GUARDIAN
AUDITS

SMART CONTRACT SECURITY AUDIT OF

Ambit

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Kiki, 0xKato, Kristian Apostolov, 0xScourgedev

Client Firm Ambit Finance

Final Report Date December 6, 2023

Audit Summary

Ambit Finance engaged Guardian to review the security of its borrowing and lending platform. From the 20th of November to the 6th of December, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **BNB Smart Chain**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/AmbitPoCs

Table of Contents

Project Information

Smart Contract Risk Assessment

Addendum

Project Overview

Project Summary

Project Name	Ambit Finance
Language	Solidity
Codebase	https://github.com/ambitfi/ambitfi-contracts
Commit(s)	89a4abaa35df3e34b472c5128fe2e960a7a31a0d

Audit Summary

Delivery Date	December 6, 2023
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	3	0	0	0	0	3
● High	5	0	0	1	0	4
● Medium	16	0	0	5	0	11
● Low	40	0	0	16	0	24

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
PAE	PortfolioAssetExtensions.sol	dd355c5b77db2bda3fd27929a6dd2295a0ca906b
PH	PortfolioHooks.sol	ab2e6707d07ab6dedd4ae0f8ab47b0b6abd5727f
NMLZ	Normalizer.sol	607793cf512286d43cea084a2c5c6025b9f51c2c
FEE	Fees.sol	3439cb8248914efdc4f81828d4822642ec452df6
ERR	Errors.sol	8a9646adcc0140c0c35289db47b37c961b8b74a1
PMATH	PercentageMath.sol	68729250f86124dfc0e64a1ef6c7aec084de4560
IMATH	InterestMath.sol	8af842b520ef68c7be7ff0e3f09de5ab985681eb
UMATH	USDMath.sol	3f62cb6aae51225cc2ce39472b82aa2ab24f34f3
EPCHL	EpochLib.sol	3f0ad5f39e73b3b10d3f4db52ed0f3bac0f8779e
ARE	AddressRegistryExtensions.sol	154520ed6d96cb2837ba8e43645dcb90b3a6eafb
REL	RewardEpochLib.sol	fe61ea8ce3412abd4e3a18d642a402d1d9d50950
TRH	TokenRewardHooks.sol	476643d7aa43db0929fb0080257d699b9d408d78
ADRG	AddressRegistry.sol	d5a72cf26b9480ab22ebfa433511b18e3084a41c
ASRG	AssetStorage.sol	41d75259266cb1d75d8a293c1b3f131073acc5d8
ADAC	AdminAccessControl.sol	cecf76e82542a999c804acc50ca355ee0c5ade4b
ACL	AccessControlList.sol	a2f8602f74ae86d7de47bb2d379189e59b77a4f1
AUAC	AuthorizedAccessControl.sol	3f667aba5b038be3a0f642c65184fe7c3e7da66f
DVMA	DepositorVaultMarketplaceAdapter.sol	839d37033c2f02b9b497cfc1783323b00dd42ade
MKTV	MarketplaceVendor.sol	d2fa0de8e20730bad1467a3603a8d8d4c8851141
SMMA	SpotMarketMarketplaceAdapter.sol	e6b43c9f4da3db308bc7fe4efde9a33b44bf0669

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
MKTP	MarketplacePurchaser.sol	6e4ea5a2201163a31669fb0a00aaf18ea91107cd
FPO	FallbackPriceOracle.sol	bfb1e4835c03a9e1921db896dc89fe0959fe17dc
DVTPO	DepositorVaultTokenPriceOracle.sol	dd337db7ce837b5ceb8a3dcbecc896974ac46a18
CAPO	ChainlinkAggregatorPriceOracle.sol	ff3f15503858a7ff87d7c726dd6ca32fba4aebc2
UNIPO	UniswapV2PriceOracle.sol	e954dad0fc863696138f419610863f8f0b67f7ae
PAUSE	Pausable.sol	995036d3ed4894ed0c9efcbad7ebf7542faa48af
MG	Migratable.sol	0f95957b34ee715c0482ec4c8f2c8447fe2a2d5f
SWEEP	Sweepable.sol	fa2d70018ecd0167aec108a5f368e729fd6e8fcf
BM	BoostModule.sol	1a8fe17c7ef48e96bad195c2fe08a8be062ea222
LOTYL	LoyaltyLib.sol	2c0a2a3011ad14618232f3617230f1da471cd2a6
LOTYS	LoyaltyStorage.sol	5ce12bb41316eec02454fac27462130ff05baec2
FLBM	FirstLoanBoostModule.sol	91c4c7c4d633e1ffcf72b56f8c74b54add457d7d
LOTY	Loyalty.sol	142852bc6d2fa7a588112108504a0a8b162f254d
LOTYH	LoyaltyHooks.sol	d32714d89ab4fd31a7b02c01c4e37a20116a9b25
GOV	Governor.sol	51f01dd3a41eee439d37da3debd82575ddb4b015
EXE	Executable.sol	9fda5e700e81c238f1b959776dbd14aae8809289
TRSY	Treasury.sol	7b8ea901d40aaabba9cee554e7d6e19e17693cd9
PORTS	PortfolioStorage.sol	be95f55ccd155c16ddc6b2d9e12f35fb3cd5686d
CM	CustodianMigrator.sol	596cbdfbee6a3123beee0f600c465d24e6c6a303
CTDN	Custodian.sol	9aeab147612cc97c289f769192878e0210eac0b4

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
PTFLO	Portfolio.sol	e8cb51655cf0b803986d2cb512025c20fa3f713b
YBC	YieldBearingCustodian.sol	d5ce9ac5f36bc6377add4eb85cf61dc124d7de02
BC	BaseCustodian.sol	e448c349aaf174fc693057b06ca9a24044ae8c4d
LDTR	Liquidator.sol	60eef99d467120b450e3989202c6df8785c8a9a9
MKTS	MarketStorage.sol	948326d873f54814b0b127eb482daffa216c4a9d
FIRM	FixedInterestRateModel.sol	f9b1322ff9f862d7089ba1371c1998ade125620f
DIRM	DynamicInterestRateModel.sol	d3269e75f9607dc1b086058ab0418a5605ceceda
DMOD	DiscountModel.sol	2a9963e0ca5ffb2d8f700a12d013cb246db08e05
MKT	Market.sol	a8ed84717f2280cb1b0079c9908c00795ddf8f63
ML	MarketLiquidation.sol	d1d7972cab1cdba6b6c181453574ec87f618d647
VEST	TokenVesting.sol	bdcf611870493cd2a002c9e3d10fce8da3c6989f
LDYV	LinearDistributedYieldVault.sol	6adb8b9d5d25582686b71c26dac177c612ad188f
DVT	DepositorVaultToken.sol	d238abcda2f74db5357163cdae17ff35defa8099
YVLT	YieldVault.sol	b04411e7b6eca19449de094471f1c2e64a316dac
SNAP	SnapshotLib.sol	d4fc981e2182db2f9592934af63db4c4474592c3
DVM	DepositorVaultMigrator.sol	346ae1606b06d5edf019cefe5e8b52bcf078a71c
DVS	DepositorVaultStorage.sol	9617c4862b8560a32222a3869e44d2610ff62b57
VLT	Vault.sol	c23250c37b997e886c7110cf0a1879803511b749
FL	FlashLender.sol	013755409df14be8e859720ba9ada1efd7fd343a
DVLT	DepositorVault.sol	206e4ec73286550a5a7fd0056696cf169e2748cb

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Ambit's Borrowing and Lending platform, fuzz-testing with Echidna was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 15,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Run Count
GENERAL-01	Does not revert with underflow/overflow	✅	❌	-
HS-01	Health score is increasing after supply	✅	✅	7,000,000+
HS-02	Health score is decreasing after withdraw	✅	✅	7,000,000+
HS-03	Health score is decreasing after borrow	✅	✅	7,000,000+
HS-04	Health score is increasing after repay	✅	✅	7,000,000+
HS-05	Health score is increasing, or 0 after successful liquidation	✅	❌	-
HS-06	Health score is decreasing after borrow	✅	✅	7,000,000+
HS-07	Health score is above UNHEALTHY_HEALTH_SCORE_THRESHOLD after borrow	✅	✅	7,000,000+
HS-08	Health score is below UNHEALTHY_HEALTH_SCORE_THRESHOLD before successful liquidation	✅	✅	7,000,000+
HS-09	Liquidation always succeeds when health score is below UNHEALTHY_HEALTH_SCORE_THRESHOLD before liquidation	✅	❌	-

