altitude.

ALTITUDE V2 SMART CONTRACTS SECURITY AUDIT REPORT

OXORIO

MAY 9, 2025

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Altitude V2 Smart Contracts.

DeFi loans are typically over-collateralized and capital-inefficient, Altitude is a non-custodial protocol that optimizes DeFi loans.
Altitude actively manages users' debt and collateral in real-time, optimizing capital efficiency.

The Altitude protocol is a set of smart contracts that allows users to take over-collateralized loans from individual vaults, where a vault represents a supply-borrow currency pair (e.g., ETH-USDC). The Altitude Protocol both finds the best possible interest rates for users and activates dormant collateral by deploying a portion of this to generate yield.

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 143 smart contracts, encompassing 8570 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Altitude Labs and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of used 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 Altitude Labs to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by Altitude Labs, contributing to the robustness of the project.

As a result, following a comprehensive review, our auditors have verified that the Altitude V2 Smart Contracts, as of audited commit `6ab784e78d21431e89853339eaa4da402dadf0e7` |, has met the security and functionality requirements established for this audit, based on the code and documentation provided, and operates as intended within the defined scope.

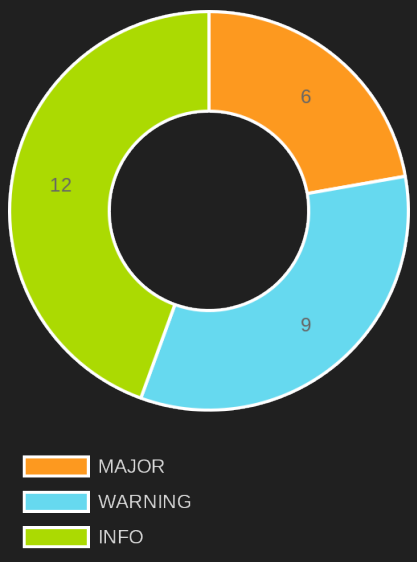
1.2 SUMMARY OF FINDINGS

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

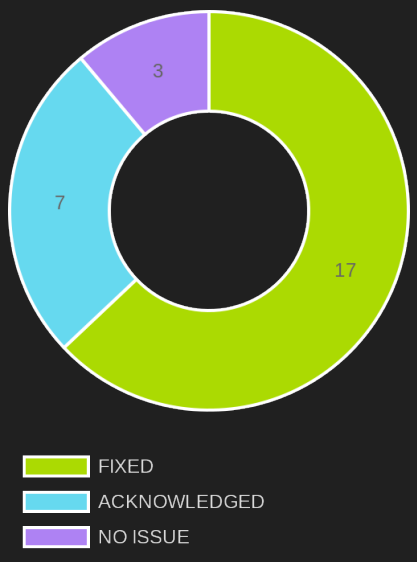
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

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

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**0**	0	0	0	0
MAJOR	**6**	0	5	0	1
WARNING	**9**	0	5	3	1
INFO	**12**	0	7	4	1
TOTAL	**27**	**0**	**17**	**7**	**3**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

2.1 DISCLAIMER

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

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

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

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

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

2.2 PROJECT BRIEF

Title	Description
Client	Altitude Labs
Project name	Altitude
Category	Lending, Asset Management
Website	altitude.fi
Repository	github.com/altitude-fi/altitude-v2
Documentation	docs.altitude.fi
Initial Commit	`c6aa8aa17293c430c55d508517bfe9e675d0e54b`
Final Commit	`6ab784e78d21431e89853339eaa4da402dadf0e7`
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
March 25, 2025	Client approached Oxorio requesting an audit.
March 27, 2025	The audit team commenced work on the project.
April 7, 2025	Submission of the preliminary report #1.
April 14, 2025	Submission of the comprehensive report.
May 7, 2025	Client feedback on the report was received.
May 9, 2025	Submission of the final report incorporating client's verified fixes.

2.4 AUDITED FILES

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

	File	Lines	Blanks	Comments	Code	Complexity
1	contracts/access/Ingress.sol	343	56	65	**222**	21%
2	contracts/common/ProxyExtension.sol	30	4	8	**18**	11%
3	contracts/common/ProxyInitializable.sol	51	11	13	**27**	30%
4	contracts/common/Roles.sol	14	2	6	**6**	0%
5	contracts/interfaces/external/IStETH.sol	8	2	1	**5**	0%
6	contracts/interfaces/external/IWStETH.sol	16	6	1	**9**	0%
7	contracts/interfaces/external/strategy/farming/Convex/IConvex.sol	39	14	1	**24**	0%
8	contracts/interfaces/external/strategy/farming/Convex/ICVXRewards.sol	40	18	1	**21**	0%
9	contracts/interfaces/external/strategy/farming/Curve/I3PoolZap.sol	31	5	1	**25**	0%
10	contracts/interfaces/external/strategy/farming/Curve/ICurve.sol	20	8	1	**11**	0%
11	contracts/interfaces/external/strategy/farming/Curve/ICurve2.sol	18	7	1	**10**	0%
12	contracts/interfaces/external/strategy/farming/Curve/ICurve4.sol	17	6	1	**10**	0%
13	contracts/interfaces/external/strategy/farming/Curve/ICurveNG.sol	14	5	1	**8**	0%
14	contracts/interfaces/external/strategy/lending/Aave/IAaveDebtToken.sol	48	14	1	**33**	0%
15	contracts/interfaces/external/strategy/lending/Aave/IAToken.sol	11	2	4	**5**	0%
16	contracts/interfaces/external/strategy/lending/Aave/IBaseLendingPool.sol	65	10	4	**51**	0%
17	contracts/interfaces/external/strategy/lending/Aave/IFlashLoanReceiver.sol	18	2	6	**10**	0%
17	contracts/interfaces/external/strategy/lending/Aave/ILendingPoolAddressProvider.sol	16	3	8	**5**	0%
19	contracts/interfaces/external/strategy/lending/Aave/IProtocolDataProvider.sol	71	7	1	**63**	0%
20	contracts/interfaces/external/strategy/lending/Aave/IWETH.sol	16	6	1	**9**	0%
21	contracts/interfaces/external/strategy/lending/Aave/IWETHGateway.sol	12	4	1	**7**	0%
22	contracts/interfaces/external/strategy/lending/Aave/v2/ILendingPool.sol	37	3	15	**19**	0%
23	contracts/interfaces/external/strategy/lending/Aave/v3/ILendingPool.sol	76	4	36	**36**	0%
24	contracts/interfaces/external/strategy/lending/Aave/v3/IPoolAddressesProvider.sol	223	30	155	**38**	0%
25	contracts/interfaces/external/strategy/lending/Aave/v3/IPriceOracleGetter.sol	63	8	44	**11**	0%
26	contracts/interfaces/external/strategy/lending/Aave/v3/IRewardsController.sol	148	14	91	**43**	0%
27	contracts/interfaces/external/strategy/lending/Aave/v3/IRewardsDistributor.sol	166	16	109	**41**	0%

