contract security:

1. Off-Chain Components: Frontend applications, APIs, monitoring bots, or keeper services interacting with the protocol
2. External Protocol Internals: Full security audits of Aave V3, Morpho Blue, Chainlink, or other dependencies (only integration points assessed)
3. Gas Optimization: Detailed gas profiling unless inefficiencies posed security risks (e.g., unbounded loops enabling DoS)
4. Economic Model Validation: Incentive structure viability, fee sustainability, or game-theoretic attack profitability beyond direct exploit paths
5. Regulatory Compliance: Legal or jurisdictional considerations for leveraged products

This structured approach ensured comprehensive coverage of critical security dimensions while maintaining audit efficiency and depth in high-risk areas.

4. FINDINGS REPORT

OXORIO

4.1 CRITICAL

No critical issues found.

4.2 MAJOR

M-01	Deposit may result in near-zero shares minted
Severity	**MAJOR**
Status	• FIXED

Description

In lending protocols, such as AAVE, a third party may repay the debt or increase the collateral of another user's position. This mechanism can be exploited to force an LTV protocol user to incur excessive costs when creating an auction.

Consider the following scenario:

- A user creates a transaction to call the `deposit` function to repay the borrow using borrow tokens.
- A "Good Samaritan" frontruns this transaction and performs a `supply` of a large amount of collateral tokens directly into the lending protocol on behalf of the LTV protocol.
- The user's transaction starts executing. Due to the Good Samaritan's actions, the vault's `ltv` becomes very low, and the user's deposit further decreases the `ltv`.
- This results in the user having to pay `futurePaymentCollateral` to create an auction, where the payment amount equals `deltaFutureCollateral * collateralSlippage`. The value of `deltaFutureCollateral` becomes very large because of the tokens supplied by the Good Samaritan.
- With a sufficiently large amount of tokens provided by the Good Samaritan, the user's payment will approach the amount of borrow tokens they deposited. As a result, the user effectively exchanges their borrow tokens for shares whose value tends toward `0`.

The Good Samaritan could be one of the vault's share holders. In this case, they would indirectly recover a portion of their "gift" proportional to their shareholding. In an extreme case, if the Good Samaritan is the only share holder in the vault, they would suffer no financial losses while being able to deprive users of their deposits as described above.

Additionally, the Good Samaritan may be affiliated with bots that execute auctions and receive part of the auction creation fee.

Test for the described scenario:

```
modifier initializeBalancedExpSlippage(
    address owner,
    address user,
    uint256 borrowAmount,
    uint256 supplyAmount,
    int256 futureBorrow,
    int256 futureCollateral,
    int256 auctionReward,
    uint256 borrowSlippage,
    uint256 collateralSlippage
) {
    vm.assume(owner != address(0));
    vm.assume(user != address(0));
    vm.assume(user != owner);
    vm.assume(int256(borrowAmount) >= futureBorrow);
    {
        BaseTestInit memory initData = BaseTestInit({
            owner: owner,
            guardian: address(123),
            governor: address(132),
            emergencyDeleverager: address(213),
            feeCollector: address(231),
            futureBorrow: futureBorrow,
            futureCollateral: futureCollateral,
            auctionReward: auctionReward,
            startAuction: uint56(1000 / 2),
            collateralSlippage: collateralSlippage,
            borrowSlippage: borrowSlippage,
            maxTotalAssetsInUnderlying: type(uint128).max,
            collateralAssets: supplyAmount,
            borrowAssets: borrowAmount,
            maxSafeLtvDividend: 9,
            maxSafeLtvDivider: 10,
            minProfitLtvDividend: 40,   // 40% minProfitLtv
            minProfitLtvDivider: 100,
            targetLtvDividend: 75,
            targetLtvDivider: 100,
            maxGrowthFeeDividend: 1,
            maxGrowthFeeDivider: 5,
            collateralPrice: 2 * 10 ** 20,
            borrowPrice: 10 ** 20,
            maxDeleverageFeeDividend: 1,
            maxDeleverageFeeDivider: 50,
            zeroAddressTokens: (supplyAmount*2 - borrowAmount) * 10
```

```
        });

        initializeDummyTest(initData);

        // transfer preminted tokens to owner
        vm.startPrank(address(0));
        IERC20(address(ltv)).safeTransfer(address(owner), ltv.balanceOf(address(0)));
        vm.stopPrank();
    }

    deal(address(borrowToken), user, type(uint112).max);
    deal(address(collateralToken), user, type(uint112).max);

    vm.startPrank(user);
    collateralToken.approve(address(ltv), type(uint112).max);
    borrowToken.approve(address(ltv), type(uint112).max);
    _;
}

function test_good_samaritan_front_run_drains_victim_assets(address owner, address user)
    public
    initializeBalancedExpSlippage(
        owner,
        user,
        (1_000_000) * 3,    // borrow assets
        (500_000) * 4,      // collateral assets
        0,                  // futureBorrow
        0,                  // futureCollateral
        0,                  // auctionReward
        10**16,             // borrowSlippage
        10**16              // collateralSlippage
    )
{
    // Create a victim address with ETH and borrow tokens
    address victim = makeAddr("victim");
    vm.deal(victim, 10 ether);
    IERC20(address(borrowToken)).safeTransfer(address(victim), 100_000);

    // Attacker front-runs the victim's deposit.
    // The attacker, acting as a "Good Samaritan", supplies a large amount of collateral
assets
    // directly into the lending protocol
    IERC20(address(collateralToken)).approve(address(lendingProtocol), 1_625_000);
    lendingProtocol.supply(address(collateralToken), 1_625_000);
    vm.stopPrank();
```

```
    vm.startPrank(victim);

    // The victim deposits 100_000 borrow tokens and receives shares
    IERC20(address(borrowToken)).approve(address(dummyLtv), 100_000);
    uint256 victimSharesInBorrow = dummyLtv.convertToAssets(dummyLtv.deposit(100_000,
victim));

    // The victim receives significantly fewer assets than they deposited
    assertLe(victimSharesInBorrow, 100_000);
    assertEq(victimSharesInBorrow, 1455);
    vm.stopPrank();
}
```

Recommendation

We recommend adding a sanity check in case the number of assets returned to the user is significantly lower than the amount deposited.

Update

Client's response

Another helper contract will be used to interract with vault. You can check `SafeERC4626` and `SafeERC4626Collateral` in this repo. Why it's separate: because there is no needs to be basic part of the protocol.