Invariants Assessed

ID	Description	Tested	Passed	Run Count
PORT-01	User can not withdraw more of one token than supplied	✅	✅	7,000,000+
LIAB-01	Liabilities are decreasing after liquidation	✅	✅	7,000,000+
LIAB-02	Liabilities of a user is increasing the exact amount that was borrowed	✅	✅	7,000,000+
LIAB-03	Liabilities of a user is decreasing after getting liquidated	✅	✅	7,000,000+
LIAB-04	Liquidation always fails if a user's liabilities is 0	✅	✅	7,000,000+
LOYAL-01	Total loyalty points is equal to the sum of all loyalty points of all users	✅	❌	-
LOYAL-02	The exact burned amount is added to the user balance2 after burn	✅	✅	5,000,000+
LOYAL-03	Claiming points sets getClaimablePoints to 0	✅	✅	5,000,000+
LOYAL-04	Loyalty boost can only be claimed once	✅	✅	5,000,000+
LOYAL-05	Loyalty points for a user should decrease or be equal after withdrawing Ambit	✅	✅	5,000,000+
LOYAL-06	Sum of pending, vesting, accrued loyalty points for a user should increase after supplying Ambit	✅	✅	5,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Run Count
REWARD-01	Rewards claimed from claim must be equal to the points predicted by getClaimableRewards	✅	✅	5,000,000+
VLTREV-01	previewRedeem() does not revert for reasonable values	✅	✅	15,000,000+
VLTREV-02	previewWithdraw() does not revert for reasonable values	✅	✅	15,000,000+
VLTREV-03	previewDeposit() does not revert for reasonable values	✅	✅	15,000,000+
VLTREV-04	getTotalAssets() never reverts	✅	✅	15,000,000+
VLTACCG-01	Redeem must deduct shares from user	✅	✅	15,000,000+
VLTACCG-02	Redeem must credit underlying asset to user	✅	✅	15,000,000+
VLTACCG-03	Redeem must credit greater than or equal to the number of underlying asset predicted by previewRedeem	✅	✅	15,000,000+
VLTACCG-04	Withdraw must deduct shares from user	✅	✅	15,000,000+
VLTACCG-05	Withdraw must credit underlying asset to the user	✅	✅	15,000,000+
VLTACCG-06	Withdraw must deduct less than or equal to the number of shares predicted by previewWithdraw	✅	✅	15,000,000+

Invariants Assessed

ID	Description	Tested	Passed	Run Count
VLTACCG-07	Deposit must deduct underlying asset from user	✅	✅	15,000,000+
VLTACCG-08	Deposit must credit shares to user	✅	✅	15,000,000+
VLTACCG-09	Deposit must mint greater than or equal to the number of shares predicted by previewDeposit	✅	✅	15,000,000+
VLTFREE-01	Underlying asset is never seen as withdrawn for free using previewRedeem	✅	✅	15,000,000+
VLTFREE-02	Underlying asset is never seen as withdrawn for free using previewWithdraw	✅	✅	15,000,000+
VLTFREE-03	Shares are never seen as minted for free using previewDeposit	✅	✅	15,000,000+
VLTFREE-04	Underlying asset is never withdrawn for free using withdraw	✅	✅	15,000,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
SNAP-1	Cardinality Errantly Incremented	Logical Error	● Critical	Resolved
LOTY-1	All Loyalty Rewards Can Be Stolen	Logical Error	● Critical	Resolved
LOTY-2	Total Points Does Not Match Sum Of User Points	Logical Error	● Critical	Resolved
LDTR-1	Small Positions Backed By The Vault Token Cause Liquidations To Revert	Logical Error	● High	Resolved
GLOBAL-1	Snapshot System Prone To Instant Balance Change	Protocol Manipulation	● High	Resolved
LOTY-3	Calling claimBoost Before Claiming Points Breaks The User's Boost	Logical Error	● High	Resolved
ML-1	Health Score Decreases After Liquidation	Logical Error	● High	Resolved
GLOBAL-2	USDT Treated as One Dollar	Logical Error	● High	Acknowledged
ML-2	findLargestPosition Uses Ordinary Price Instead Of Discounted	Logical Error	● Medium	Acknowledged
TRH-1	Stored Custodian Becomes Invalid Upon Migration	Migration	● Medium	Resolved
TRH-2	The Balance Of Any Disabled Token Can Get Hijacked	Access Control	● Medium	Resolved
ML-3	User Loses More Collateral Than Necessary	Logical Error	● Medium	Acknowledged
ML- 4	findLargestPosition Exposes Liquidator to Unnecessary Slippage	Logical Error	● Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
MKT-1	User Can Avoid Accrued Interest	Protocol Manipulation	● Medium	Resolved
MKT-2	MarketState Not Updated Prior To Calling pause/unpause	Logical Error	● Medium	Resolved
LDTR-2	Possible Lack Of Incentive For Liquidations	Incentives	● Medium	Resolved
MKT-3	discountModel Updated Before Liabilities Accrue	Logical Error	● Medium	Resolved
TRH-3	Token Rewards DoS	DoS	● Medium	Resolved
TRH-4	Rewards During A Pause For A Reward Token Can Be Accrued	Logical Error	● Medium	Resolved
TRH-5	Invalid Result Returned From getClaimableRewards	Logical Error	● Medium	Resolved
SNAP-2	Untruncated Timestamps Are Unmatchable	Logical Error	● Medium	Resolved
YBC-1	Invalid totalAssets Validation	Logical Error	● Medium	Resolved
MKTV-1	Deleveraging Does Not Consider The Flash Fee	Configuration	● Medium	Acknowledged
LDTR-3	Liquidations Prevented By Slippage	Protocol Manipulation	● Medium	Acknowledged
LOTY-4	User Points Not Getting Claimed In accruePoints Loses Them Rewards	Logical Error	● Low	Acknowledged
MKT-4	Discounts Do Not Apply Without An Update To Liabilities	Logical Error	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
PTFLO-1	User Can Withdraw And Supply In The Same Block	Logical Error	● Low	Acknowledged
ML-5	User's Discount Not Considered On Liquidation	Logical Error	● Low	Acknowledged
MKT-5	Discounts Favor Borrowers With A Higher Liabilities/Principal Ratio	Logical Error	● Low	Resolved
UNIPO-1	Uniswap Oracle Manipulation	Oracle Manipulation	● Low	Resolved
LDTR- 4	MIN_SLIPPAGE_ALLOWED is never used	Superfluous Code	● Low	Resolved
PTFLO-2	Portfolio Hooks Prone To Reentrancy	Reentrancy	● Low	Acknowledged
DIRM-1	Current Market Liabilities Used For Utilization	Logical Error	● Low	Acknowledged
LDTR-5	Liquidations Halted	Warning	● Low	Acknowledged
MKT-6	Users Are Charged Interest On Fee Amounts	Unexpected Behavior	● Low	Acknowledged
LOTY-5	Superfluous Pending Rewards Calculation	Superfluous Code	● Low	Resolved
GLOBAL-3	Interest Fee Design	Protocol Design	● Low	Resolved
MKT-7	Typo	Typo	● Low	Resolved
TRH-6	Donations Lost When totalShares Is 0	Lost Rewards	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
ML-6	Blacklisted Accounts May Avoid Liquidation	Blacklist	● Low	Acknowledged
VEST-1	Typo	Typo	● Low	Resolved
DVLT-1	No Basis Point Validation For setBorrowLimit	Validation	● Low	Resolved
TRH-7	Redundant Accruals	Optimization	● Low	Resolved
TRH-8	Superfluous tokenReward.index Update	Superfluous Code	● Low	Resolved
YBC-2	Lacking Migration Logic	Configuration	● Low	Resolved
YBC-3	Lack Of Donation Fee In The YieldBearingCustodian	Protocol Fees	● Low	Acknowledged
DVM-1	Depositor Vault Key Re-computed	Best Practices	● Low	Resolved
DVM-2	Redundant Inheritance	Superfluous Code	● Low	Resolved
YVLT-1	Invalid 0 Price Returned From getExchangeRate	Unexpected Behavior	● Low	Resolved
SNAP-3	Unnecessary Truncate Call	Optimization	● Low	Resolved
ML- 7	Users can be Liquidated for More than Needed	Documentation	● Low	Acknowledged
SNAP-4	Unexpected Found Value Returned	Unexpected Behavior	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
DMOD-1	Points Discount Extremely Low	Logical Error	● Low	Resolved
MKTP-1	Late Validation	Validation	● Low	Resolved
DVLT-2	safeTransferFrom Should Occur Before Updates	Reentrancy	● Low	Resolved
SMMA-1	Invalid UniswapV2 Expiration Timestamp	Logical Error	● Low	Resolved
DMOD-2	Precision Loss Can Lead to Loss of Loyalty Points	Precision	● Low	Resolved
DMOD-3	Incorrect Integer Casting	Logical Error	● Low	Resolved
LDYV-1	Wrong Event Emission	Logical Error	● Low	Resolved
LDYV-2	Vault Distributes Yield Even With No Shares Present	Logical Error	● Low	Acknowledged
DVLT-3	Vault Migration Sets The First Snapshot With The Newest Values	Logical Error	● Low	Acknowledged
ML-8	Very Small Positions In <8 Tokens Cannot Be Liquidated	Logical Error	● Low	Acknowledged
MKT-8	Incorrect Order Of Operations	Optimization	● Low	Resolved
GLOBAL-4	Disabling Hooks May Lead To Accounting Inaccuracies	Logical Error	● Low	Resolved