	File	Lines	Blanks	Comments	Code	Complexity
28	contracts/interfaces/external/strategy/lending/migration/Aave/IAaveMigration.sol	21	4	1	**16**	0%
29	contracts/interfaces/external/strategy/lending/migration/Compound/ICompoundMigration.sol	20	3	1	**16**	0%
30	contracts/interfaces/external/strategy/lending/migration/Compound/ICompoundV2Migration.sol	12	3	2	**7**	0%
31	contracts/interfaces/external/strategy/lending/migration/ILendingProtocolMigration.sol	64	9	12	**43**	0%
32	contracts/interfaces/external/strategy/swap/ICurveRouter.sol	27	3	1	**23**	0%
33	contracts/interfaces/external/strategy/swap/IQuoter.sol	153	10	61	**82**	0%
34	contracts/interfaces/external/strategy/swap/IUniswapV2Router02.sol	11	1	1	**9**	0%
35	contracts/interfaces/internal/access/IIngress.sol	82	26	7	**49**	0%
36	contracts/interfaces/internal/flashloan/IFlashLoanStrategy.sol	16	4	5	**7**	0%
37	contracts/interfaces/internal/misc/IBorrowVerifier.sol	22	4	5	**13**	0%
38	contracts/interfaces/internal/misc/incentives/rebalance/IRebalanceIncentivesController.sol	30	11	5	**14**	0%
39	contracts/interfaces/internal/oracles/IChainlinkPrice.sol	18	4	5	**9**	0%
40	contracts/interfaces/internal/oracles/IPriceSource.sol	14	3	6	**5**	0%
41	contracts/interfaces/internal/strategy/farming/IConvexFarmStrategy.sol	78	25	5	**48**	0%
42	contracts/interfaces/internal/strategy/farming/IFarmBuffer.sol	30	11	4	**15**	0%
43	contracts/interfaces/internal/strategy/farming/IFarmBufferDispatcher.sol	24	7	4	**13**	0%
44	contracts/interfaces/internal/strategy/farming/IFarmDispatcher.sol	74	23	1	**50**	0%
45	contracts/interfaces/internal/strategy/farming/IFarmDropStrategy.sol	26	9	4	**13**	0%
46	contracts/interfaces/internal/strategy/farming/IFarmStrategy.sol	44	16	4	**24**	0%
47	contracts/interfaces/internal/strategy/farming/IMorphoVault.sol	19	6	4	**9**	0%
48	contracts/interfaces/internal/strategy/farming/IPendleFarmStrategy.sol	49	18	4	**27**	0%
49	contracts/interfaces/internal/strategy/IFlashLoanCallback.sol	10	2	4	**4**	0%
50	contracts/interfaces/internal/strategy/ISkimStrategy.sol	15	4	5	**6**	0%
51	contracts/interfaces/internal/strategy/lending/IAaveStrategy.sol	25	7	5	**13**	0%
52	contracts/interfaces/internal/strategy/lending/ICompStrategy.sol	16	2	5	**9**	0%
53	contracts/interfaces/internal/strategy/lending/ILenderStrategy.sol	69	29	4	**36**	0%
54	contracts/interfaces/internal/strategy/lending/IMorphoStrategy.sol	16	4	6	**6**	0%
55	contracts/interfaces/internal/strategy/swap/ISwapStrategy.sol	70	16	4	**50**	0%
56	contracts/interfaces/internal/strategy/swap/ISwapStrategyConfiguration.sol	14	5	1	**8**	0%
57	contracts/interfaces/internal/tokens/IDebtToken.sol	18	4	5	**9**	0%
58	contracts/interfaces/internal/tokens/IInterestToken.sol	66	24	5	**37**	0%
59	contracts/interfaces/internal/tokens/ISupplyToken.sol	19	5	5	**9**	0%
60	contracts/interfaces/internal/tokens/ITokensFactory.sol	41	12	4	**25**	0%
61	contracts/interfaces/internal/vault/extensions/configurable/IConfigurableVault.sol	36	7	5	**24**	0%
62	contracts/interfaces/internal/vault/extensions/groomable/IGroomableManager.sol	33	7	5	**21**	0%

	File	Lines	Blanks	Comments	Code	Complexity
63	contracts/interfaces/internal/vault/extensions/groomable/IGroomableVault.sol	24	8	5	**11**	0%
64	contracts/interfaces/internal/vault/extensions/harvestable/IHarvestableManager.sol	40	9	5	**26**	0%
65	contracts/interfaces/internal/vault/extensions/harvestable/IHarvestableVault.sol	26	8	4	**14**	0%
66	contracts/interfaces/internal/vault/extensions/IVaultExtensions.sol	13	3	4	**6**	0%
67	contracts/interfaces/internal/vault/extensions/liquidatable/ILiquidatableManager.sol	20	5	5	**10**	0%
68	contracts/interfaces/internal/vault/extensions/liquidatable/ILiquidatableVault.sol	17	5	4	**8**	0%
69	contracts/interfaces/internal/vault/extensions/snapshotable/ISnapshotableManager.sol	29	7	4	**18**	0%
70	contracts/interfaces/internal/vault/extensions/snapshotable/ISnapshotableVault.sol	26	7	4	**15**	0%
71	contracts/interfaces/internal/vault/extensions/supply-loss/ISupplyLossManager.sol	18	6	4	**8**	0%
72	contracts/interfaces/internal/vault/extensions/supply-loss/ISupplyLossVault.sol	13	3	4	**6**	0%
73	contracts/interfaces/internal/vault/IInterestVault.sol	12	3	4	**5**	0%
74	contracts/interfaces/internal/vault/IVaultCore.sol	71	15	5	**51**	0%
75	contracts/interfaces/internal/vault/IVaultCoreV1Initializer.sol	32	9	8	**15**	0%
76	contracts/interfaces/internal/vault/IVaultRegistry.sol	162	36	5	**121**	0%
77	contracts/interfaces/internal/vault/IVaultStorage.sol	57	24	5	**28**	0%
78	contracts/libraries/types/CommonTypes.sol	32	4	9	**19**	0%
79	contracts/libraries/types/HarvestTypes.sol	56	6	10	**40**	0%
80	contracts/libraries/types/SupplyLossTypes.sol	26	3	8	**15**	0%
81	contracts/libraries/types/VaultTypes.sol	78	10	15	**53**	0%
82	contracts/libraries/uniswap-v3/FullMath.sol	120	10	51	**59**	20%
83	contracts/libraries/uniswap-v3/OracleLibrary.sol	70	10	17	**43**	14%
84	contracts/libraries/uniswap-v3/PoolAddress.sol	47	4	12	**31**	3%
85	contracts/libraries/uniswap-v3/TickMath.sol	216	16	22	**178**	39%
86	contracts/libraries/uniswap-v3/TransferHelper.sol	66	9	13	**44**	18%
87	contracts/libraries/utils/CommitMath.sol	365	46	121	**198**	14%
88	contracts/libraries/utils/FlashLoan.sol	22	2	11	**9**	0%
89	contracts/libraries/utils/HealthFactorCalculator.sol	88	9	30	**49**	6%
90	contracts/libraries/utils/Utils.sol	70	6	20	**44**	27%
91	contracts/misc/BorrowVerifier.sol	59	8	12	**39**	8%
92	contracts/misc/incentives/rebalance/RebalanceIncentivesController.sol	121	22	29	**70**	9%
93	contracts/oracles/ChainlinkPrice.sol	169	28	48	**93**	19%
94	contracts/oracles/UniswapV3Twap.sol	152	24	39	**89**	19%
95	contracts/strategies/farming/FarmBuffer.sol	102	20	22	**60**	10%
96	contracts/strategies/farming/FarmBufferDispatcher.sol	101	19	22	**60**	10%
97	contracts/strategies/farming/FarmDispatcher.sol	404	69	85	**250**	19%
98	contracts/strategies/farming/strategies/convex/StrategyGenericPool.sol	299	44	74	**181**	23%