4.3 WARNING

W-01	Division by Zero Risk in `AdministrationSetters`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setMaxGrowthFee`	153

Description

The function `_setMaxGrowthFee` in the contract `AdministrationSetters` allows setting both `dividend` and `divider` to zero, which could cause a division-by-zero error during fee calculations.

```
function _setMaxGrowthFee(uint16 dividend, uint16 divider) internal {
    // Allows divider = 0, risking division-by-zero in fee calculations
    require(dividend >= 0 && dividend <= divider, "InvalidMaxGrowthFee");
    maxGrowthFeeDividend = dividend;
    maxGrowthFeeDivider = divider;
}
```

Recommendation

We recommend updating the validation to ensure `divider > 0`.

```
require(divider > 0 && dividend >= 0 && dividend <= divider, "InvalidMaxGrowthFee");
```

Update

Oxirio's response

Fixed at c37e41c9fefea5b51e6eb1be415188a4b2d6bc71

W-02	Missing Zero Address Validation in Constructors in `AaveV3Connector`, `MorphoConnector`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AaveV3Connector.sol	contract `AaveV3Connector` > `constructor`	17
MorphoConnector.sol	contract `MorphoConnector` > `constructor`	24

Description

The constructors in these contracts lack validation to prevent zero address assignments. This can lead to critical failures, such as failed external calls or unintended interactions with the zero address, potentially causing the contract to become unusable or exposing it to denial-of-service attacks. For example, in `AaveV3Connector.sol` at line 18, the constructor assigns addresses like `POOL` without checking `if (POOL == address(0)) revert();`. Similarly, in MorphoConnector.sol at line 28, `morphoBlue` is set without zero-address validation.

Recommendation

We recommend adding explicit checks in the constructors to ensure all addresses are non-zero before assignment. This prevents deployment with invalid configurations and enhances contract robustness. For example:

```
require(POOL != address(0), "Zero address not allowed");
```

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda

W-03	Missing Validation for eMode Category Existence in `AaveV3Connector`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AaveV3Connector.sol	contract `AaveV3Connector` > function `initializeLendingConnectorData`	69

Description

The function `initializeLendingConnectorData` in the contract `AaveV3Connector` does not validate whether the specified eMode category exists in the Aave protocol or whether the provided tokens (collateral and borrow) are supported under that eMode. This could allow initialization with invalid eMode values, leading to runtime errors during borrowing/lending operations or exposure to unsupported asset risks, as the Aave pool might reject or mishandle unsupported configurations.

Recommendation

We recommend integrating calls to Aave's pool configurator or data provider interfaces within `initializeLendingConnectorData` to verify eMode existence (e.g., via `getEModeCategoryData`), reverting if not supported. Here's an example code snippet to add after parameter validation:

```
// Fetch eMode data from Aave Pool
IPoolAddressesProvider provider = IPoolAddressesProvider(POOLAddressesProvider);
IPoolDataProvider dataProvider = IPoolDataProvider(provider.getPoolDataProvider());
(, uint256 ltv, uint256 liquidationThreshold, uint256 liquidationBonus, address priceSource,
bool isolationMode) = dataProvider.getEModeCategoryData(emode);

require(ltv > 0 || liquidationThreshold > 0, "Invalid eMode category");
```

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-04	Missing Verification of Support for Borrow and Collateral Tokens in AaveV3OracleConnector, MorphoOracleConnector
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AaveV3OracleConnector.sol	contract `AaveV3OracleConnector` > `constructor`	16
MorphoOracleConnector.sol	contract `MorphoOracleConnector` > `constructor`	16

Description

The constructors in these contracts initialize the `ORACLE` variable without verifying that the provided oracle supports pricing for the intended borrow and collateral token pair, nor confirming that these tokens are enabled in the underlying POOL for Aave or Morpho. The code directly assigns the oracle address but does not query the oracle (e.g., via `getSourceOfAsset` for Aave or market params for Morpho) to ensure valid price sources for the token pair, or check the POOL configuration to confirm token enablement. This could result in failed price queries, invalid feeds, or runtime errors during liquidation or borrowing operations.

Recommendation

We recommend adding post-assignment checks in the constructors to verify oracle support for the token pair by calling protocol-specific methods: for Aave, use `getSourceOfAsset` to ensure a valid price source is set for each token; for Morpho, fetch the market parameters via `idToMarketParams(marketId)` and confirm the oracle matches the one stored in MorphoConnector (or the fetched market oracle), reverting if mismatched or unsupported. This ensures consistency with lending connector parameters. For example (for AaveV3 Oracle):

```
address sourceCollateral = IAaveOracle(ORACLE).getSourceOfAsset(collateralToken);
address sourceBorrow = IAaveOracle(ORACLE).getSourceOfAsset(borrowToken);
require(sourceCollateral != address(0), "No price source for collateral");
require(sourceBorrow != address(0), "No price source for borrow");
```

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-05	Missing Data Validation in `MorphoConnector`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
MorphoConnector.sol	contract `MorphoConnector` > function `initializeLendingConnectorData`	107

Description

The function `initializeLendingConnectorData` in the contract `MorphoConnector` lacks proper validation of input parameters against the Morpho protocol's requirements. This could lead to invalid configurations or unexpected behavior when interacting with the Morpho lending protocol, potentially causing operational failures or security vulnerabilities.

```
function initializeLendingConnectorData(bytes memory data) external {
    (address oracle, address irm, uint256 lltv, bytes32 marketId) =
            abi.decode(data, (address, address, uint256, bytes32));

    // Missing validation for Morpho protocol compatibility (e.g., marketId, oracle, irm,
lltv)
    MorphoConnectorStorage storage s = _getMorphoConnectorStorage();

    s.oracle = oracle;
    s.irm = irm;
    s.lltv = lltv;

    lendingConnectorGetterData = abi.encode(marketId);
}
```

Recommendation

We recommend implementing validation checks by querying the Morpho Blue contract's `idToMarketParams` view function to retrieve the actual market parameters for the provided `marketId` and ensuring they match the input `oracle`, `irm`, and `lltv`. This confirms the market exists (by checking non-zero values) and the parameters are correct, preventing misconfigurations. For example:

```
// Assume morphoBlue is the Morpho Blue contract address, stored or passed appropriately
(
    address loanToken,
    address collateralToken,
    address fetchedOracle,
    address fetchedIrm,
    uint256 fetchedLltv
) = IMorpho(morphoBlue).idToMarketParams(marketId);

// Validate market exists (non-zero params)
require(loanToken != address(0), "Invalid market: loan token zero");
require(collateralToken != address(0), "Invalid market: collateral token zero");
require(fetchedLltv > 0, "Invalid market: LLTv zero");

// Validate parameters match inputs
require(fetchedOracle == oracle, "Invalid oracle");
require(fetchedIrm == irm, "Invalid IRM");
require(fetchedLltv == lltv, "Invalid LLTv");
```

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-06	Non-Upgradeable Contract in `MorphoConnector`, `AaveV3Connector`, `VaultBalanceAsLendingConnector`, `AaveV3OracleConnector`, `MorphoOracleConnector`, `ConstantSlippageConnector`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
AaveV3Connector.sol	contract `AaveV3Connector`	-
MorphoConnector.sol	contract `MorphoConnector`	-
VaultBalanceAsLendingConnector.sol	contract `VaultBalanceAsLendingConnector`	-
AaveV3OracleConnector.sol	contract `AaveV3OracleConnector`	-
MorphoOracleConnector.sol	contract `MorphoOracleConnector`	-
ConstantSlippageConnector.sol	contract `ConstantSlippageConnector`	-

Description

These contracts do not support upgradeability, unlike other system modules. This limitation could complicate future updates or bug fixes, requiring a full redeployment and potentially disrupting integration with the Morpho protocol.