SNAP-1 | Cardinality Errantly Incremented

Category	Severity	Location	Status
Logical Error	● Critical	SnapshotLib.sol: 49	Resolved

Description

In the write function, if the cardinality is less than the size then the cardinality is always incremented, regardless of if a new index was written to or not.

This perturbs the checking of the oldest snapshot on line 68 in the find function as well as everything in the search function.

Recommendation

Only increment the cardinality when a new index is written to in the write function.

Resolution

Ambit Team: The issue was resolved in commit ac233d2.

LOTY-1 | All Loyalty Rewards Can Be Stolen

Category	Severity	Location	Status
Logical Error	● Critical	Loyalty.sol	Resolved

Description PoC

The functions supply() and withdraw() call accrueRewards() before mutating the points balance of a user. burnTokens() does not which allows someone to claim an abnormally large portion of the rewards by abusing their rewardsIndex and an artificially high balance of points that did not get accrued when incremented.

This allows such a user to burn AMBT just before they withdraw and receive much more rewards than they should. This will also brick the reward claim process for other users as the reward token balance of the contract will not be enough to cover for their rewards.

Recommendation

Call accrueRewards() before mutating the user's balance2 in burnTokens().

Resolution

Ambit Team: The issue was resolved in commit 277fba3.

LOTY-2 | Total Points Does Not Match Sum Of User Points

Category	Severity	Location	Status
Logical Error	● Critical	Loyalty.sol: 159	Resolved

Description PoC

When a user burns their Ambit, they are immediately credited with points that experience a 5x multiplier. However, if a user burns less than 1e7 of an Ambit, the total.points will not increment: total.points += LoyaltyLib.boostMul(amount.toUint64(), LoyaltyLib.BURN_BOOST);

This is because in the boostMul function, the returned point value will be truncated to 0: points / POINT_DENOMINATOR * boost / DEFAULT_BOOST; A user could maliciously take advantage of this by initially burning more than 1e7 Ambit. This will be reflected in both the total and the user's point balances as there will not be truncation.

The user can then burn less than 1e7 Ambit which will not be reflected in the total of the Loyalty due to the truncation, but it will be reflected in the point balance of the user because the user's balance2 is already above 1e7: points.balance2 / POINT_DENOMINATOR * BURN_BOOST / DEFAULT_BOOST;

Because the total points are less than needed, the tokenReward.index will be larger which will inflate the rewards to be dispersed: tokenReward.index += ((amount * REWARDS_MULTIPLIER) / loyalty.getTotalPoints()).toUint128();

Users will not be able to claim their rewards as more reward tokens are attempted to be sent than actually exist in the contract. This can be repeatedly done by malicious users to widen the spread between the total points and sum of all users' points.

Recommendation

In function burnTokens, remove the user's prior points from the total and add their new points with the boost similar to function updateBoost

Similarly adjust functions withdraw and claimPoints to avoid the discrepancy as well.

Resolution

Ambit Team: The issue was resolved in commits a414931 and d4b3fa3.

LDTR-1 | Small Positions Backed By The Vault Token Cause Liquidations To Revert

Category	Severity	Location	Status
Logical Error	● High	Liquidator.sol: 98	Resolved

Description

In the liquidateVaultToken function it is assumed that the totalAmount is always available for withdrawal from the vault since in many cases it would have been previously repaid to the vault in the MarketLiquidation.settleLiquidation function.

However in the case where all funds are actively being borrowed from the vault and a small account is liquidated while the totalAmount is greater than the liabilities, there is not enough USDT to redeem from the vault and the liquidation will revert on line 397 in the DepositorVault.withdrawInternal function.

Recommendation

Ensure the totalAmount is capped at the liabilities earlier on in the liquidation logic, this way there is always guaranteed to be enough USDT to redeem from the vault in the event that the user's collateral is the vault token.

Additionally, do not allow withdrawals from the DepositorVault to go below the available balance returned by getAvailableBalance. This way it is nontrivial to enter a scenario where there is no USDT left in the vault.

Resolution

Ambit Team: The issue was resolved in commit a9d6785.

GLOBAL-1 | Snapshot System Prone To Instant Balance Change

Category	Severity	Location	Status
Protocol Manipulation	● High	Global	Resolved

Description

The snapshot system was implemented in Ambit V2 to prevent the manipulation of the utilization rate through instantaneous balance changes in the Depositor Vault. However, the system only examines the latest snapshot, which can be trivially updated because the function `takeSnapshot` is `public`.

A user can take a snapshot right after depositing in the vault, and now the snapshot will reflect the current balance. Consequently, a user can still instantly manipulate the utilization rate at will through their deposits and withdrawals and take a snapshot right after.

Recommendation

Modify the visibility of function of `takeSnapshot` to `private`. Furthermore, take a snapshot upon borrowing and repaying as the total liabilities are modified.

Consider taking a weighted average of the assets available across the past 4-10 snapshots so it is less prone to instantaneous manipulation. Also, consider utilizing a minimum `deposit` and `borrow` amount as well to limit a user from trivially creating many snapshots and pushing the 4-10 snapshots back.

Resolution

Ambit Team: The issue was resolved in commit 8530b71.

LOTY-3 | Calling claimBoost Before Claiming Points Breaks The User's Boost

Category	Severity	Location	Status
Logical Error	● High	Loyalty.sol	Resolved

Description PoC

The UserPoints.boost is supposed to be at 10 (equating to 1x) but is 0 initially. When claimPoints() gets called after waiting for 2 EPOCHs the UserPoints.boost is set in the following snippet:
user.boost = user.boostOrDefault(); // @audit gets set from 0 to 10 here

The user can make their boost larger by calling claimBoost() if one is available to them. The current implementation of the system has a FirstLoanBoostModule which gives users an additional 0.1x on top of their 1x boost. The requirement for it is that they have borrowed more than 0 from the protocol and that they have a positive point balance through UserPoints.total().

The issue arises if the user burns AMBT with burnTokens() before calling claimPoints() to set their initial UserPoints.boost to 10 and then calls claimBoost(). This would set their boost to 0.1x instead of to 1x due to the following LoC:
user.boost = Math.min(user.boost + boost, LoyaltyLib.MAX_BOOST).toUint64();

Here UserPoints.boost is not yet set to the default boost, hence the user's boost for their balance1 becomes 10x smaller than it should be and causes them to lose out on 90% of the rewards they should be receiving.

Recommendation

Consider implementing the following changes across the Loyalty.sol file:

1. Check whether the UserPoints.balance1 specifically is > 0 for boost eligibility.

2. Set the UserPoints.boost to UserPoints.boostOrDefault() in burnTokens() and supply().

Resolution

Ambit Team: The issue was resolved in commit 8c37f44.