	File	Lines	Blanks	Comments	Code	Complexity
99	contracts/strategies/farming/strategies/convex/StrategyMeta3Pool.sol	55	9	15	**31**	6%
100	contracts/strategies/farming/strategies/convex/StrategyMetaPool.sol	54	8	15	**31**	6%
101	contracts/strategies/farming/strategies/convex/StrategyStable2Pool.sol	54	8	15	**31**	6%
102	contracts/strategies/farming/strategies/convex/StrategyStableNGPool.sol	56	9	15	**32**	6%
103	contracts/strategies/farming/strategies/FarmDropStrategy.sol	163	23	40	**100**	12%
104	contracts/strategies/farming/strategies/FarmStrategy.sol	191	35	45	**111**	13%
105	contracts/strategies/farming/strategies/morpho/MorphoVault.sol	131	21	27	**83**	17%
106	contracts/strategies/farming/strategies/pendle/StrategyPendleBase.sol	179	24	26	**129**	9%
107	contracts/strategies/farming/strategies/pendle/StrategyPendleLP.sol	198	24	26	**148**	11%
108	contracts/strategies/farming/strategies/pendle/StrategyPendlePT.sol	157	18	19	**120**	11%
109	contracts/strategies/flashloan/Aavev2FlashLoanStrategy.sol	59	7	13	**39**	0%
110	contracts/strategies/flashloan/FlashLoanStrategy.sol	77	14	19	**44**	14%
111	contracts/strategies/flashloan/MorphoFlashLoanStrategy.sol	39	6	12	**21**	0%
112	contracts/strategies/lending/aave/v3/StrategyAaveV3.sol	194	27	47	**120**	3%
113	contracts/strategies/lending/LenderStrategy.sol	217	44	46	**127**	13%
114	contracts/strategies/lending/morpho/StrategyMorphoV1.sol	300	34	52	**214**	11%
115	contracts/strategies/SkimStrategy.sol	50	8	13	**29**	17%
116	contracts/strategies/swap/CurveV2Strategy.sol	314	34	69	**211**	15%
117	contracts/strategies/swap/SwapStrategy.sol	140	23	43	**74**	11%
118	contracts/strategies/swap/SwapStrategyConfiguration.sol	29	5	11	**13**	0%
119	contracts/strategies/swap/UniswapV3Strategy.sol	244	36	60	**148**	11%
120	contracts/tokens/DebtToken.sol	109	18	26	**65**	3%
121	contracts/tokens/InterestToken.sol	283	53	80	**150**	10%
122	contracts/tokens/SupplyToken.sol	152	22	40	**90**	4%
123	contracts/tokens/TokensFactory.sol	144	20	25	**99**	6%
124	contracts/vaults/v1/base/InterestVault.sol	19	4	1	**14**	7%
125	contracts/vaults/v1/base/JoiningBlockVault.sol	30	5	8	**17**	6%
126	contracts/vaults/v1/base/VaultStorage.sol	114	32	34	**48**	4%
127	contracts/vaults/v1/ERC20/VaultERC20.sol	26	4	11	**11**	0%
128	contracts/vaults/v1/ETH/VaultETH.sol	36	6	11	**19**	16%
129	contracts/vaults/v1/extensions/configurable/ConfigurableManager.sol	111	12	30	**69**	16%
130	contracts/vaults/v1/extensions/configurable/ConfigurableVault.sol	78	7	11	**60**	0%
131	contracts/vaults/v1/extensions/groomable/GroomableManager.sol	215	38	39	**138**	21%
132	contracts/vaults/v1/extensions/groomable/GroomableVault.sol	71	10	17	**44**	2%
133	contracts/vaults/v1/extensions/liquidatable/LiquidatableManager.sol	119	22	19	**78**	12%
134	contracts/vaults/v1/extensions/liquidatable/LiquidatableVault.sol	84	11	22	**51**	4%
135	contracts/vaults/v1/extensions/snapshotable/harvest/HarvestableManager.sol	376	55	101	**220**	19%

	File	Lines	Blanks	Comments	Code	Complexity
136	contracts/vaults/v1/extensions/snapshotable/harvest/HarvestableVault.sol	134	19	33	**82**	7%
137	contracts/vaults/v1/extensions/snapshotable/SnapshotableManager.sol	157	20	43	**94**	13%
138	contracts/vaults/v1/extensions/snapshotable/SnapshotableVault.sol	136	17	34	**85**	5%
139	contracts/vaults/v1/extensions/snapshotable/supply-loss/SupplyLossManager.sol	305	52	93	**160**	12%
140	contracts/vaults/v1/extensions/snapshotable/supply-loss/SupplyLossVault.sol	32	6	11	**15**	0%
141	contracts/vaults/v1/VaultCore.sol	471	74	112	**285**	11%
142	contracts/vaults/v1/VaultInitializer.sol	121	17	29	**75**	11%
143	contracts/vaults/v1/VaultRegistry.sol	457	58	98	**301**	6%
Total		**14001**	**2339**	**3092**	**8570**	**9%**

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 Altitude protocol is a set of smart contracts that allows users to take over-collateralized loans from individual vaults, where a vault represents a supply-borrow currency pair (e.g., ETH-USDC). The Altitude Protocol both finds the best possible interest rates for users and activates dormant collateral by deploying a portion of this to generate yield.

Each supply/borrow currency pair will be managed in a single vault, for example, an ETH-USDC vault where a user can supply ETH and borrow USDC. These vaults will be created based on user demand. Each vault will facilitate a few key functions including:

- **Vault Core:** Main user interaction point with the contracts for user deposit, borrow, repay, withdraw, etc.
- **Lender Strategy:** Deploying user assets into the lenders where the best rates can be achieved.
- **Farm Dispatcher:** Deploying previously dormant capital (active capital) in one or more farm strategy to earn interest on the user's behalf.
- **Rebalancing:** Ensuring the vault position stays healthy by borrowing and repaying lenders when needed.
- **Harvesting:** Recognizing earnings from the Farm Optimizations and enabling distribution to users.
- **Position Update:** Updating user balances to recognize their latest position, including earnings from the farm strategy.
- **Liquidations:** Enabling user funds to be liquidated when the user's position becomes unhealthy.
- **Tokenization:** Tokenizing user supply and debt positions.

2.6 CODEBASE QUALITY ASSESSMENT

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

Category	Assessment	Result
Access Control	The project's codebase implements a robust access control mechanism with multiple differentiated roles to manage system functionalities efficiently. Additionally, it includes validations to filter out prohibited addresses under sanctions and permits authorized addresses.	**Good**
Arithmetic	The project diligently manages arithmetic operations to ensure accuracy and security. However, specific concerns outlined in `M-07`, `M-08` warrant further review to reinforce the robustness of these operations.	**Good**
Complexity	The project benefits from a well-structured modular architecture that enhances readability and maintainability. However, the complexity introduced by the upgradeable scheme using proxy extensions warrants careful consideration.	**Good**
Data Validation	The project performs data validation across many components, but a significant portion of the issues highlighted in this report stem from insufficient validation processes. It is crucial to enhance the validation mechanisms to address these deficiencies and improve the overall robustness of the system.	**Fair**
Decentralization	The project does not incorporate a decentralized approach to management, and therefore, the metric is not applicable in this context.	**Not Applicable**