Recommendation

We recommend adopting an upgradeable contract pattern, such as OpenZeppelin's `UUPSUpgradeable`, to ensure these connectors can be updated seamlessly in alignment with other system modules.

Update

Client's response

We do not expect these contracts to be upgradeable. If a specific pool requires an upgrade to its connector in the future, it will be handled individually.

Our upgrade mechanism relies on setters in the main contract. Since all connectors are stateless, they can be easily replaced by updating their addresses.

W-07	Weak Address Validation in `ModulesProvider`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
ModulesProvider.sol	contract `ModulesProvider` > `constructor`	31

Description

The constructor of the contract `ModulesProvider` does not validate input addresses during initialization, allowing duplicate addresses, Externally Owned Accounts (EOAs), or zero addresses. Validation occurs only during the `_delegate` function call, which may permit invalid configurations to persist until runtime errors occur.

```
constructor(address[] memory addresses, address[] memory modules) {
    // No validation for duplicate, EOA, or zero addresses in addresses or modules arrays
    _setModule(BORROW_VAULT_MODULE_SLOT, address(state.borrowVaultModule));
    _setModule(COLLATERAL_VAULT_MODULE_SLOT, address(state.collateralVaultModule));
    _setModule(LOW_LEVEL_REBALANCE_MODULE_SLOT, address(state.lowLevelRebalanceModule));
    _setModule(AUCTION_MODULE_SLOT, address(state.auctionModule));
    _setModule(ERC20_MODULE_SLOT, address(state.erc20Module));
    _setModule(ADMINISTRATION_MODULE_SLOT, address(state.administrationModule));
    _setModule(INITIALIZE_MODULE_SLOT, address(state.initializeModule));
}
```

Recommendation

We recommend introducing validation in the constructor to ensure addresses are unique, non-EOA (if applicable), and non-zero to prevent invalid configurations at deployment.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-08	No Signature Expiry in `WhitelistRegistry`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
WhitelistRegistry.sol	contract `WhitelistRegistry` > function `addAddressToWhitelistBySignature`	67

Description

The function `addAddressToWhitelistBySignature` in the contract `WhitelistRegistry` does not validate signature expiry, allowing potentially outdated or replayed signatures to be used, which could enable unauthorized access or replay attacks.

```
function addAddressToWhitelistBySignature(address account, uint8 v, bytes32 r, bytes32 s)
external {
    // Missing expiry timestamp validation for signature
    bytes32 digest = keccak256(abi.encodePacked(block.chainid, address(this), account));
    require(ECDSA.recover(digest, v, r, s) == signer, InvalidSignature());
    require(!wasWhitelistedBySignature[account], DoubleSignatureUse());
    wasWhitelistedBySignature[account] = true;
    isAddressWhitelisted[account] = true;
    emit AddressWhitelisted(account, true);
}
```

Recommendation

We recommend incorporating an expiry timestamp in the signature structure and validating it during processing. Adopt the EIP-712 standard for secure, typed data hashing to mitigate replay risks.

Update

Client's response

Here we decided to take a different approach: to prevent someone who has been removed from the whitelist from using their signature a second time.

Why do we want this approach? We aim to enable deployment of our dApp frontend without requiring any additional API. To achieve that, we want to keep signatures permanent.

W-09	Storage Incompatibility Risk in `AdministrationWrite`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AdministrationWrite.sol	contract `AdministrationWrite` > function `setModules`	172

Description

The function `setModules` in the contract `AdministrationWrite` allows changing module providers without verifying compatibility, which could disrupt the system if new modules have incompatible storage layouts or interfaces, potentially causing runtime errors or data corruption.

```
function setModules(IModules _modules) external onlyOwner {
    // No checks for storage layout or interface compatibility of _modules
    _setModules(_modules);
}
```

Recommendation

We recommend implementing compatibility checks or versioning for new modules to ensure consistent storage layouts and interfaces, preventing system disruptions.

Update

Client's response

Fixed at a1559794b1278687fe4a05edf308836883e4925f.

W-10	Unsafe Delegatecall in `InitializeWrite`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
InitializeWrite.sol	contract `InitializeWrite` > function `initialize`	24

Description

The function `initialize` in the contract `InitializeWrite` uses a delegatecall that returns `true` even when called on a zero address, masking potential errors. This behavior is inconsistent with other delegatecall implementations, increasing the risk of undetected failures.

```
function initialize(StateInitData memory initData, IModules modules) external initializer {
    _setModules(modules);

    // No check for modules.initializeModule() being non-zero address before delegatecall
    (bool isSuccess, bytes memory data) =

address(modules.initializeModule()).delegatecall(abi.encodeCall(IInitializeModule.initialize,
(initData)));
    revertWithDataIfNeeded(isSuccess, data);
}
```

Recommendation

We recommend adding validation to ensure the module address is non-zero before executing delegatecall and standardizing delegatecall handling across the system to properly manage return values.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda

W-11	Unclear Deleveraging State in `OnlyEmergencyDeleverager`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
OnlyEmergencyDeleverager.sol	contract `OnlyEmergencyDeleverager`	97

Description

The contract `OnlyEmergencyDeleverager` does not clearly specify whether the `IS_VAULT_DELEVERAGED_BIT` can be set multiple times after deleveraging or is intended as a one-time operation, potentially leading to state management issues during repeated emergency operations.

```
function deleverageAndWithdraw(...) {
    // Unclear if IS_VAULT_DELEVERAGED_BIT can be set multiple times or is one-time
    setBool(IS_VAULT_DELEVERAGED_BIT, true);
    // ...
}
```

Recommendation

We recommend clarifying the logic for setting `IS_VAULT_DELEVERAGED_BIT`, either allowing multiple sets with proper controls or enforcing a one-time operation, and documenting the behavior accordingly.

Update

Client's response

Fixed at 6d6f1c7b67a6c06eebcdf9120268c3413d7dcaec.