ML-1 | Health Score Decreases After Liquidation

Category	Severity	Location	Status
Logical Error	● High	MarketLiquidation.sol	Resolved

Description PoC

Because each token has a different LTV, a liquidation in a specific token produces a change in borrowing power that is not proportionate to the change in liabilities. As a result, it is easily possible for a position to have a lower health score (become unhealthier) post-liquidation.

The position would require more than 1 liquidation at 50% of the liabilities until the health score increased. Consequently, bad debt can stay in the system for a prolonged period and put the protocol at risk of supporting an insolvent position.

- 1000 USDT in liabilities
- 6 ETH each at $100 for 100% LTV ($600 borrowing power)
- 1 BTC each at $400 for 50% LTV ($200 borrowing power)
- Health Factor: ($600 + ($400 * 0.5)) / $1000 = **0.80**

Because ETH is the largest position in the portfolio, the ETH will be liquidated.

- 50% of 1000 USDT (liabilities) = 500 USDT
- Proportionate supply = 500 USDT / $100 (ETH price) = 5 ETH
- 500 USDT in liabilities
- 1 ETH in the portfolio ($100 borrowing power)
- 1 BTC in the portfolio ($200 borrowing power)
- New Health Factor: ($100 + ($400 * 0.5)) / $500 = $300 / $500 = **0.60**

Recommendation

Prior to liquidating a position, simulate the health factor post-liquidation. Carefully select the asset to liquidate which will ultimately increase the health factor. Furthermore, consider supporting full liability liquidations.

Although this may add complexity as multiple swaps in the portfolio may be needed due to the different supply tokens, it will ensure the borrow position is healthier post-liquidation.

Resolution

Ambit Team: The issue was resolved in commit 2b4c71d.

GLOBAL-2 | USDT Treated as One Dollar

Category	Severity	Location	Status
Logical Error	● High	Global	Acknowledged

Description

During the liquidation of a position involving a different ERC-20 token, a swap occurs. In this swap, the minAmountOut is calculated to ensure a minimum amount of tokens is received. However, the estimateSellAmount function calculates the minAmountOut based on the notional value, by multiplying the amount (amount1) by the price (amount2):

```
uint256 minAmountOut = amount1.mulDiv(amount2, 10 ** decimals);
```

Changes in the asset's price can lead to minAmountOut being either larger or smaller than the input amount. If the price rises, the minAmountOut may exceed the input, causing the swap and liquidation attempts to fail. Conversely, if the price falls, minAmountOut may be much lower than intended, enabling swaps to exceed the intended max slippage.

Other areas where USDT is assumed to be $1 is the liabilities. Because the liabilities are used in calculations such as an account's health factor, an account may appear unhealthy when it is not and vice versa.

Recommendation

Modify the calculation for minAmountOut in the estimateSellAmount function to be based on the expected token amount rather than the notional value. This adjustment ensures more accurate and reliable swaps during liquidation. Furthermore, use a USDT price feed to retrieve the market price.

Resolution

Ambit Team: Because we are a lending protocol that only lends out a single asset, the liabilties and borrow limit are denominated in that base asset, in this case USDT. Additionally, all price oracles are based on USD.

ML-2 | findLargestPosition Uses Ordinary Price Instead Of Discounted

Category	Severity	Location	Status
Logical Error	● Medium	MarketLiquidation.sol: 284	Acknowledged

Description

`findLargestPosition` finds the largest position in a user's portfolio and returns it to be liquidated. The issue arises due to it using the ordinary prices from the oracle instead of the ones with applied discounts when calculating position size.

This will cause some positions, which are normally of higher value but have a higher discount to still be selected over ones with a lower value and a lower discount that equates to them being worth more when liquidating: `(, USD[] memory totals) = portfolio.getPortfolioValue(portfolioAssets);`

Recommendation

Consider using the discounted prices to find the position that will liquidate the most amount of assets.

Resolution

Ambit Team: We have discussed this in the past, however, using the position that has the largest USD value (ignoring the discount) is preferrable as it allows for the greater portion of liabilities to be repaid.

TRH-1 | Stored Custodian Becomes Invalid Upon Migration

Category	Severity	Location	Status
Migration	● Medium	TokenRewardHooks.sol: 60	Resolved

Description

The Custodian is kept as a separate, immutable variable inside of the TokenRewardHooks instead of `_asset.custodian` being used. Upon Custodian migration through the `CustodianMigrator`, the address of the Custodian will change and `TokenRewardHooks` will continue to reference the old Custodian. This necessitates a `TokenRewardHooks` re-deploy as `_custodian` cannot be reset.

Recommendation

Consider using the Custodian directly on the `_asset` instead of setting `_custodian`.

Resolution

Ambit Team: The issue was resolved in commit 310287d.

TRH-2 | The Balance Of Any Disabled Token Can Get Hijacked

Category	Severity	Location	Status
Access Control	● Medium	TokenRewardHooks.sol	Resolved

Description

Any asset, that gets disabled will have it's balance hijacked due to a lack of access control on claim(). Function claim() calls accrue(_tokenRewards[token]), which is a private function only also called in accrue(). This function updates the current index of each epoch on the token based on the totalSupply of custodian shares at the moment.

Due to the asset not being in the _rewardTokens anymore, it does not get its index updated upon a user change.Consequently claim() is prone to a Portfolio share inflation attack that steals the whole balance of the said reward token.

This can be achieved by depositing a large amount of the asset, calling claim(), which updates the accrued for that token based on the now inflated balance of shares and then transfers an inflated portion to the user.

Recommendation

Put only active token access control on claim().

Resolution

Ambit Team: The issue was resolved in commit 115e128.

ML-3 | User Loses More Collateral Than Necessary

Category	Severity	Location	Status
Logical Error	● Medium	MarketLiquidation.sol: 203	Acknowledged

Description

During a liquidation, the total amount to be liquidated and the amount of supply to withdraw from the user's portfolio to the Liquidator is calculated in function `calculateLiquidation()`

With those parameters the liquidation is settled in function `settleLiquidation()`, where the total amount being repaid is adjusted to be no more than the user's liabilities:
`uint256 repayAmount = Math.min(context.totalAmount, context.liabilities);`

Consequently, the amount to repay can be dramatically decreased yet the amount withdrawn from the user's supply is still the originally calculated supply. Although the the excess of the user's liabilities is refunded, the supply was valued at the discounted price so the user has more supply withdrawn from their portfolio than necessary.

This results in an extra "liquidation fee" causing loss of funds for users.

Recommendation

Consider adjusting `context.position` by the discounted price to accurately reflect the amount being repaid. Otherwise, clearly document this behavior to users.

Resolution

Ambit Team: Yes this is correct, however, it will only occurr for accounts that have liabilities less than the smallAccountThreshold (which would probably be set to somewhere between 250-500).

ML-4 | findLargestPosition Exposes Liquidator to Unnecessary Slippage

Category	Severity	Location	Status
Logical Error	● Medium	MarketLiquidation.sol: 135	Acknowledged

Description

In the liquidate function, a liquidator is forced to liquidate from the asset with the largest notional value. This approach can result in less profit for the liquidator, especially when the position with the highest notional value involves a less liquid pool, this is because smaller pools experience more slippage during swaps.

This limitation makes positions with certain assets as their largest holding less attractive to liquidate due to reduced profits, increasing the likelihood of incurring bad debt.

Recommendation

Allow liquidators to choose which asset they want to liquidate.

Resolution

Ambit Team: The MarketLiquidation contract does expose a second liquidate method that allows the liquidator to determine which asset they want to liquidate as opposed to the asset with the largest notional value. However, the initial implementation of the Liquidator is relying on the method that utilizes the findLargestPosition in order to favor paying off more debt in this scenario.

MKT-1 | User Can Avoid Accrued Interest

Category	Severity	Location	Status
Protocol Manipulation	● Medium	Market.sol: 396	Resolved

Description PoC

The Ambit protocol supports the usage of a DiscountModel to decrease the interest accrued on a user's borrow position. Once the DiscountModel is set, the interest is calculated with the function calculateDiscountedLiabilities instead of the typical interest calculation.

With the model a user can receive a discount for their entire borrow amount: Math.min(userLiability.borrowed, points * AMOUNT_PER_POINT).toUint128();

A user may be in the market for a prolonged period of time while there is no discount campaign and the marketState.borrowIndex will largely increase during that elapsed time. In order to avoid paying that interest, a user can wait to accrueLiabilities until a discount model is set.