Category	Assessment	Result
Documentation	The project's documentation effectively explains the complex logic integral to the system. However, it lacks comprehensive details on the architecture and the interactions among contracts. This absence of a detailed architectural blueprint could impede understanding of the overall system design and operational coherence.	**Good**
External Dependencies	The project effectively manages a significant number of external dependencies, including integrations with prominent projects such as `Morpho`, `Convex`, `Curve`, `Aave` and `Uniswap V3`. While some of these integrations were outside the scope of this audit, those that were reviewed exhibited robust implementation practices.	**Excellent**
Error Handling	The project demonstrates competent exception handling throughout the codebase. However, it is important to address the issues outlined in the report that highlight potential error scenarios, including several instances where necessary `revert` statements are missing.	**Good**
Logging and Monitoring	The project exhibits excellent logging capabilities, recording all important events within the system. This comprehensive logging framework enables the effective use of third-party monitoring services such as `Tenderly` or `Forta`, which facilitate real-time data analysis and enhance the ability to track system performance and security incidents accurately.	**Excellent**
Low-Level Calls	The project is free from low-level calls, ensuring a higher level of security by avoiding potential pitfalls associated with direct, low-level interactions with the blockchain.	**Not Applicable**
Testing and Verification	The codebase exhibits commendable test coverage, demonstrating a strong commitment to verifying functionality and reliability. However, there are notable gaps in the test suite, particularly in key scenarios that remain untested. Addressing these gaps by including these crucial test cases will enhance the robustness of the testing framework and ensure more comprehensive verification of the system's behavior under various conditions.	**Fair**

2.7 FINDINGS BREAKDOWN BY FILE

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

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
contracts/strategies/farming/FarmDispatcher.sol	**9**	0	3	4	2
contracts/strategies/farming/FarmBufferDispatcher.sol	**4**	0	1	1	2
contracts/vaults/v1/VaultCore.sol	**3**	0	1	2	0
contracts/strategies/SkimStrategy.sol	**2**	0	0	1	1
contracts/strategies/farming/FarmBuffer.sol	**2**	0	0	0	2
contracts/strategies/farming/strategies/FarmStrategy.sol	**2**	0	0	1	1
contracts/strategies/farming/strategies/pendle/StrategyPendleBase.sol	**2**	0	0	1	1
contracts/libraries/utils/CommitMath.sol	**1**	0	0	1	0
contracts/strategies/farming/strategies/convex/StrategyGenericPool.sol	**1**	0	0	1	0
contracts/strategies/farming/strategies/morpho/MorphoVault.sol	**1**	0	0	1	0
contracts/strategies/lending/LenderStrategy.sol	**1**	0	0	0	1
contracts/vaults/v1/extensions/liquidatable/LiquidatableManager.sol	**1**	0	0	0	1
contracts/vaults/v1/extensions/snapshotable/harvest/HarvestableManager.sol	**1**	0	1	0	0
contracts/vaults/v1/extensions/snapshotable/supply-loss/SupplyLossManager.sol	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on 143 smart contracts, revealing 0 critical and 6 major issues, along with numerous warnings and informational notes. The audit highlighted several areas requiring improvement, such as logical discrepancies in the fund withdrawal mechanisms, risks related to inaccurate cumulative withdrawal calculations, a lack of sufficient condition checks for edge cases, and inadequate documentation that hampers understanding of critical processes. These issues emphasize the importance of addressing both functional flaws and security vulnerabilities.

The proposed changes are aimed at improving the overall fairness, efficiency, and security of the system. Enhancing the loss distribution logic is intended to ensure equitable treatment between users with different actions, such as borrowers and non-borrowers, by preventing any undue disadvantage to one group over the other. Revising the buffer management process seeks to prioritize user withdrawals over buffer replenishment, ensuring that users' immediate needs are met before focusing on system reserves. Maintaining consistency in roles and permissions across strategies will help streamline operations and prevent potential misuse or unauthorized access.

Furthermore, robust parameter validation, including checks for strategies in "emergency mode" and ensuring that critical addresses are not set to zero, is essential for safeguarding against fund mismanagement or operational errors. Addressing issues such as rounding errors in loss calculations, optimizing the deactivation and withdrawal processes for strategies, and resolving potential inefficiencies—such as over-approvals or inadequate withdrawal mechanisms—will enhance the overall resiliency and reliability of the system.

Following our initial audit, Altitude Labs worked closely with our team to address the identified issues. Through multiple rounds of interaction, all identified issues have been successfully addressed or formally acknowledged.

As a result, the project has passed our audit. Our auditors have verified that the Altitude V2, as of audited commit `6ab784e78d21431e89853339eaa4da402dadf0e7`, 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.

To enhance the project's security and readiness for production, we recommend conducting additional security reviews after each significant contract change. Additionally, increasing test coverage is advised to ensure thorough validation and reliability of the codebase.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

No critical issues found.

3.2 MAJOR

M-01	Repeated calls draining all funds from strategies in `FarmBufferDispatcher`
Severity	**MAJOR**
Status	• NO ISSUE

Location

File	Location	Line
FarmBufferDispatcher.sol	Contract `FarmBufferDispatcher` > Function `decreaseBufferCapacity`	82

Description

In the `decreaseBufferCapacity` function of the `FarmBufferDispatcher` contract, funds are withdrawn from strategies to fill the buffer and subsequently emptied entirely, sending all funds to the caller:

```
uint256 amountWithdrawn = super._withdraw(farmBuffer.capacityMissing());
_fillBuffer(amountWithdrawn);

uint256 capacity = farmBuffer.capacity();
farmBuffer.decreaseCapacity(msg.sender);
```

There is nothing preventing the caller from invoking this function repeatedly until all funds from the strategies are depleted. If the `Roles.BETA` role or its equivalent in the permissioning system is compromised, all user funds in strategies could be stolen.

Recommendation

We recommend limiting the maximum amount that can be withdrawn from the buffer per call to `decreaseBufferCapacity` to prevent abuse and potential theft of user funds.

Update

Client's response

User funds are not at risk as the buffer is pre-funded and the amount withdrawn is limited to the buffer capacity.

M-02	Deposits allowed in strategies with `inEmergency` mode in `FarmDispatcher`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	Contract `FarmDispatcher` > Function `_dispatch`	245

Description

The `_dispatch` function of the `FarmDispatcher` contract performs deposits into strategies. However, there are no restrictions preventing calls to the deposit function when the strategy is in `inEmergency` mode (`inEmergency=true`).

In other words, after a strategy owner invokes the `emergencyWithdraw` function to pull all available funds from the strategy, user funds can still be sent into the strategy.

Withdrawing funds from a strategy in `inEmergency=true` mode is not possible. In order to withdraw, the `inEmergency` status must first be deactivated (`inEmergency=false`) via the `emergencyDeactivateStrategy` function in the dispatcher. However, this function also deactivates the strategy, making it unavailable for withdrawals via the dispatcher.

If users need to withdraw from such a strategy, the only option is to re-add the strategy to the dispatcher's active pool via the `addStrategy` function. However, this action may be undesirable, given that there was a compelling reason to activate the emergency mode in the first place, which generally indicates the strategy's unreliability.

Recommendation

We recommend prohibiting deposits into strategies that are in `inEmergency` mode.