W-12	Unrestricted `approve` Function in `Approve`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Approve.sol	contract `Approve` > function `approve`	28

Description

The function `approve` in the contract `Approve` can be called by non-whitelisted users when whitelisting is enabled, potentially allowing unauthorized approvals of token spending.

```
function approve(address spender, uint256 amount) external isFunctionAllowed nonReentrant
returns (bool) {
    // Missing whitelist check for msg.sender when whitelisting is active
    allowance[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;
}
```

Recommendation

We recommend adding a modifier to restrict `approve` calls to whitelisted users when whitelisting is active.

Update

Client's response

For us, the whitelist exists to restrict certain users from receiving any assets from our protocol. The absence of approval whitelisting does not violate this intention.

W-13	Missing Zero Address Check in `Approve`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Approve.sol	contract `Approve` > function `approve`	29

Description

The function `approve` in the contract `Approve` does not check if the `spender` address is zero, which could result in unintended approvals or loss of funds in certain ERC20 implementations.

```
function approve(address spender, uint256 amount) external isFunctionAllowed nonReentrant
returns (bool) {
    // No check for spender != address(0)
    allowance[msg.sender][spender] = amount;
    emit Approval(msg.sender, spender, amount);
    return true;
}
```

Recommendation

We recommend adding a `require` statement to ensure `spender` is not `address(0)` to prevent invalid approvals.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-14	Missing Balance Checks in `Transfer`, `TransferFrom`, and `Burn`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Transfer.sol	contract `Transfer` > function `transfer`	38
TransferFrom.sol	contract `TransferFrom` > function `transferFrom`	40
ERC20.sol	contract `ERC20` > function `_burn`	29

Description

These functions lack explicit checks for sufficient balance, resulting in a generic panic error instead of a user-friendly revert message when funds are insufficient.

```
// Transfer.sol L38
function transfer(address recipient, uint256 amount)
    external
    isFunctionAllowed
    isReceiverWhitelisted(recipient)
    nonReentrant
    returns (bool)
{
    if (recipient == address(0)) {
        revert TransferToZeroAddress();
    }

    // Missing balance check for msg.sender before subtraction
    balanceOf[msg.sender] -= amount;
    balanceOf[recipient] += amount;
    emit Transfer(msg.sender, recipient, amount);
    return true;
}

// TransferFrom.sol L40
function transferFrom(address sender, address recipient, uint256 amount)
```

```
    external
    isFunctionAllowed
    isReceiverWhitelisted(recipient)
    nonReentrant
    returns (bool)
{
    if (recipient == address(0)) {
        revert TransferToZeroAddress();
    }

    _spendAllowance(sender, msg.sender, amount);
    // Missing balance check for sender before subtraction
    balanceOf[sender] -= amount;
    balanceOf[recipient] += amount;
    emit Transfer(sender, recipient, amount);
    return true;
}

// ERC20.sol L29
function _burn(address from, uint256 amount) internal {
    require(!_isWithdrawDisabled(boolSlot), WithdrawIsDisabled());
    // Missing balance check for from before subtraction
    balanceOf[from] -= amount;
    baseTotalSupply -= amount;
    emit Transfer(from, address(0), amount);
}
```

Recommendation

We recommend adding explicit `require` statements to check for sufficient balance and provide clear error messages, such as `require(balanceOf[msg.sender] >= amount, "Insufficient balance");`.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-15	Zero Address Receiver in `Mint`, `MintCollateral`, `Redeem`, `RedeemCollateral`, `Withdraw`, and `WithdrawCollateral`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Mint.sol	contract `Mint` > function `mint`	113
Redeem.sol	contract `Redeem` > function `redeem`	122
Withdraw.sol	contract `Withdraw` > function `withdraw`	121
MintCollateral.sol	contract `MintCollateral` > function `mintCollateral`	121
RedeemCollateral.sol	contract `RedeemCollateral` > function `redeemCollateral`	123
WithdrawCollateral.sol	contract `WithdrawCollateral` > function `withdrawCollateral`	123

Description

The `mint`, `redeem`, and `withdraw` functions in these contracts allow the `receiver` address to be zero, which could result in unintentional burning of tokens or assets. Similar issues exist in the collateral counterparts.

```
// Mint.sol L113
function mint(uint256 shares, address receiver) external isFunctionAllowed
isReceiverWhitelisted(receiver) returns (uint256 assets) {
    require(shares != 0, ZeroAmount());
    // No check for receiver != address(0)
    assets = _previewMint(shares);

    _transferFrom(msg.sender, assets);

    _mint(receiver, shares);

    emit Deposit(msg.sender, receiver, assets, shares);
}

// Redeem.sol L122
```

```
function redeem(uint256 shares, address receiver, address owner) external isFunctionAllowed
isReceiverWhitelisted(receiver) returns (uint256 assets) {
    require(shares != 0, ZeroAmount());
    // No check for receiver != address(0)
    if (msg.sender != owner) {
        _spendAllowance(owner, msg.sender, shares);
    }

    assets = _previewRedeem(shares);

    _burn(owner, shares);

    _transfer(receiver, assets);

    emit Withdraw(msg.sender, receiver, owner, assets, shares);
}

// Withdraw.sol L121
function withdraw(uint256 assets, address receiver, address owner) external
isFunctionAllowed isReceiverWhitelisted(receiver) returns (uint256 shares) {
    require(assets != 0, ZeroAmount());
    // No check for receiver != address(0)
    shares = _previewWithdraw(assets);

    if (msg.sender != owner) {
        _spendAllowance(owner, msg.sender, shares);
    }

    _burn(owner, shares);

    _transfer(receiver, assets);

    emit Withdraw(msg.sender, receiver, owner, assets, shares);
}
```

Recommendation

We recommend adding `require` statements to ensure the `receiver` is not `address(0)` in these functions.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-16	Unsafe Delegatecall to Zero Address in `AdministrationSetters`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setLendingConnector`	216
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setOracleConnector`	229
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setVaultBalanceAsLendingConnector`	179