A user can burn Ambit to immediately increase their points such that the discount matches the borrow position, although this may require a large amount of Ambit. Afterwards the user will call function accrueLiabilities where the resultant interest will be zero, and the user will avoid the interest that they would have had to pay otherwise.

Recommendation

Be extremely cautious with the discounts in the DiscountModel to prevent users from creating interest-free borrow positions. Furthermore, accrue liabilities for the user prior to burning Ambit.

Resolution

Ambit Team: The issue was resolved in commit 02bedd8.

MKT-2 | MarketState Not Updated Prior To Calling Pause/Unpause

Category	Severity	Location	Status
Logical Error	● Medium	Market.sol: 348	Resolved

Description

When a market is paused, interest no longer accrues for positions in the calculateMarketState function as the function ceases execution early if paused() is true. However, when a market is paused with the pause function, the pending interest is not accrued.

Therefore, when a market is paused, the interest accrued since the last accrueLiabilities call is effectively lost, as subsequent calls to accrueLiabilities update the marketState.lastUpdate timestamp without accruing interest.

Additionally, when a market is unpaused, the protocol does not ensure that accrueLiabilities is called prior to unpausing the market. This would result in the interest accrued since the last accrueLiabilities call needing to be paid, since we no longer enter the paused() is true

Recommendation

Call the accrueLiabilities function prior to calling pause/unpause on the market

Resolution

Ambit Team: The issue was resolved in commit 48aeb70.

LDTR-2 | Possible Lack Of Incentive For Liquidations

Category	Severity	Location	Status
Incentives	● Medium	Liquidator.sol: 173	Resolved

Description

The treasuryFee is taken at a higher priority than the caller fee for liquidations. In the event where there is only enough profit to fully pay out the treasury fee, the treasury fee is paid and the callerFee is reduced. Therefore the incentive for a liquidator to expeditiously liquidate an account may be reduced and possibly insufficient in some cases.

Recommendation

Consider taking the callerFee as the highest priority so that there is always sufficient incentive for liquidations to occur in the system.

Resolution

Ambit Team: The issue was resolved in commit ce65380.

MKT-3 | discountModel Updated Before Liabilities Accrue

Category	Severity	Location	Status
Logical Error	● Medium	Market.sol: 109	Resolved

Description

In the setInterestRateModel function, the pending interest is updated with the accrueLiabilities function to correctly account for the outstanding interest having accrued under the previous interest rate model.

However in the setDiscountModel function the new _discountModel is set before the interest is updated with the accrueLiabilities function, therefore any pending interest is treated as if it had accrued with the new discount model configured while this is not the case.

Recommendation

Update the pending interest values with the accrueLiabilities function before updating the _discountModel in the setDiscountModel function.

Resolution

Ambit Team: The issue was resolved in commit 1a1d62c.

TRH-3 | Token Rewards DoS

Category	Severity	Location	Status
DoS	● Medium	TokenRewardHooks.sol	Resolved

Description

Since the token reward epochs are being pushed into the array on every donation and never removed, there is a risk of DoS because users who supply for the first time will need to go through the entire list of expired epochs for each of the reward tokens present.

This will cost users who are supplying to their portfolio for the first time to pay an excessive amount of gas overhead, which may potentially exceed the block gas limit and prevent the user from supplying.

Recommendation

Consider not going through all past epochs when accruing for a new user and just setting the epoch index to the current one.

Resolution

Ambit Team: The issue was resolved in commit 9b48119.

TRH-4 | Rewards During A Pause For A Reward Token Can Be Accrued

Category	Severity	Location	Status
Logical Error	● Medium	TokenRewardHooks.sol	Resolved

Description

Paused tokens are not differentiated from when calling accrue() on them. This allows a user to accrue them even though they should not be by calling claim(). This then allows users to claim yield for a token that is currently on pause.

Furthermore, tokens paused through disableRewardToken() do not get accrued when calling accrue(). This would effectively lose users yield.

Recommendation

Consider having a check for if a token is paused or not in accrue(TokenReward storage tokenReward) and returning early if it is.

Also consider accruing paused tokens up until the pause point in order for all users to be able to claim them.

Resolution

Ambit Team: The issue was resolved in commit c4b376a.

TRH-5 | Invalid Result Returned From getClaimableRewards

Category	Severity	Location	Status
Logical Error	● Medium	TokenRewardHooks.sol: 328	Resolved

Description

Each reward epoch restarts the accumulationIndex from 0. However in the getClaimableRewards function, the userReward.accumulationIndex is never reset when iterating through the list of epochs. Therefore the resulting claimable reward amount received from the getClaimableRewards function will be inaccurate.

Recommendation

Reset the userReward.accumulationIndex to zero upon iterating to a new epoch.

Resolution

Ambit Team: The issue was resolved in commit 746b070.

SNAP-2 | Untruncated Timestamps Are Unmatchable

Category	Severity	Location	Status
Logical Error	● Medium	SnapshotLib.sol: 70, 89	Resolved

Description

In both the find and search functions, the method for validating whether a snapshot has been found or not is to check whether the supplied timestamp is exactly equal to the snapshot timestamp. However, this will rarely be the case as the provided timestamp is not truncated to satisfy the 5-minute intervals that the snapshot timestamps are stored with.

Recommendation

Truncate the provided timestamp so that it will be much more likely to line up with the snapshot timestamps.

Resolution

Ambit Team: The issue was resolved in commit 1b4c59a.

YBC-1 | Invalid totalAssets Validation

Category	Severity	Location	Status
Logical Error	● Medium	YieldBearingCustodian.sol: 36	Resolved

Description

The getTotalSupply function returns the totalAssets balance rather than the supply of the vault token.

However in the BaseCustodian.withdraw function, the shares amount is compared with the result of the getTotalSupply function. Therefore a shares amount is validated being against an underlying token amount.

Recommendation

In the check in the BaseCustodian on line 79 compare the amount which is a token amount against the result of the getTotalSupply function.

Otherwise change the definition of the getTotalSupply function, but be careful to update where it is used, as the validation using the getTotalSupply function in the Portfolio contract is currently correct as it assumes the getTotalSupply is an underlying token amount.

Resolution

Ambit Team: The issue was resolved in commit 989c186.

MKTV-1 | Deleveraging Does Not Consider The Flash Fee

Category	Severity	Location	Status
Configuration	● Medium	MarketplaceVendor.sol: 191	Acknowledged

Description

The flash fee is not considered when approving the flash lender to pay back the flash loan, therefore the MarketplaceVendor cannot be used to deleverage positions unless the MarketplaceVendor is a flash fee whitelisted address.

The MarketplacePurchaser on the other hand does account for the flash fee.

Recommendation

Consider implementing flash fee support in the MarketplaceVendor. If the MarketplaceVendor contract is intended to always be a flash fee whitelisted address then be careful to always whitelist it.

Resolution

Ambit Team: This is fine as the MarketplaceVendor is whitelisted and excluded from fees.

LDTR-3 | Liquidations Prevented By Slippage

Category	Severity	Location	Status
Protocol Manipulation	● Medium	Liquidator.sol: 77	Acknowledged

Description

Because some liquidations require a swap, they could be prevented if a malicious actor front-runs the transaction and moves the pool price out of the acceptable range. The malicious actor could sandwich the transaction to undo this price manipulation so that minimal capital is lost.

Consequently, the liquidations will be prevented by an insufficient output amount revert, leaving bad debt in the protocol.

Recommendation

Use an aggregator and/or ensure the discount is large enough to meet slippage requirements. Furthermore, consider allowing unprofitable liquidations to occur, marking the forceful liquidation with a boolean flag if necessary.

Resolution

Ambit Team: This should be fine as the Liquidator is the on-chain backup liquidation function. We have an off-chain liquidator that will be the primary source of liquidations and that uses 1inch to sell assets where applicable.

LOTY-4 | User Points Not Getting Claimed In accruePoints Loses Them Rewards

Category	Severity	Location	Status
Logical Error	● Low	Loyalty.sol	Acknowledged

Description

Since accrued points are practically points that don't get rewards accrued on them users who call `accruePoints()` but then do not directly claim in the same will lose out on rewards from those points.