Update

Fixed at `6430b1bcf44fa9eb679632e9be90a81a76691b9c`

Client's response

To simplify we have removed the `inEmergency` state and separated management:

- dispatcher: controls if funds should/shouldn't be deposited into the strategy (activating, deactivating, adjust cap, etc.)
- strategy: controls any recovery of funds in case of an emergency

M-03	Incomplete withdrawals in the `withdraw` function in `FarmDispatcher`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	Contract `FarmDispatcher` > Function `_withdraw`	316

Description

In the `_withdraw` function of the `FarmDispatcher` contract, funds are withdrawn sequentially from strategies in a loop. If the full requested withdrawal amount cannot be fulfilled by one strategy, the remaining amount is withdrawn from subsequent strategies. The variable `toWithdraw` is reduced by the amount withdrawn in each iteration.

However, instead of comparing the remaining amount (`toWithdraw`) against the amount withdrawn in the current iteration (`strategyWithdrawn`), it is compared against the total amount withdrawn so far (`withdrawn`):

```
try IFarmStrategy(strategyAddr).withdraw(toWithdraw) returns (uint256 strategyWithdrawn) {
    withdrawn += strategyWithdrawn;

    // Decrease totalDeposit to release capacity
    // ...

    // Remainder to withdraw in the next iteration
    if (toWithdraw < withdrawn) {
        toWithdraw = 0;
    } else {
        toWithdraw = requested - withdrawn;
    }
```

This causes the loop to terminate prematurely when the cumulative amount withdrawn exceeds the remaining amount needed (`toWithdraw < withdrawn`). Consequently, the required amount may not be fully withdrawn even though sufficient funds remain in the strategies.

Recommendation

We recommend adjusting the logic so that `toWithdraw` comparisons are based on the amount withdrawn in the current iteration rather than the cumulative `withdrawn` amount.

Update

Fixed at `ba52aec6246be447f3474be5c7e0c9713ef99a1f`

Client's response

Refactored logic and fixed incomplete withdrawals bug by compare to requested.

M-04	Using `uint256.max` for full balance withdrawals may cause overflows in `FarmDispatcher`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	Contract `FarmDispatcher` > Function `setStrategyMax`	93

Description

In the `setStrategyMax` function of the `FarmDispatcher` contract, when `newMax=0`, the code assumes that all funds from a strategy should be withdrawn by setting the `withdrawAmount` to `type(uint256).max`:

```
if (deposited > newMax) {
    uint256 withdrawAmount = type(uint256).max;
    if (newMax != 0) {
        withdrawAmount = deposited - newMax;
    }
    IFarmStrategy(strategyAddress).withdraw(withdrawAmount);
```

Not all strategies can handle such a large withdrawal value. For example, in the `_withdraw` function of the `StrategyPendleLP` strategy, mathematical operations with the withdrawal amount may lead to an overflow:

```
/// @param amountToWithdraw Amount of asset to withdraw
function _withdraw(uint256 amountToWithdraw) internal override {
    // ...
    // Determine LP needed by proportion
    amountToWithdraw = (lpBalance * (((amountToWithdraw * 1e18) / farmBalance) + 1)) / 1e18;
```

Recommendation

We recommend refactoring the logic to allow explicit handling of `uint256.max` as an indication to withdraw all available funds.

Update

Fixed at `5dff05cacd65f89f725e37b5ec38d19ae7ee9c32`

Client's response

To raised issue is valid, but not quite as suggested. If the farm strategy has a different farmAsset and uses Uniswap, the Quoter will revert without a reason for sums the pool can't satisfy.

Limited withdraws to the amount available within FarmStrategy.withdraw()

M-05	Inability to commit earnings due to rounding errors When calculating `uncommittedLossPerc` in `HarvestableManager`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
HarvestableManager.sol	contract `HarvestableManager` > function `_repayLoan`	282

Description

In the `_repayLoan` function of the `HarvestableManager` contract, an underflow error could occur due to rounding during the calculation of `farmLoss`.

Consider a scenario where a farm loss is captured during the `harvest` process.

Assume that `loss` in the `_splitFarmLoss` function is greater than `vaultBorrow` but less than `harvestStorage.realUncommittedEarnings`. In this case:

- The percentage of `loss` relative to the total `harvestStorage.realUncommittedEarnings` is calculated. This percentage is then deducted from the `uncommittedEarnings` of each user when they call `updatePosition`.
- The entire `loss` amount is immediately deducted from `harvestStorage.realUncommittedEarnings`.

```
uncommittedLossPerc = (loss * 1e18) / harvestStorage.realUncommittedEarnings;
harvestStorage.realUncommittedEarnings -= loss;
return (vaultBorrows, uncommittedLossPerc, 0);
```

However, if `harvestStorage.realUncommittedEarnings` is large and `loss` is very small, rounding during division may result in the `uncommittedLossPerc` being calculated as 0.

In such a case, users would not account for any loss in their `uncommittedEarnings` when updating their positions, even though the loss has already been deducted by the protocol from `harvestStorage.realUncommittedEarnings`.

This may lead to a situation where the last users committing to this harvest snapshot would not be allocated the correct amount of `uncommittedEarnings`, and the `_repayLoan` function would encounter an underflow error:

```
uint256 realUncommittedEarnings = harvestStorage.realUncommittedEarnings;
// ...
realUncommittedEarnings -= commit.userHarvestUncommittedEarnings;
harvestStorage.realUncommittedEarnings = realUncommittedEarnings;
```

Recommendation

We recommend refactoring the logic to prevent scenarios where a loss deducted from the general balance exceeds the amount deducted from users' balances due to rounding.

Update

Fixed at `0c6ed94c27b57b88e7f71f8280461b7b1bcc276d`, `6ab784e78d21431e89853339eaa4da402dadf0e7`

Client's response

Updated to prevent a possible underflow when updating realUncommittedEarnings

M-06	No supply tokens minted for portion of `withdrawFee` after deduction in `VaultCoreV1`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
VaultCore.sol	contract `VaultCoreV1` > function `_applyWithdrawal`	346

Description

In the `_applyWithdrawal` function of the `VaultCoreV1` contract, the `withdrawFee` is distributed among all users by increasing the index. Thus, all users share the fee.

The index increase is calculated using the ratio of `supplyBalance - withdrawFee` to `supplyBalance`. This calculation considers the entire `supplyBalance` on the lender, including the portion remaining for the user after the fee deduction.

```
// Manually apply the fee to be distributed among all the users by increasing the index
uint256 balanceNow = supplyToken.storedTotalSupply();

uint256 indexIncrease = supplyToken.calcIndex(balanceNow - withdrawFee);
supplyToken.setInterestIndex(indexIncrease);
```

However, the user's index is updated immediately, meaning they do not participate in the distribution of the `withdrawFee`. As a result, tokens for the user's share of the fee are not minted, leaving a portion of the supply on the lender unallocated:

```
// Don't include user into the fee distribution
supplyToken.setBalance(account, maxWithdrawalAmount - (withdrawAmount + withdrawFee),
indexIncrease);
```

Recommendation

We recommend revisiting the logic for distributing `withdrawFee` to ensure that no unaccounted supply balance remains in the system after deducting the user's funds.