Description

The `_setLendingConnector`, `_setOracleConnector`, and `_setVaultBalanceAsLendingConnector` functions allow delegatecalls to a zero address, which return `true` and mask errors, potentially causing undetected failures.

```
function _setLendingConnector(ILendingConnector _lendingConnector, bytes memory
lendingConnectorData) internal {
    address oldAddress = address(lendingConnector);
    lendingConnector = _lendingConnector;
    // No check for _lendingConnector != address(0) before delegatecall
    (bool success,) = address(lendingConnector).delegatecall(
        abi.encodeCall(ILendingConnector.initializeLendingConnectorData,
(lendingConnectorData))
    );
    require(success, FailedToSetLendingConnector(address(_lendingConnector),
lendingConnectorData));
    emit LendingConnectorUpdated(oldAddress, lendingConnectorData,
address(_lendingConnector), lendingConnectorData);
}

function _setOracleConnector(IOracleConnector _oracleConnector, bytes memory
oracleConnectorData) internal {
    address oldAddress = address(oracleConnector);
    oracleConnector = _oracleConnector;
```

```
    // No check for _oracleConnector != address(0) before delegatecall
    (bool success,) = address(oracleConnector).delegatecall(
        abi.encodeCall(IOracleConnector.initializeOracleConnectorData, (oracleConnectorData))
    );
    require(success, FailedToSetOracleConnector(address(_oracleConnector),
oracleConnectorData));
    emit OracleConnectorUpdated(oldAddress, oracleConnectorData, address(_oracleConnector),
oracleConnectorData);
}

function _setVaultBalanceAsLendingConnector(
    ILendingConnector _vaultBalanceAsLendingConnector,
    bytes memory vaultBalanceAsLendingConnectorGetterData
) internal {
    address oldAddress = address(vaultBalanceAsLendingConnector);
    vaultBalanceAsLendingConnector = ILendingConnector(_vaultBalanceAsLendingConnector);
    // No check for _vaultBalanceAsLendingConnector != address(0) before delegatecall
    (bool success,) = address(vaultBalanceAsLendingConnector).delegatecall(
        abi.encodeCall(ILendingConnector.initializeLendingConnectorData,
(vaultBalanceAsLendingConnectorGetterData))
    );
    require(
        success,
        FailedToSetVaultBalanceAsLendingConnector(
            address(_vaultBalanceAsLendingConnector),
vaultBalanceAsLendingConnectorGetterData
        )
    );
    emit VaultBalanceAsLendingConnectorUpdated(oldAddress,
address(_vaultBalanceAsLendingConnector));
}
```

Recommendation

We recommend adding validation to ensure connector addresses are non-zero before delegatecalls and handling return values appropriately or using the `_delegate` function from the CommonWrite contract.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-17	Unbounded LTV Parameters in `AdministrationSetters`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setMaxSafeLtv`	43
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setMinProfitLtv`	62
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setTargetLtv`	23

Description

The `_setMaxSafeLtv` and `_setMinProfitLtv` functions do not impose upper limits on `dividend` and `divider`, allowing extreme values that could lead to precision issues or unintended LTV ratios.

Also, in the function `_setTargetLtv` of the contract `AdministrationSetters`, before setting the `targetLtv`, the following check is performed: `targetLTV < maxSafeLTV <= 1`. Thus, the value of `targetLtv` can be set very close to `1`:

```
function _setTargetLtv(uint16 dividend, uint16 divider) internal {
    require(dividend >= 0 && dividend < divider, UnexpectedtargetLtv(dividend, divider));

    // ...
}

function _setMaxSafeLtv(uint16 dividend, uint16 divider) internal {
    // No upper limit on dividend or divider, risking extreme LTV values
    require(dividend > 0 && dividend <= divider, InvalidMaxSafeLtv(dividend, divider));
    uint16 oldDividend = maxSafeLtvDividend;
    uint16 oldDivider = maxSafeLtvDivider;
    maxSafeLtvDividend = dividend;
    maxSafeLtvDivider = divider;
    emit MaxSafeLtvUpdated(oldDividend, oldDivider, dividend, divider);
}
```

```
function _setMinProfitLtv(uint16 dividend, uint16 divider) internal {
    // No upper limit on dividend or divider, risking extreme LTV values
    require(dividend > 0 && dividend <= divider, InvalidMinProfitLtv(dividend, divider));
    uint16 oldDividend = minProfitLtvDividend;
    uint16 oldDivider = minProfitLtvDivider;
    minProfitLtvDividend = dividend;
    minProfitLtvDivider = divider;
    emit MinProfitLtvUpdated(oldDividend, oldDivider, dividend, divider);
}
```

This allows setting an excessively large leverage value:

```
leverage = 1 / (1 - targetLtv)
targetLtv=0.99999999 => leverage=99999999
```

However, setting `targetLtv` higher than the threshold `ltv` values of the lending protocol may result in position liquidation within the lending system, as bots and users would have no incentive to reduce the protocol's `ltv` in advance during unfavorable asset price changes.

It should also be noted that allowing excessively high `ltv` values enables an attacker to significantly amplify potential protocol losses. Even minor inaccuracies or residual amounts (e.g., single wei) in the protocol can become substantial at high leverage levels.

Recommendation

We recommend introducing upper bounds for the `dividend` and `divider` parameters to prevent extreme LTV values(e.g., `require(divider <= MAX_VALUE)`). Additionally, the `targetLtv` should be validated against the maximum allowable limit defined by the corresponding lending protocol.

Update

Client's response

We expect uint16 to be bounds for our parameters

W-18	No Pause Mechanism in `Initialize`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
Initialize.sol	contract `Initialize`	59

Description

The contract `Initialize` lacks a mechanism to pause the protocol during updates or migrations (e.g., from Morpho to Aave), exposing users to risks during transitions.

```
function initialize(StateInitData memory initData) external initializer {
    // No pausability mechanism (e.g., PausableUpgradeable) for protocol updates
    __Ownable_init(initData.owner);
    // ... no pause setup ...
}
```

Recommendation

We recommend integrating pausability using OpenZeppelin's `PausableUpgradeable` to allow safe protocol pauses during updates or migrations.

Update

Client's response

Fixed at 955d623d392f18198b28db28bbcaf5154f919dd3.

W-19	Unfair fee collection on price increase in `ApplyMaxGrowthFee`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
ApplyMaxGrowthFee.sol	contract `ApplyMaxGrowthFee` > function `applyMaxGrowthFee`	25

Description

In the function `applyMaxGrowthFee` of contract `ApplyMaxGrowthFee`, the state variable `lastSeenTokenPrice` is updated, and a fee is collected for the `feeCollector`, but only when `supplyAfterFee > supply`:

```
if (supplyAfterFee > supply) {
    _mintToFeeCollector(supplyAfterFee - supply);
    lastSeenTokenPrice = withdrawTotalAssets.mulDivUp(Constants.LAST_SEEN_PRICE_PRECISION,
supplyAfterFee);
}
```

The value of `supplyAfterFee` is calculated in the `_previewSupplyAfterFee` function and can only be greater than `supply` if the share price has increased since the last update of the `lastSeenTokenPrice` variable:

```
if (
    data.withdrawTotalAssets.mulDivDown(Constants.LAST_SEEN_PRICE_PRECISION, data.supply)
        <= data.lastSeenTokenPrice
) {
    return data.supply;
}

// ... calculation of supplyAfterFee
```

Thus, if the share price decreases after the `lastSeenTokenPrice` is set, the protocol will not charge a growth fee in favor of the `feeCollector`.