Recommendation

Consider calling `claimRewards()` for users in `accruePoints()` .

Resolution

Ambit Team: Users will only be subjected to claiming points which will internally accrue. `getClaimablePoints()` method will return the up-to-date points without a call to `accruePoints()`.

MKT-4 | Discounts Do Not Apply Without An Update To Liabilities

Category	Severity	Location	Status
Logical Error	● Low	Market.sol: 389	Acknowledged

Description

Liability interest discounts do not apply to a user's position if they do not explicitly update their liabilities in the discount period even if they have had that position since before the discount period.

For example, if the protocol goes from a standard interest rate to a discount rate and finally back to a normal rate, the user will not experience the discounted rate at all if the function `calculateUserLiability` is not called during the discount period.

Recommendation

Clearly document to users that not updating their liabilities during the discount period will not apply the said discounts.

Resolution

Ambit Team: The drawbacks are understood here, but unlikely to cause an issue as the primary purpose of the discount model is to allow points holders to have discounted rates.

PTFLO-1 | User Can Withdraw And Supply In The Same Block

Category	Severity	Location	Status
Logical Error	● Low	Portfolio.sol	Acknowledged

Description

Users are not supposed to be able to complete a withdrawal and supply in the same block. It is prevented with the use of the ensureWithdrawAvailability() function.

The issue arises due to the _lastUpdateBlocks mapping of a user for a particular token being deleted upon their whole balance being taken out. This allows a user to first withdraw and then supply the same token within the same block.

Recommendation

Do not delete the _lastUpdateBlocks mapping even if the user withdraws their whole balance.

Resolution

Ambit Team: The check was only supposed to stop a supply followed by a withdraw in the same block, a withdraw followed by a supply should be fine.

ML-5 | User's Discount Not Considered On Liquidation

Category	Severity	Location	Status
Logical Error	● Low	MarketLiquidation.sol	Acknowledged

Description

When liquidating an account, market.getLiabilities(account) is called to calculate the user's liabilities. However, the user can have their liabilities reduced if their accrued loyalty points were to be claimed and consequently have a healthy position.

As a result, a user is liquidated when they could have a healthy position just by claiming their points, leading to loss of funds for the liquidated account.

Recommendation

Claim the account's points on the call to function accrueLiabilities.

Resolution

Ambit Team: We want to encourage the users to interact with the protocol and keeping the manual claiming of points is a way to do that.

MKT-5 | Discounts Favor Borrowers With A Higher Liabilities/Principal Ratio

Category	Severity	Location	Status
Logical Error	● Low	Market.sol	Resolved

Description

The discount percentage gets calculated as a fraction of the principle of the borrower's loan and then that percentage gets taken out of the total liabilities of the loan. The issue here is that this model heavily favors users who have a high liabilities/principal ratio.

This is against the interest of the protocol since those types of users likely haven't yet repaid large amounts of interest or have borrowed riskier assets.

Recommendation

Consider calculating the percentage with the total liabilities instead in order to favor users with lower accrued interest.

Resolution

Ambit Team: The issue was resolved in commit 1406fa3.

UNIPO-1 | Uniswap Oracle Manipulation

Category	Severity	Location	Status
Oracle Manipulation	● Low	UniswapV2PriceOracle.sol: 31-33	Resolved

Description

One of the price oracles Ambit utilizes is the ratio of the reserves in a Uniswap pool. However, these reserves are highly susceptible to manipulation, as an attacker can manipulate the reserves of one of the tokens with a flashloan and exaggerate the price.

Due to the inflated worth of their collateral, an attacker's borrowing limit is increased and they are able to borrow more funds from the protocol than originally intended. This can drain the lending market and leave the protocol holding less value than stolen once the price of the collateral is restored.

Recommendation

Use the Uniswap TWAP for the price instead of the reserves calculation.

Resolution

Ambit Team: The issue was resolved in commit c58344b.

LDTR- 4 | MIN_SLIPPAGE_ALLOWED is never used

Category	Severity	Location	Status
Superfluous Code	● Low	Liquidator.sol: 32	Resolved

Description

In the liquidator contract the constant MIN_SLIPPAGE_ALLOWED is set to 0.5%. However, the constant is never used and no minimum slippage is enforced.

Recommendation

If the MIN_SLIPPAGE_ALLOWED is not intended to be used remove it.

Resolution

Ambit Team: The issue was resolved in commit baa9a01.

PTFLO-2 | Portfolio Hooks Prone To Reentrancy

Category	Severity	Location	Status
Reentrancy	● Low	Portfolio.sol	Acknowledged

Description

The Portfolio provides the capability to perform a particular action before supplying/withdrawing and after supplying/withdrawing through the use of hooks on a particular asset. Because state changes occur between the before hook and after hook, if a hook were to expose an external call outside of the protocol's system, a user could reenter.

One potential issue that could arise is if a user could use the before hook to re-enter into function supply and circumvent the maximum supply validation because the Custodian's supply has yet to be updated.

Recommendation

Be extremely careful with the hooks that are supported on each asset, so that calls outside of Ambit are restricted. Furthermore, consider adding `nonReentrant` guards.

Resolution

Ambit Team: This is accepted, however, hooks are still only an internal component used by the dev team when extending the protocol and this can be controlled.

DIRM-1 | Current Market Liabilities Used For Utilization

Category	Severity	Location	Status
Logical Error	● Low	DynamicInterestRateModel.sol: 78	Acknowledged

Description

The market's utilization ratio is calculated using the instantaneous liabilities instead of using the liabilities stored in the snapshots: `uint256 liabilities = depositorVault.getLiabilities(marketAddress);`

Furthermore, the `Snapshot.totalLiabilities` is not used at all. Ultimately this may lead to some gaming of the utilization ratio, and the liabilities can be directly affected with a `borrow` and `repay`.

Recommendation

Ensure the fees for borrowing are large enough to deter utilization ratio manipulation. Furthermore, consider using the average of the snapshot liabilities in the utilization calculation.

Resolution

Ambit Team: This has now been changed to use an average of the totalAssets. Additionally, the push the utilization higher, the user would have to borrow and in which case fees would be applied, and they would also have to have the assets supplied to their portfolio.

To push the utilization lower (and this decrease the rate) they would have to continually repay a loan and then borrow again and in which case the fees should provide an economic disincentive.

LDTR-5 | Liquidations Halted

Category	Severity	Location	Status
Warning	● Low	Liquidator.sol	Acknowledged

Description

In times of volatility it may be possible for liquidations to drain the Liquidator contract of the liquidation fund. In most cases the liquidator should receive back the USDT that was used to initiate the liquidation, however in some cases when liquidating vault tokens it may be possible for the amount of USDT in the Liquidator contract to decrease.

Once the liquidator is drained no liquidations can occur and the protocol will accrue bad debt. Additionally, particularly large liquidations may eclipse the size of the current liquidation fund, in which case these liquidations would only partially occur and require several liquidations to finalize.

The risk is that the price of the collateral decreases further, increasing the chances of an insolvent position.

Recommendation

Be aware of this risk and ensure that the liquidation fund is always large enough to cover any liquidations.

Resolution

Ambit Team: The Liquidator is just the on-chain backup, there will be an off-chain liquidator running also that will handle this.

MKT-6 | Users Are Charged Interest On Fee Amounts

Category	Severity	Location	Status
Unexpected Behavior	● Low	Market.sol: 215	Acknowledged

Description

When user's are charged fees, the fee amount is added to their principle and they simply do not receive that fee amount in USDT, instead it goes to the treasury.

Since the fee amount is still added to the user's principle however, the user still must pay interest on this fee amount that is taken. This may be unexpected for users and can cause confusion on the real amount of interest being charged.

Recommendation

Consider if this is the expected behavior, and be sure to document it for users.

Resolution

Ambit Team: Yes this is expected behavior.

LOTY-5 | Superfluous Pending Rewards Calculation

Category	Severity	Location	Status
Superfluous Code	● Low	Loyalty.sol: 260	Resolved

Description

The pending rewards for an account are added to the userReward.accrued at the end of the getClaimableRewards function. However this is unnecessary as the pending rewards are already updated in the accrueRewards function call on line 258.

Recommendation

Remove the pending rewards calculation from the return statement in the getClaimableRewards function.