Update

Fixed at `5c74f71e3f7ddf4ad3bdfb1baf582b6102babcc8`

Client's response

Resolved by distributed portion of withdrawFee allocated to withdrawing user to all other users instead

3.3 WARNING

W-01	Farm loss at the moment of snapshot `SupplyLoss` is distributed proportionally to users' supply balance in `CommitMath`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
CommitMath.sol	contract `CommitMath` > function `_distributeWithdrawShortage`	328

Description

In the function `_distributeWithdrawShortage` of contract `CommitMath`, the user's share of the total `withdrawShortage` calculated during `snapshotSupplyLoss` is determined. This share is calculated for each user in proportion to their `supplyBalance` relative to the total `supplyBalanceAtSnapshot`:

```
withdrawShortage = Utils.divRoundingUp(
    supplyBalance * snapshot.withdrawShortage,
    snapshot.supplyBalanceAtSnapshot
);
```

In fact, `withdrawShortage` represents the farm loss obtained during `snapshotSupplyLoss` in the function `_withdrawVaultBorrows`:

```
// Consider the missing amount as a loss to allow for distribution amongst users
farmLoss = vaultBorrows - withdrawn;
```

Consider two users who deposited the same supply amount, but:

- The first user took no debt.
- The second user borrowed the maximum possible amount.

As a result, `activeAsset` will exist only for the first user, meaning only their available capital can be used for farming.

At the same time, when updating positions during the calculation of the `SupplyLoss` snapshot, both users will pay the same `withdrawShortage`.

Thus, in the event of farm profits, only the first user benefits, whereas in the case of farm losses, both users share the losses equally.

Recommendation

we recommend revisiting this logic and implementing a fairer distribution of the loss.

Update

Client's response

This is the behaviour as designed but there is a workaround to avoid the described effect.

Running `snapshotSupplyLoss` is permissioned and can be preceded with a harvest when needed. Running the harvest automatically however isn't desirable as it is only beneficial if there is a farmLoss at the time of the supplyLoss and there aren't one or more large user position(s) liquidatable.

W-02	Address validation confusion in `_validateBorrow` in `VaultCoreV1`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
VaultCore.sol	Contract `VaultCoreV1` > Function `_borrow`	200

Description

In the `_borrow` function of the `VaultCoreV1` contract, the `_validateBorrow` function is called for validation, where address verification is performed in `ingressControl`:

```
function _borrow(uint256 amount, address onBehalfOf, address receiver) internal {
    _validateBorrow(onBehalfOf, amount);

// ...

function _validateBorrow(address account, uint256 amount) internal {
    IIngress(ingressControl).validateBorrow(msg.sender, account, amount);

// ...

function validateBorrow(address borrower, address recipient, uint256 amount) external
override {

// ...
```

Within `ingressControl`, the `validateBorrow` function receives `msg.sender` as the `borrower` and `account` (which maps to `onBehalfOf`) as the `recipient`.

This logic assumes that the `onBehalfOf` user, the one incurring the debt, is validated as the recipient of the borrowed funds. However, the actual `receiver` specified in the `_borrow` function is not validated.

Currently, functions such as `borrow`, `depositAndBorrow`, and `borrowOnBehalfOf` call `_borrow` with `receiver` set to `msg.sender`. However, any future change in this behavior

could lead to vulnerabilities due to the lack of validation for the `receiver` address in `ingressControl`.

Recommendation

We recommend reviewing the logic and adding the receiver parameter where appropriate.

Update

Fixed at `209056935ccc8e30428931b919d86a2a2fea7b36`

Client's response

Refactored validation logic and added receiver check in `_validateBorrow`.

W-03	Inability to withdraw funds from a deactivated strategy in `FarmDispatcher`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	Contract `FarmDispatcher` > Function `deactivateStrategy`	173

Description

In the `deactivateStrategy` function of the `FarmDispatcher` contract, a strategy is deactivated even if it still holds funds. However, funds can only be withdrawn from active strategies.

As a result, reactivating the strategy becomes necessary in order to withdraw funds after it has been deactivated.

Moreover, while funds can be withdrawn during deactivation using the `emergencyDeactivateStrategy` function, this can only occur if the strategy is in `inEmergency` mode.

Recommendation

We recommend adding logic to allow funds to be withdrawn during strategy deactivation. While `emergencyDeactivateStrategy` provides a solution for withdrawing funds with emergency status, adding functionality for regular deactivation would prevent the need to reactivate strategies unnecessarily.

Update

Fixed at `78417ea2bffa0cc4cef1b6814423ce90ce39ae07`

W-04	Inaccurate `strategy.totalDeposit` calculation after withdrawal with losses in `FarmDispatcher`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
FarmDispatcher.sol	Contract `FarmDispatcher` > Function `_withdraw`	307

Description

In the `_withdraw` function of the `FarmDispatcher` contract, withdrawals from strategies are conducted in a loop. If losses occur within a strategy, the amount withdrawn might be less than the requested `toWithdraw`.

Although the withdrawal amount is smaller due to losses, the full `toWithdraw` value is subtracted from `strategy.totalDeposit`:

```
try IFarmStrategy(strategyAddr).withdraw(toWithdraw) returns (uint256 strategyWithdrawn) {
    // ...
    if (strategy.totalDeposit < toWithdraw) {
        availableLimit += strategy.totalDeposit;
        strategy.totalDeposit = 0;
    } else {
        availableLimit += toWithdraw;
        strategy.totalDeposit -= toWithdraw;
    }
```

For example, if `strategy.totalDeposit > toWithdraw` but the strategy incurs losses, all funds may still be withdrawn. In such cases, the actual balance in the strategy would drop to zero, yet `strategy.totalDeposit` and `availableLimit` may still indicate there are remaining funds.

Recommendation

We recommend revising the balance calculation logic for `strategy.totalDeposit`. Specifically, ensure that `strategy.totalDeposit` reflects the actual deposited balance

rather than being arbitrarily updated during withdrawals. This approach has already been implemented in the `setStrategyMax` function:

```
uint256 deposited = IFarmStrategy(strategyAddress).balance();
```

Update

Client's response

This is working as designed. strategy.totalDeposit() and strategy.balance() are intentionally different to prevent costly `dust` transactions being made into one or more strategies.

W-05	Inconsistent permissions for strategy functions in `FarmStrategy`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
FarmStrategy.sol	contract `FarmStrategy` > function `emergencyWithdraw`	85
FarmStrategy.sol	contract `FarmStrategy` > function `emergencySwap`	96

Description

Different roles are used to manage the functions in the specified locations. For example, the `emergencyWithdraw` function can only be executed by the owner, while the `emergencySwap` function is executed by the dispatcher role.

This creates a situation where a dispatcher cannot call the `emergencyDeactivateStrategy` function to deactivate the strategy and withdraw funds. To do so, the strategy must be set to the `inEmergency` mode, which can only be activated by the strategy owner.

At the same time, only the dispatcher can exit the strategy from the `inEmergency` mode, which the owner cannot influence.

Recommendation

We recommend considering the use of a unified role to manage the `inEmergency` mode to avoid inconsistent permissions while working with the contract functions.

Update

Fixed at `6430b1bcf44fa9eb679632e9be90a81a76691b9c`

Client's response

To simplify we have removed the `inEmergency` state and separated management:

- dispatcher: controls if funds should/shouldn't be deposited into the strategy (activating, deactivating, adjust cap, etc.)
- strategy: controls any recovery of funds in case of an emergency

W-06	Ability to successfully complete `repay` with `repayAmount = 0` in `VaultCoreV1`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
VaultCore.sol	contract `VaultCoreV1` > function `_repay`	360

Description

In the `_repay` function of the `VaultCoreV1` contract, it can be successfully called when the balance of `debtToken` tokens is zero for the `onBehalfOf` user. In this case, the `_repayUnchecked` function will return `0`, and the `repay` will complete successfully while emitting a `Repay` event.

As a result, it is possible to spam monitoring and logging systems with `Repay` events while only paying gas fees.

Recommendation

We recommend adding an error to prevent repayment of zero tokens.