However, this leads to an uneven socialization of the fee distribution since it occurs through share dilution. Each time the share price rises above `lastSeenTokenPrice`, all users' shares are diluted by the fee amount. At the same time, when the price is below `lastSeenTokenPrice`, shareholders are exempt from the fee.

This imbalance becomes especially noticeable when the `maxGrowthFee` rate is temporarily set to `0`. In this case, `lastSeenTokenPrice` remains unchanged because `supplyAfterFee` always equals `supply`, even though the actual share price may be increasing.

Once `maxGrowthFee` is changed to a non-zero value, the first operation in the protocol will result in a fee being calculated based on the price difference between `lastSeenTokenPrice` and the current price. Consequently, users' shares will immediately lose value due to this one-time fee, which can be substantial.

Recommendation

We recommend reviewing and refactoring the fee calculation logic to avoid a sudden, one-time fee being charged solely to current share holders at that moment.

Update

Client's response

This is a deliberate design choice related to the fee mechanism and will not be changed. There is no practical way to implement a performance fee, because some users may have losses (for example, if they deposited when the price was higher) while others simultaneously have profits. In such a scenario, it is impossible to apply a unified performance fee to a tokenized position.
Therefore, we decided to proceed with the MaxGrowthFee design.

W-20	Readonly reentrancy during the execution of `lowLevel` functions in `ExecuteLowLevelRebalance`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
ExecuteLowLevelRebalance.sol	contract `ExecuteLowLevelRebalance` > function `executeLowLevelRebalance`	87

Description

In the function `executeLowLevelRebalance` of the contract `ExecuteLowLevelRebalance`, token balance changes are accounted for. It should be noted that the protocol does not store actual token balances in the contract but instead uses balances directly from the lending protocol. As a result, during the sequential invocation of `transfer` and `mint` functions within `executeLowLevelRebalance`, a readonly reentrancy attack is possible.

Assume that the `borrowToken` contract has been compromised and a hacker has gained the ability to perform a reentrant call.

Consider the following scenario. Using the `executeLowLevelRebalanceCollateral` function, the hacker deposits `deltaRealCollateralAssets=100` `stETH`. Under normal circumstances, the hacker should receive `deltaRealBorrowAssets=80` `ETH` and `deltaShares=20`. However, during the calculations in the `executeLowLevelRebalance` function, the hacker performs the following steps:

- Deposits funds into the lending protocol - `supply(deltaRealCollateralAsset)`;
- Borrows from the lending protocol - `borrow(deltaRealBorrowAssets)`;
- During the `borrowToken` transfer call, performs readonly reentrancy and calls a view function in an external or partner application, for example, `convertToAssets(shares)`. This function effectively performs `shares * (totalAssets/totalShares)`, where `totalAssets` is obtained from the lending protocol as `supply - borrow`, which has already been modified by the hacker in the previous steps, while `totalShares` is taken from the protocol's state and has not yet changed. Thus, the hacker obtains an inflated `convertToAssets(shares)` value in the external application.
- Returns to the execution of `executeLowLevelRebalance` and mints shares, increasing `totalShares`.

```
if (deltaRealCollateralAsset > 0) {
    collateralToken.safeTransferFrom(msg.sender, address(this),
uint256(deltaRealCollateralAsset));
    supply(uint256(deltaRealCollateralAsset));
}

// ...

if (deltaRealBorrowAssets > 0) {
    borrow(uint256(deltaRealBorrowAssets));
    transferBorrowToken(msg.sender, uint256(deltaRealBorrowAssets));
}

if (deltaShares > 0) {
    _mintToUser(msg.sender, uint256(deltaShares));
}
```

Such a scenario affects `view` functions that retrieve, via external calls to the lending protocol, data about asset amounts - specifically `totalAssets`, `max*`, `preview*`, `convertTo*`.

As a result, an attacker may obtain incorrect data from the vault in a third-party application integrated on top of the LTV protocol, which can put that external application at risk.

Recommendation

We recommend considering the addition of extra validation checks for `view` functions to detect incomplete state updates or partially cache actual token balances in the protocol's state.

Update

Client's response

Fixed at [34a566632819e01554ad2ad55af299163261a634](34a566632819e01554ad2ad55af299163261a634).

W-21	Missing Protections Against Price Manipulation in Oracle Functions in `AaveV3OracleConnector`, `MorphoOracleConnector`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
AaveV3OracleConnector.sol	contract `AaveV3OracleConnector` > function `getPriceCollateralOracle`	22
AaveV3OracleConnector.sol	contract `AaveV3OracleConnector` > function `getPriceBorrowOracle`	30
MorphoOracleConnector.sol	contract `MorphoOracleConnector` > function `getPriceCollateralOracle`	22
MorphoOracleConnector.sol	contract `MorphoOracleConnector` > function `getPriceBorrowOracle`	30

Description

The functions in these contracts directly fetch prices from the oracle without any safeguards against manipulation, such as using Time-Weighted Average Price (TWAP) or bounds checks. For example, in `AaveV3OracleConnector`, `getPriceCollateralOracle` calls the oracle's `getAssetPrice` raw and returns it unverified, making it vulnerable to flash loan attacks or oracle exploits where momentary price spikes could trigger erroneous liquidations or undercollateralized borrows. The other functions similarly lack TWAP (e.g., via Chainlink or Uniswap TWAP oracles), exposing the system to stale or manipulated data and amplifying risks in high-volatility scenarios.

Recommendation

We recommend implementing TWAP mechanisms or price deviation checks (e.g., compare against a secondary oracle or historical averages) in these functions to mitigate manipulation risks, reverting or adjusting prices if they fall outside safe bounds (e.g., ±10% deviation). For Aave-specific integration, here's an example for `getPriceCollateralOracle` using Aave's TWAP fallback if available:

```
function getPriceCollateralOracle(address collateral) external view returns (uint256) {
    // Primary oracle price
```

```
    uint256 currentPrice = IAaveOracle(aaveOracle).getAssetPrice(collateral);

    // TWAP check (example with a simple 1-hour average; use UniswapV3 or Chainlink for real
TWAP)
    uint256 twapPrice = _getTWAPPrice(collateral, 1 hours); // Implement _getTWAPPrice using
external TWAP oracle

    // Bounds check: Ensure prices are within 10% deviation
    uint256 minPrice = twapPrice * 90 / 100;
    uint256 maxPrice = twapPrice * 110 / 100;
    require(currentPrice >= minPrice && currentPrice <= maxPrice, "Price deviation too
high");

    return currentPrice;
}

// Helper function placeholder
function _getTWAPPrice(address asset, uint256 period) internal view returns (uint256) {
    // Integrate with TWAP source, e.g., UniswapV3Pool.consult()
    // Return averaged price over the period
    return 0; // Placeholder
}
```