Resolution

Ambit Team: The issue was resolved in commit 1c1580d.

GLOBAL-3 | Interest Fee Design

Category	Severity	Location	Status
Protocol Design	● Low	Global	Resolved

Description

Depositors only receive their rewards when borrowers choose to pay back their debt, however given that this is perpetual debt it is possible that depositors do not accrue their rightful interest until much later.

You lent funds in the vault in block 100, in block 101 User A borrows funds from the vault, in block 200 you withdraw funds from the vault, in block 210 User A finally repays their borrowed amount and pays out their interest to the vault.

However now you are not able to collect this interest given the time duration mismatch between the borrower leveraging the funds in the vault and actually paying back the accrued interest.

In other systems such as Abracadabra Money, the accrued interest is credited as it is generated and the system does not wait for users to repay their debt to provide yield.

Recommendation

Consider the drawbacks of rewarding depositors when a borrower repays their debt. In a future iteration it may be useful to consider a different method for distributing fees so that lenders are fairly compensated.

Resolution

Ambit Team: The issue was resolved in commit 1c1580d.

MKT-7 | Typo

Category	Severity	Location	Status
Typo	● Low	Market.sol: 258	Resolved

Description

The word “accrued” is misspelled as “accurred” on line 258.

Recommendation

Correct it to “accrued”.

Resolution

Ambit Team: The issue was resolved in commit 929534e.

TRH-6 | Donations Lost When totalShares Is 0

Category	Severity	Location	Status
Lost Rewards	● Low	TokenRewardHooks.sol: 183	Acknowledged

Description

When the totalShares is 0, the accumulate function returns a 0 amount to be distributed, therefore donations will not be distributed when there is 0 shares for the _asset.

Recommendation

The TokenRewardHooks contract is Sweepable, so these funds may be rescued by the treasury, but consider if this is the expected behavior. If not, consider implementing a method for donators to reclaim undistributed rewards. Or for an additional incentive for suppliers to acquire shares when the supply is 0.

Resolution

Ambit Team: Yes this is fine, it’s unlikely that donations will be occuring without any assets being supplied.

ML-6 | Blacklisted Accounts May Avoid Liquidation

Category	Severity	Location	Status
Blacklist	● Low	MarketLiquidation.sol: 224	Acknowledged

Description

Accounts that are blacklisted for USDT cannot be liquidated when the refundAmount is nonzero as the settleLiquidation function would attempt to transfer USDT to the blacklisted account.
On BSC there is no blacklist functionality for USDT, however on other chains USDT does have a blacklist feature, so care should be taken when deploying to new chains.

Recommendation

Carefully consider new chain deployments as certain features may be incompatible or exploitable on new chains.

Resolution

Ambit Team: When we do deploy to these chains we will look at the option to actually just keep the funds in the treasury if the account is blacklisted.

VEST-1 | Typo

Category	Severity	Location	Status
Typo	● Low	TokenVesting.sol: 21	Resolved

Description

There is a typo in the variable ellapsed as it should be named elapsed.

Recommendation

Rename the ellapsed variable to elapsed.

Resolution

Ambit Team: The issue was resolved in commit 929534e.

DVLT-1 | No Basis Point Validation For setBorrowLimit

Category	Severity	Location	Status
Validation	● Low	DepositorVault.sol: 248	Resolved

Description

In the setBorrowLimit function, there is no safety check that the provided amount is within the max for a basis point value if absoluteOrRelative is AbsoluteOrRelative.RELATIVE.

Recommendation

Consider adding a safety check that the amount is within the expected range for basis point values if the absoluteOrRelative value is AbsoluteOrRelative.RELATIVE.

Resolution

Ambit Team: The issue was resolved in commit 382b0ad.

TRH-7 | Redundant Accruals

Category	Severity	Location	Status
Optimization	● Low	TokenRewardHooks.sol: 101, 113	Resolved

Description

In the afterSupply and afterWithdraw functions the accrue method is invoked. However the accrue method will have already been called previously as a part of the beforeSupply and beforeWithdraw functions.

Recommendation

Remove the redundant calls to accrue in the afterSupply and afterWithdraw hooks.

Resolution

Ambit Team: The issue was resolved in commit 60a421b.

TRH-8 | Superfluous tokenReward.index Update

Category	Severity	Location	Status
Superfluous Code	● Low	TokenRewardHooks.sol: 136	Resolved

Description

In the enableRewardsToken function the tokenReward.index is assigned to 0 if the tokenReward.index is 0 and otherwise it is assigned to the tokenReward.index.
This is pointless and therefore the line can be removed.

Recommendation

Remove the superfluous tokenReward.index assignment on line 136.

Resolution

Ambit Team: This was removed with the update to token rewards referenced in a few other issues.

YBC-2 | Lacking Migration Logic

Category	Severity	Location	Status
Configuration	● Low	YieldBearingCustodian.sol	Resolved

Description

There is no migration logic for the yield bearing custodian.

Recommendation

Consider if this is the expected behavior, implement migration logic if it should be migratable.

Resolution

Ambit Team: The issue was resolved in commit eeadec7.

YBC-3 | Lack Of Donation Fee In The YieldBearingCustodian

Category	Severity	Location	Status
Protocol Fees	● Low	YieldBearingCustodian.sol	Acknowledged

Description

There is no implementation for the `previewDonationFee`, therefore there will be no donation fee or fee receiver.

Recommendation

Be sure this is the desired behavior for the `YieldBearingCustodian` donations.

Resolution

Ambit Team: Yes this is fine, only the DepositorVault has a donation fee.

DVM-1 | Depositor Vault Key Re-computed

Category	Severity	Location	Status
Best Practices	● Low	DepositorVaultMigrator.sol: 79	Resolved

Description

When the depositorVault is assigned in the registry the depositor vault key is recomputed by hashing the “ambit.depositorVault” string. However it would be a better practice to access and use the already computed keys from the AddressRegistryExtensions file. This way potential typos can be avoided when they would cause critical issues.

Recommendation

Rather than manually hashing the key for the depositor vault, use the pre-computed AMBIT_DEPOSITOR_VAULT key from the AddressRegistryExtensions contract.

Resolution

Ambit Team: The issue was resolved in commit b8239f5.

DVM-2 | Redundant Inheritance

Category	Severity	Location	Status
Superfluous Code	● Low	DepositorVaultMigrator.sol: 22	Resolved

Description

The DepositorVaultMigrator contract inherits from both the AdminAccessControl and AuthorizedAccessControl contracts.

However the AuthorizedAccessControl also inherits from the AdminAccessControl contract. Therefore it is unnecessary for the DepositorVaultMigrator contract to directly inherit from the AdminAccessControl contract.

Recommendation

Remove the direct inheritance from the AdminAccessControl in the DepositorVaultMigrator contract.

Resolution

Ambit Team: The issue was resolved in commit 77197c8.

YVLT-1 | Invalid 0 Price Returned From getExchangeRate

Category	Severity	Location	Status
Unexpected Behavior	● Low	YieldVault.sol: 39	Resolved

Description

The getExchangeRate ought to return an exchange rate of _scalar in the case where the totalShares is 0 as this will be the exchange rate for any deposit when the totalShares are 0.

Recommendation

Consider updating the getExchangeRate implementation such that it returns _scalar when the totalShares is 0.

Resolution

Ambit Team: The issue was resolved in commit 6d498c3.

SNAP-3 | Unnecessary Truncate Call

Category	Severity	Location	Status
Optimization	● Low	SnapshotLib.sol: 41	Resolved

Description

There is no need to truncate the snapshot.timestamp a second time as the timestamp has already been truncated on line 38.

Recommendation

Remove the truncate function call on line 41.

Resolution

Ambit Team: The issue was resolved in commit 6d498c3.

ML- 7 | Users can be Liquidated for More than Needed

Category	Severity	Location	Status
Documentation	● Low	MarketLiquidation.sol: 141	Acknowledged

Description

Trusted liquidators have the capability to liquidate an arbitrary amount and from any asset in a user's portfolio. This flexibility allows a liquidator to intentionally liquidate specific assets and amounts in a strategic order to maximize their profit.

If a liquidator performs two liquidations— the first being as much as possible while keeping the position unhealthy, and the second being the full 50% of the user's largest position—they can liquidate more than if they had only done the 50% liquidation in the first attempt.