Update

Fixed at `b9ea6f0534394662a966c217141ad6692c9c17ff`

Client's response

Prevents repayment of zero debt tokens

W-07	Insufficient Parameter Validation in `FarmDispatcher`, `MorphoVault`, `SkimStrategy`, `StrategyGenericPool`, `StrategyPendleBase`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	contract `FarmDispatcher` > function `_initialize`	33
FarmDispatcher.sol	contract `FarmDispatcher` > function `addStrategy`	121
StrategyGenericPool.sol	contract `StrategyGenericPool` > `constructor`	60
MorphoVault.sol	contract `MorphoVault` > function `setRewardAssets`	40
StrategyPendleBase.sol	contract `StrategyPendleBase` > `constructor`	59
StrategyPendleBase.sol	contract `StrategyPendleBase` > `constructor`	61
StrategyPendleBase.sol	contract `StrategyPendleBase` > `constructor`	62
StrategyPendleBase.sol	contract `StrategyPendleBase` > function `setRewardAssets`	71
SkimStrategy.sol	contract `SkimStrategy` > function `skim`	37

Description

Insufficient parameter validation is observed in the indicated locations:

- In the `skim` function within the `SkimStrategy` contract, no validation is performed for the `receiver` parameter. For example, it is possible to pass a zero address to receive tokens.
- In the `constructor` of the `StrategyPendleBase` contract:
  - It is possible to set a `market` that has already reached maturity and will cause the `market.isExpired` function to return `true`. In such cases, initializing the strategy with such a `market` becomes meaningless.
  - The `rewardAssets` array may contain duplicate addresses or zero addresses. This also applies to the `setRewardAssets` setter function.
  - The `slippage` variable is not validated and can take any value, including those greater than 100%, which would lead to an underflow during validation in the `_validateRate` function.

- In the `constructor` of the `MorphoVault` contract, the `rewardAssets` array may contain duplicate addresses or zero addresses. The same issue applies to the `setRewardAssets` setter function.
- In the `_initialize` function of the `FarmDispatcher` contract, parameters are not validated during contract initialization. For instance, it is possible to pass a zero address as the `admin`.
- In the `constructor` of the `StrategyGenericPool` contract, it is possible to set a `curvePool` that does not support the specified `farmAsset` for the strategy.
- In the `addStrategy` function of the `FarmDispatcher` contract, there is no validation of the strategy's address being added. For example, the strategy might have a different `farmDispatcher` set. In such cases, the strategy will fail the `onlyDispatcher` check for functions such as `deposit`, `withdraw`, and `emergencySwap`.

Recommendation

We recommend adding parameter validations in the specified locations.

Update

Fixed at `c59a9700247e2a24271b527204f1314b5c63ba14`

Client's response

These are partially addressed. Exceptions are:

- Where adding the validation would require looping over an array (which we believe to be unnecessary for admin functions)
- Checking curvePool if it supports the farmAsset - this may be added later

W-08	A portion of user funds may remain in the buffer, which can impact `farmLoss` in the case of a `SupplyLoss` in `FarmBufferDispatcher`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
FarmBufferDispatcher.sol	contract `FarmBufferDispatcher` > function `_withdraw`	41

Description

In the `_withdraw` function of the `FarmBufferDispatcher` contract, when withdrawing funds from strategies, the buffer is filled if it is not full and if there are sufficient funds available. In this case, only the amount remaining after filling the buffer is returned as the withdrawal result:

```
function _withdraw(uint256 amountRequested) internal override returns (uint256
amountWithdrawn) {
    uint256 bufferCapacity = farmBuffer.capacity();

    if (amountRequested > bufferCapacity || bufferCapacity >= balance()) {
        amountWithdrawn = super._withdraw(amountRequested + farmBuffer.capacityMissing());

        amountWithdrawn = _fillBuffer(amountWithdrawn);
```

However, when withdrawing from strategies, it is possible to withdraw less than requested. For example, this might occur:

- as a result of a loss in the strategy,
- due to the `inEmergency` mode,
- because of an error in the strategy that triggers a `revert`, which is caught by the `try-catch` structure in the dispatcher.

Despite this, the buffer will still be filled first, and only the remainder after filling the buffer will be returned as the withdrawal result.

This behavior means, for instance, that in the case of `farmLoss`, the loss will primarily be covered by the users' funds rather than the buffer.

Recommendation

We recommend reviewing the logic of buffer management. Instead of filling the buffer after withdrawing funds from strategies, funds should first be drawn from the buffer, and only then from the strategies.

Update

Client's response

This behaviour is as designed for the following reasons:

- The buffer is pre-filled, therefore we don't consider these user funds
- For this reason we prioritise the buffer over user withdraws/claims

It's probably easiest to look at a few different scenarios to outline the behaviour:

Case	A	B	C	D	E
Request can be fulfilled from buffer	Yes	No	Yes	No	Yes
Farm has funds to fully refill the buffer	Yes	Yes	Yes	No	No
Farm has funds to refill buffer & satisfy user request	Yes	Yes	No	No	No

If we model out the behaviour in each of these cases we get the following:

Item	A	B	C	D	E
size	100	100	100	100	100
requested	30	30	30	30	30
START					
capacityRemaining	50	20	50	20	90
dispatcher.balance()	150	120	20	0	0
strategy.balance()	200	200	70	70	0
RESULT			*	*	*
capacity	20	100	100	90	90
dispatcher.balance()	120	90	0	0	0
strategy.balance()	200	90	0	0	0

Item	A	B	C	D	E
received	30	30	20	0	0

[*] In the above, the withdraw in scenario C, D, and E will revert unless `dropThreshold` is set to 100% on the farm strategy (or a harvest is run to recognise the losses)

In normal behaviour, scenarios A and B are the commonly triggered scenarios. Although in some edge-cases (e.g. when a significant farmLoss is present), this can result in a 0 value withdraw from the farmDispatcher, this behaviour is unlikely due to:

- Operationally, the buffer would typically be much smaller than the deposits into the farm dispatcher. Equally, there are likely multiple strategies active, meaning that an issue with a single strategy is not likely to trigger this issue.
- Should a farmLoss happen, then the `dropThreshold` parameter will typically prevent the withdraw from occurring in the first place (at least until these losses can be formally recognised through a harvest).

As there is no identified risk of exploit, we have decided to not change this behaviour at this point.

W-09	Missing Check for actual withdrawal amount from strategy in `FarmDispatcher`
Severity	**WARNING**
Status	• NO ISSUE

Location

File	Location	Line
FarmDispatcher.sol	contract `FarmDispatcher` > function `setStrategyMax`	97

Description

In the `setStrategyMax` function of the `FarmDispatcher` contract, when a `newMax` value smaller than the current strategy balance is being set, a withdrawal is performed from the strategy. However, there is no verification to ensure that the intended amount has actually been withdrawn.

```
IFarmStrategy(strategyAddress).withdraw(withdrawAmount);
strategy.totalDeposit = newMax;
```

However, in certain cases, the actual amount withdrawn from the strategy may be less than requested.

For example, assuming that the strategy is in `inEmergency = true` mode, the call to `IFarmStrategy(strategyAddress).withdraw(withdrawAmount)` would not withdraw any funds but would not throw an error either. Thus, excess funds would remain in the strategy balance even though they should have been withdrawn.

Recommendation

We recommend adding a verification step to ensure that the requested funds have actually been withdrawn to prevent inconsistencies between the contract's tracked balances and the actual funds on the strategy.

Update

Client's response

This is working as designed. strategy.totalDeposit() and strategy.balance() are intentionally different to prevent costly `dust` transactions being made into one or more strategies.

3.4 INFO

I-01	Non-optimal gas usage in `LiquidatableManager`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
LiquidatableManager.sol	contract `LiquidatableManager` > function `liquidateUsers`	95

Description

In the function `liquidateUsers` of the `LiquidatableManager` contract, user liquidations are conducted in a loop. After this, the following check is performed:

```
if (
    liquidatedUsers < liquidatableStorage.minUsersToLiquidate &&
    totalRepayAmount < liquidatableStorage.minRepayAmount
) {
    revert LQ_V1_LIQUIDATION_CONSTRAINTS();
}
```

As a result, under certain conditions where only a small number of users in the system have unhealthy positions, liquidation cannot take place due to this condition, leading to unnecessary gas expenditure for the liquidator.

Recommendation

We recommend exploring optimization options or providing the liquidator with the ability to specify the values of `minUsersToLiquidate` and `minRepayAmount`.

Update

Fixed at `e4a982d6fc3215dec146c1f5d1419cb8f76483fc`

Client's response

We have removed the liquidation constraints

I-02	Missing validation that `amountToTransfer` is not less than `amount` in `LenderStrategy`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
LenderStrategy.sol	contract `LenderStrategy` > function `borrow`	102

Description

In the function `borrow` of the `LenderStrategy` contract, there is no validation to ensure that `amountToTransfer` is not less than the requested `amount`. At the same time, in `VaultCoreV1`, it is assumed that the requested `amount` is received.

Recommendation

We recommend adding a validation check to ensure that `amountToTransfer` is not less than the requested `amount`.

Update

Client's response

Check that the lender is actually sending us the requested borrow tokens. `9525525b755f7f0899a766cc5755055360dd99ba`

However, the fixes for this issue are being made in code that is not part of the current audit scope, so the status for this finding has been set to ACKNOWLEDGED.

I-03	Setter required or make `slippage` immutable in `StrategyPendleBase`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
StrategyPendleBase.sol	contract `StrategyPendleBase`	34

Description

In the `StrategyPendleBase` contract, there is a state variable `slippage`. However, it cannot be modified after being set once during contract deployment because there is no setter defined for it. At the same time, the variable is not immutable.

Recommendation

We recommend adding a setter for the `slippage` variable. Alternatively, consider defining it as an immutable variable.

Update

Fixed at `5fefd1dfe196b26392ab0f41f5bb50683536ade0`

Client's response

Setter added

I-04	Missing validation for `decrease != 0` in `FarmBuffer`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
FarmBuffer.sol	contract `FarmBuffer` > function `decreaseCapacity`	100

Description

In the function `decreaseCapacity` of the `FarmBuffer` contract, there is no validation to ensure that `decrease` is not equal to 0. This can result in an emitted event with a value of 0.

Recommendation

We recommend adding a validation check to ensure that `decrease != 0`.

Update

Client's response

As this is a permissioned function we feel adding this would only introduce unnecessary complexity

I-05	Unable to withdraw mistakenly sent tokens in `FarmBuffer`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
FarmBuffer.sol	contract `FarmBuffer`	33

Description

In the contract `FarmBuffer`, it is not possible to withdraw tokens that were mistakenly transferred to the contract balance.

Recommendation

We recommend adding a `withdraw` function to remove mistakenly sent tokens.

Update

Client's response

To limit complexity we don't plan to include this. We may consider in future versions.

I-06	Typo in `FarmBufferDispatcher`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
FarmBufferDispatcher.sol	contract `FarmBufferDispatcher`	15

Description

In the `FarmBufferDispatcher` contract, there is a typo in the following comment:

```
/** @notice Buffer for gas optimizaiton */
```

Recommendation

We recommend correcting the typo in the word "optimization."

Update

Fixed at `370591a07b0337a251b4134141bb61159f2f34ce`

I-07	Incorrect approval in `FarmBufferDispatcher`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
FarmBufferDispatcher.sol	contract `FarmBufferDispatcher` > function `_fillBuffer`	97

Description

In the `_fillBuffer` function of the `FarmBufferDispatcher` contract, approval is given for the full `amount`, but only the required amount is used, leaving the remaining balance under approval.

Recommendation

We recommend removing the approval after the operations are executed.

Update

Fixed at `3aea7b4d1f1e16cb9c3e29995617d11594b838b9`

I-08	Approval remains active if deposit fails in `FarmDispatcher`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	contract `FarmDispatcher` > function `_dispatch`	242

Description

In the `_dispatch` function of the `FarmDispatcher` contract, approval for the strategy remains active if the deposit operation fails.

Recommendation

We recommend removing the approval from the strategy in the case of deposit failure.

Update

Fixed at `0597cc58559912dfb65bf01a4ba83924efd6b263`

Client's response

Resets approval after dispatch

I-09	It is possible to pass `strategyAddress == 0` in `FarmDispatcher`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
FarmDispatcher.sol	contract `FarmDispatcher` > function `setStrategyMax`	89

Description

In the `setStrategyMax` function of the `FarmDispatcher` contract, it is possible to pass a `strategyAddress` with the value `address(0)`.

Recommendation

We recommend adding a validation to ensure that `strategyAddress != 0`.

Update

Fixed at `70c3a738b3de8e263c6bf36e4b104556103ed46c`

I-10	No Setter for `nonSkimAssets` in `SkimStrategy`
Severity	**INFO**
Status	• NO ISSUE

Location

File	Location	Line
SkimStrategy.sol	contract `SkimStrategy`	36

Description

In the `SkimStrategy` contract, there is no setter function for the `nonSkimAssets` variable.

Recommendation

We recommend adding a setter for the `nonSkimAssets` variable.

Update

Client's response

For security reasons we didn't include a setter, any tokens specified as non-skimmable will remain so. If new nonSkimAssets are required to be added/removed we can relaunch the strategy.

I-11	Typo in `toWithdaw` in `SupplyLossManager`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
SupplyLossManager.sol	contract `SupplyLossManager` > function `_withdrawVaultBorrows`	172

Description

In the `_withdrawVaultBorrows` function of the `SupplyLossManager` contract, there is a typo in the word `toWithdaw`.

Recommendation

We recommend correcting the typo to `toWithdraw`.

Update

Fixed at `3a197bf001c964bcd7c6749a298cbd7e2c993551`

I-12	`farmDispatcher` should be used instead of rewardsRecipient in `FarmStrategy`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
FarmStrategy.sol	contract `FarmStrategy` > function `recogniseRewardsInBase`	123

Description

In the function `recogniseRewardsInBase` of the `FarmStrategy` contract, accumulated rewards are sent to a separate address, `rewardsRecipient`, instead of the `farmDispatcher` address.

Recommendation

We recommend setting the specific address of `farmDispatcher` as the recipient of rewards.

Update

Client's response

We are using a common pattern between the lenderStrategies and farmStrategies, by maintaining this pattern we ensure consistency.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

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

1. Project Architecture Review

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

2. Vulnerability Assessment

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

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

3. Security Model Evaluation

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

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

4. Cross-Verification by Multiple Auditors

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

5. Report Consolidation

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

6. Reaudit of Revised Submissions

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

7. Final Audit Report Publication

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

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

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

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

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

4.2.1 Rating Criteria

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

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

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

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

4.3.2 Status Level Reference

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

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

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 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