Update

Client's response

- Our initial vault candidates for testing the entire system are wstETH over ETH and sUSDe over USDT vaults on Aave v3. We have investigated the current oracle structure and confirmed that the price feeds for Lido (wstETH) and Ethena (sUSDe) depend entirely on the respective vendors' own mechanisms. This means that only the vendors themselves can manipulate these prices, and therefore the relevant risk model must assume potential mistakes or malicious behavior by Lido or Ethena. We consciously accept these risks for the first version and proceed with a simple and straightforward oracle connector.
- The design proposed by auditors - or similar approaches that balance the trade-offs differently - will be implemented in future versions of our connectors. This is one of the primary reasons why we structured the oracle connectors to be modular: we can upgrade oracle logic without touching the core protocol.
- We have already introduced improvements to mitigate parts of this risk category. The soft liquidation mechanism. This enhancement was directly inspired by the oracle-manipulation concerns raised in the audit, as well as our own internal risk analysis. https://github.com/ltvprotocol/ltv_v0/pull/241.

W-22	Slippage value validation in `ConstantSlippageConnector`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
ConstantSlippageConnector.sol	contract `ConstantSlippageConnector` > function `initializeSlippageConnectorData`	33

Description

In the function `initializeSlippageConnectorData` of contract `ConstantSlippageConnector`, there is no validation of slippage values. For example, it is possible to set a slippage greater than `1`, which would result in an auction fee exceeding the auction amount itself.

Additionally, the absence of a reasonable minimum value may lead to incorrect slippage configuration. For instance, if `borrowSlippage` and `collateralSlippage` are mistakenly specified with different precision levels, while the contract uses a single hardcoded `SLIPPAGE_PRECISION` value for both.

Recommendation

We recommend adding additional validation checks when setting `slippage` to prevent the use of incorrect values.

Update

Client's response

Fixed at 2b6f4024dd7b3ad2fa3d615e8963e8a071812eda.

W-23	Inability to deposit when `borrow` is negative in `Vault`, `VaultCollateral`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
Vault.sol	contract `Vault` > function `getAvailableSpaceInShares`	31
VaultCollateral.sol	contract `VaultCollateral` > function `getAvailableSpaceInShares`	39

Description

In these functions, `totalAssets` is calculated, and the final value is cast from `int256` to `uint256`. For example, in the `getAvailableSpaceInShares` function:

```
// casting to uint256 is safe because collateral and borrow are considered to be greater
than zero
uint256 totalAssetsInUnderlying = uint256(collateral) - uint256(borrow);
```

However, it is possible for `collateral` or `borrow` to be negative.

These values are computed as the sum of `real + future + futureReward`:

```
data.collateral = int256(realCollateral) + data.futureCollateral + futureRewardCollateral;
data.borrow = int256(realBorrow) + data.futureBorrow + futureRewardBorrow;
```

Assume that the vault has a large negative auction and `futureRewardBorrow` equals zero, meaning `realBorrow > 0`, `futureBorrow < 0`, and `futureRewardBorrow = 0`.

In such a case, if the vault position in the lending protocol is liquidated or a Good Samaritan performs a large debt repayment directly into the lending protocol, the `realBorrow` value may drop to zero, resulting in `realBorrow = 0`, `futureBorrow < 0`, `futureReward = 0`. This means that the resulting `borrow` value will be negative.

In these situations, users will experience a revert and will not be able to perform `deposit` or `mint` operations for their tokens or shares until an action that improves the protocol's `ltv` is executed.

Recommendation

We recommend adding handling for such cases, including providing users with a clear error message to prevent confusion.

Update

Client's response

No, borrow always positive according to connectors design.

W-24	Missing handling of attempts to liquidate more collateral than available in `OnlyEmergencyDeleverager`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
OnlyEmergencyDeleverager.sol	contract `OnlyEmergencyDeleverager` > function `_liquidate`	127

Description

In the function `_liquidate` of contract `OnlyEmergencyDeleverager`, a liquidation bonus is added to the collateral amount to be deducted:

```
uint256 liquidationAmountCollateralInUnderlying = liquidationAmountBorrowInUnderlying;

liquidationAmountCollateralInUnderlying +=
    liquidationAmountCollateralInUnderlying.mulDivDown(bonusDividend, bonusDivider);

uint256 liquidationAmountCollateral = liquidationAmountCollateralInUnderlying.mulDivDown(
    10 ** collateralTokenDecimals,
    oracleConnector.getPriceCollateralOracle(_oracleConnectorGetterData)
);
```

However, the bonus percentage and the collateral base used to compute this bonus may result in the final collateral amount exceeding the actual collateral available on the lending protocol.

For example, if the debt is `99` USD, the collateral is `100` USD, and the liquidation bonus is `2%`, then when calling `deleverageAndWithdraw`, the full debt must be repaid:

```
liquidationAmountCollateralInUnderlying = liquidationAmountBorrowInUnderlying = 99
liquidationAmountCollateralInUnderlying += 1.98      // 99 * 2%
```

Thus, `liquidationAmountCollateralInUnderlying > 100`, which exceeds the available collateral.

In the case of a `deleverageAndWithdraw` call, this results in a revert when attempting to transfer collateral:

```
collateralToken.safeTransfer(msg.sender, liquidationAmountCollateral);
```

In the case of a `softLiquidation` call, this results in an underflow during the calculation:

```
uint256 expectedCollateralInUnderlying =
    (uint256(totalAssetsData.collateral) - liquidationAmountCollateralInUnderlying);
```

Recommendation

We recommend checking whether the calculated liquidation amount with bonus exceeds the total collateral available in the protocol and reverting with a clear and user-friendly error if so.

Update

Client's response

Fixed at [c8a080d1a79d03877ab8b1605ee061010afd5dba](c8a080d1a79d03877ab8b1605ee061010afd5dba).

4.4 INFO

I-01	Incorrect `uint256` Type for `eMode` Parameter in `AaveV3Connector`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
AaveV3Connector.sol	contract `AaveV3Connector` > function `initializeLendingConnectorData`	69

Description

The function `initializeLendingConnectorData` in the contract `AaveV3Connector` declares the `emode` parameter as `uint256`, whereas Aave's `eMode` categories are inherently limited to `uint8` (0-255 range per protocol specs). This mismatch could lead to unnecessary gas overhead from larger type handling or potential overflow issues if larger values are passed, though the protocol would likely revert internally.

Recommendation

We recommend changing the `emode` parameter type to `uint8` in initializeLendingConnectorData to align with Aave's constraints, improving gas efficiency and type safety.

Update

Client's response

Fixed at 587eef8281682dbd881c954d914563807dc96985.

I-02	Unsafe Use of abi.encodePacked for Hashing in `WhitelistRegistry`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
WhitelistRegistry.sol	contract `WhitelistRegistry` > function `addAddressToWhitelistBySignature`	67

Description

The function `addAddressToWhitelistBySignature` in the contract `WhitelistRegistry` uses `abi.encodePacked` for the message hash in signature verification. This approach risks hash collisions if inputs have ambiguous padding (e.g., mixing addresses and numbers), potentially allowing forged signatures and unauthorized whitelist additions.

Recommendation

We recommend replacing `abi.encodePacked` with `abi.encode` for domain separator and message hashing to ensure unambiguous encoding and prevent collision attacks.

Update

Client's response

Fixed at 6a6bffd7b9dd7f6864103c8d9c07e3397051128f.

I-03	Missing Minimum Value Enforcement for Auction Duration in `Initialize`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
Initialize.sol	contract `Initialize` > function `initialize`	46

Description

The function `initialize` in the contract `Initialize` does not enforce a minimum value for the `auctionDuration` parameter, allowing it to be set to zero. A zero duration would lead to a revert due to division by zero in other parts of the code (e.g., during auction progress calculations like `(block.timestamp - startTime) / auctionDuration`), potentially causing initialization to succeed but subsequent auction operations to fail unexpectedly without clear error handling.

Recommendation

We recommend adding a minimum duration check, e.g., `require(auctionDuration >= 1 hours, "Auction duration too short");`, to ensure functional auctions and prevent misconfigurations.

Update

Client's response

Fixed at fab6dccf75cbdbc9c47606d9c7161cb4b5617feb.

I-04	No sweep of excessive tokens in `LTV`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
LTV.sol	contract `LTV`	57

Description

The contract `LTV` lacks a function for withdrawing tokens that might have been mistakenly sent to the contract. As a result, any user can send tokens to the `LTV` contract, causing these tokens to become permanently locked within it.

Recommendation

We recommend adding a `sweep` function to withdraw tokens that were sent by mistake.

Update

Client's response

Fixed at ea75b55a51443fd04e1c9bd91323fbf8fce3edc7.

I-05	Redundant parameter type in function `_getBool` in `BoolReader`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
BoolReader.sol	contract `BoolReader` > function `_getBool`	44

Description

In the function `_getBool` of contract `BoolReader`, the second argument uses the type `uint256`:

```
function _getBool(uint8 boolSlot, uint256 bit) private pure returns (bool) {
    return boolSlot & (2 ** bit) != 0;
}
```

However, in the code, the second argument passed to the `_getBool` function is always a constant of type `uint8`:

```
uint8 public constant IS_DEPOSIT_DISABLED_BIT = 0;
uint8 public constant IS_WITHDRAW_DISABLED_BIT = 1;
uint8 public constant IS_WHITELIST_ACTIVATED_BIT = 2;
uint8 public constant IS_VAULT_DELEVERAGED_BIT = 3;
```

Recommendation

We recommend changing the parameter type of `bit` from `uint256` to `uint8` for optimization purposes.

Update

Client's response

Fixed at d8d055c969e81bcd7b44b9bfdb672cda359ddd28.

I-06	`feeCollector` cannot burn shares when withdrawals are disabled in `AdministrationSetters`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setIsDepositDisabled`	187
AdministrationSetters.sol	contract `AdministrationSetters` > function `_setIsWithdrawDisabled`	196

Description

These functions set flags to disable deposits and withdrawals, respectively. The `_setIsDepositDisabled` function prevents minting shares for users but does not prevent minting shares for the `feeCollector`:

```
function _mintToUser(address to, uint256 amount) internal isReceiverWhitelisted(to) {
    require(!_isDepositDisabled(boolSlot), DepositIsDisabled());
    _mint(to, amount);
}

function _mintToFeeCollector(uint256 amount) internal {
    _mint(feeCollector, amount);
}
```

However, the `_setIsWithdrawDisabled` function prohibits any share burning.

This creates a situation where the `feeCollector` cannot, for example, convert its shares into assets for protocol operational needs without lifting the withdrawal restriction.

Recommendation

We recommend considering the removal of the burn restriction for the `feeCollector`.

Update

Client's response

Not considered as problem for us. Fee collector should behave as normal user, he can’t deposit or withdraw assets when administration doesn’t let him to do it. All he can do - receive fees. It’s achitecture design to not give fee collector any extra permissions. For securing protocol we have guardian role.

I-07	Misleading comments and typos in `TotalAssets`, `TotalSupply`, `MaxGrowthFee`, `DeltaSharesAndDeltaRealBorrow`, `DeltaSharesAndDeltaRealCollateral`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
TotalSupply.sol	contract `TotalSupply` > function `totalSupply`	15
TotalAssets.sol	contract `TotalAssets` > function `_totalAssets`	36
MaxGrowthFee.sol	contract `MaxGrowthFee`	15
DeltaSharesAndDeltaRealBorrow.sol	contract `DeltaSharesAndDeltaRealBorrow`	558
DeltaSharesAndDeltaRealCollateral.sol	contract `DeltaSharesAndDeltaRealCollateral`	565
MaxGrowthFee.sol	contract `MaxGrowthFee` > function `_previewSupplyAfterFee`	33
DeltaSharesAndDeltaRealBorrow.sol	contract `DeltaSharesAndDeltaRealBorrow` > function `calculateDividentByDeltaSharesAndDeltaRealBorrow`	36
DeltaSharesAndDeltaRealCollateral.sol	contract `DeltaSharesAndDeltaRealCollateral` > function `calculateDividentByDeltaSharesAndRealCollateral`	36

Description

These code sections contain several typos and inaccurate comments:

- `VIRTUAL_ASSETS_AMOUNT` equals `10^4`, while the code comments refer to the addition of `100` tokens.
- Typographical errors such as "untill" and "roundind".
- In several parts of the code, the word "divident" is used, including in function names, whereas the correct spelling "dividend" is used elsewhere in the codebase.

Recommendation

We recommend correcting these typos to maintain codebase clarity and prevent misunderstandings.

Update

Client's response

Fixed at 508980bf47a1cfc4f4ffb8be5ac8297aa5c15301.

5 APPENDIX

OXORIO

5.1 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.

5.2 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.

5.2.1 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.

5.3 FINDINGS CLASSIFICATION REFERENCE

5.3.1 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.

5.3.2 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 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.

5.4 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, 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