Recommendation

It is recommended to clearly document that there is no hard cap for how much a user can be liquidated.

Resolution

Ambit Team: Liquidation will be restricted to trusted actors so this shouldn't present a problem initially. If its decided to open this up then we will revisit.

SNAP-4 | Unexpected Found Value Returned

Category	Severity	Location	Status
Unexpected Behavior	● Low	SnapshotLib.sol: 73	Acknowledged

Description

In the find function on line 73, in the event that the queried timestamp is in the middle of the latest and oldest timestamps, the find function always returns true for the found result.

However a snapshot match might not have been found in the search function, and instead the first snapshot that is larger will be returned. This may not fit the consumer's idea of the found boolean from the find function.

Recommendation

Consider if the search function should determine whether a snapshot was found or not depending on if the match case was hit.

Resolution

Ambit Team: The found parameter is supposed to represent that the requested timestamp was prior to any data that currently exists in the history and therefore the value could not be found, however, it returns the first available value in this case (which should be after the timestamp requested).

DMOD-1 | Points Discount Extremely Low

Category	Severity	Location	Status
Logical Error	● Low	DiscountModel.sol: 41	Resolved

Description

- 1e8 AMBT = 10 PTS
- 1 PTS = 1e10 Wei Discount Amount (assuming 18 decimal precision of underlying vault token)
- 1e8 AMBT = 1e11 Wei Discount Amount
- USDT = 1e18 Wei
- 1e15 AMBT = 10,000,000 AMBT = 1e18 Wei Discount Amount
- To obtain 1 USDT discount a user may require 10,000,000 AMBT which is 10% of its supply.

Recommendation

Document to users this behavior and/or increase the discount.

Resolution

Ambit Team: This issue was resolved in commit 9fb99ed

MKTP-1 | Late Validation

Category	Severity	Location	Status
Validation	● Low	MarketplacePurchaser.sol: 98	Resolved

Description

In the buy function, the amount is first validated to be less than the maxAmount and the adjusted maxAmount that is subsequently used to flash loan is the minimum of the amount and the maxFlashLoan available in the FlashLender.

However when the maxFlashLoan is less than the amount the flash loan will always fail with the validation later on in the onFlashLoan function on line 130. Therefore this case is not validated early enough and needlessly allowed by the Math.min operation on line 98.

Recommendation

Validate that the amount is less than the maxFlashLoan directly in the buy function to avoid unnecessary logic.

Resolution

Ambit Team: This issue was resolved in commit 5e78aa3

DVLT-2 | safeTransferFrom Should Occur Before Updates

Category	Severity	Location	Status
Reentrancy	● Low	DepositorVault.sol: 320, 359	Resolved

Description

Calls to `safeTransferFrom` should generally occur before any state updates in a function to avoid yielding an invalid state where updates have occurred before the tokens have been received.

Recommendation

Move uses of `safeTransferFrom` to the beginning of the `repay` and `deposit` functions in the `DepositorVault`.

Resolution

Ambit Team: This issue was resolved in commit 6652b39

SMMA-1 | Invalid UniswapV2 Expiration Timestamp

Category	Severity	Location	Status
Logical Error	● Low	SpotMarketMarketplaceAdapter.sol	Resolved

Description

block.timestamp is used as an expiration timestamp for UniswapV2 swaps, practically the same as passing no timestamp. This presents an issue as the swap will go through no matter if the transaction stays in the mempool for a prolonged period.

Recommendation

Consider using a timestamp passed from the caller.

Resolution

Ambit Team: This issue was resolved in commit 9b9ddda

DMOD-2 | Precision Loss Can Lead to Loss of Loyalty Points

Category	Severity	Location	Status
Precision	● Low	DiscountModel.sol: 41	Resolved

Description

In the calculateDiscountAmount function, if the vault asset's decimals is less than 8 decimals the user can lose points due to precision loss when the normalize function is called:

```
uint256 points = loyalty.getPoints(account).normalize(8, decimals);
```

Recommendation

To minimize the loss of points perform multiplication before division by normalizing points * AMOUNT_PER_POINT instead of just points.

Resolution

Ambit Team: This issue was resolved in commit a414931

DMOD-3 | Incorrect Integer Casting

Category	Severity	Location	Status
Logical Error	● Low	DiscountModel.sol	Resolved

Description

The following `return (rate - DISCOUNT_RATE.percentOf(rate)).toUint128();` is incorrect as `percentOf()` returns `uint256`. Instead of only `DISCOUNT_RATE.percentOf(rate)` being cast, the whole expression is cast.

Recommendation

Consider re-implementing the expression like so:

```
return rate - (DISCOUNT_RATE.percentOf(rate).toUint128());
```

Resolution

Ambit Team: This issue was resolved in commit 8df6db8

LDYV-1 | Wrong Event Emission

Category	Severity	Location	Status
Logical Error	● Low	LinearDistributedYieldVault.sol	Resolved

Description

`emit Donate(msg.sender, amount - feeAmount, feeAmount, feeReceiver);`

The event above logs the donation amount as `amount - feeAmount` even though the fees also get logged by the event.

Recommendation

Consider changing `amount - feeAmount` to `amount`.

Resolution

Ambit Team: This issue was resolved in commit 5126b26

LDYV-2 | Vault Distributes Yield Even With No Shares Present

Category	Severity	Location	Status
Logical Error	● Low	LinearDistributedYieldVault.sol	Acknowledged

Description

LinearDistributedYieldVault distributes interest even if there are no shares/depositors in it. This will allow the first share depositor to get all the leftover yield.

Recommendation

Consider not distributing yield if there are no shares in the vault.

Resolution

Ambit Team: This is fine as it should be rare that we yield without any shares as in theory if there's nothing in the vault then there is nothing to borrow to generate yield anway. Additionally if this did occur, it should encourage users to deposit.

DVLT-3 | Vault Migration Sets The First Snapshot With The Newest Values

Category	Severity	Location	Status
Logical Error	● Low	DepositorVault.sol	Acknowledged

Description

When migrating to a new DepositorVault instance the first snapshot in the observer gets set as the latest params instead of copying the last few snapshots.

Recommendation

Consider copying the last n snapshots into the new DepositorVault.

Resolution

Ambit Team: This is fine given;

- the frequency that we migrate the vault will be very low
- there's only a small window (10-15 mins) where the rate could be artificially forced low

ML-8 | Very Small Positions In <8 Tokens Cannot Be Liquidated

Category	Severity	Location	Status
Logical Error	● Low	MarketlLiquidation.sol	Acknowledged

Description

All positions are denominated in USD with 8 decimals of precision. A very small loan in a token with less precision than the USD precision will get truncated to 0 when normalizing the decimals in `uint256 total = totals[j - 1].normalize(_registry.getDepositorVault().getDecimals());`

This will then offset all following calculations in `liquidateInternal()` and will not allow for the position to be liquidated, leaving the protocol with bad debt it cannot remove.

Recommendation

Consider introducing a minimum deposit amount.

Resolution

Ambit Team: This would be a 0 total as you've identified. However, the user wouldn't even be able to borrow against that position in the first place as the borrow limit is also normalized to the decimal precision of the base asset. It's a different story if the price of the asset was a lot higher and then crashed, but that's an inherent risk with an oracle based lending protocol.

MKT-8 | Incorrect Order Of Operations

Category	Severity	Location	Status
Optimization	● Low	Market.sol	Resolved

Description

`uint256 elapsed = block.timestamp - marketState.lastUpdate;` gets calculated before timestamp validity gets checked.

Recommendation

Consider calculating `elapsed` after the check.

Resolution

Ambit Team: This issue was resolved in commit f698173

GLOBAL-4 | Disabling Hooks May Lead To Accounting Inaccuracies

Category	Severity	Location	Status
Logical Error	● Low	Global	Resolved

Description

Since Loyalty and TokenRewardHooks rely on hooks to accrue points and update indexes each time there is a change in the user's portfolio, freezing the hooks of a token will cause all logic in one of those two systems to experience accounting issues.

Recommendation

Consider implementing contract freezing logic for the two systems to freeze all their logic in regards to the particular token when its hooks get removed.

Resolution

Ambit Team: This issue was resolved in commit cfd4e20

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits