GUARDIAN

Sentiment

Leveraged Lending

Security Assessment

August 5th, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Wafflemakr, 0xCiphky, Osman Ozdemir, Cosine, Michael Lett

Client Firm Sentiment

Final Report Date August 5, 2024

Audit Summary

Sentiment engaged Guardian to review the security of its leveraged lending protocol, allowing for permissionless SuperPool and Pool creation. From the 17th of June to the 27th of June, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 24 High/Critical issues were uncovered and promptly remediated by the Sentiment team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the lending protocol product.

Security Recommendation Given the number of High and Critical issues detected, Guardian supports an independent security review of the protocol at a finalized frozen commit. Furthermore, the Sentiment team should increase testing with a variety of share valuations, collateral and borrow assets, and LTV ratios. The engagement exposed multiple blind spots that should be thoroughly tested and presented numerous opportunities for system malfunction.

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: **Arbitrum, Mainnet**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/sentiment-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	Sentiment
Language	Solidity
Codebase	https://github.com/sentimentxyz/protocol-v2
Commit(s)	23406ecf2a2f14bf10298cc695e4d9995129a529

Audit Summary

Delivery Date	August 5, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	5	0	0	0	0	5
● High	19	0	0	1	0	18
● Medium	23	0	0	2	0	21
● Low	15	0	0	4	0	11

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 Sentiment, fuzz-testing with Echidna was performed on the protocol's main functionalities. 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 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
SP-01	SuperPool.deposit() must consume exactly the number of assets requested	✅	✅	✅	10M+
SP-02	SuperPool.deposit() must credit the correct number of shares to the receiver	✅	✅	✅	10M+
SP-03	SuperPool.deposit() must credit the correct number of assets to the pools in depositQueue	✅	✅	✅	10M+
SP-04	SuperPool.deposit() must credit the correct number of shares to the pools in depositQueue	✅	✅	✅	10M+
SP-05	SuperPool.deposit() must credit the correct number of shares to the SuperPool for the pools in depositQueue	✅	✅	✅	10M+
SP-06	SuperPool.deposit() must update lastUpdated to the current block.timestamp for the pools in depositQueue	✅	✅	✅	10M+
SP-07	SuperPool.deposit() must credit pendingInterest to the totalBorrows asset balance for pools in depositQueue	✅	✅	✅	10M+
SP-08	SuperPool.deposit() must transfer the correct number of assets to the base pool for pools in depositQueue	✅	✅	✅	10M+
SP-09	SuperPool.deposit() must increase the lastTotalAssets by the number of assets provided	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SP-10	SuperPool.deposit() must always mint greater than or equal to the shares predicted by previewDeposit()	✅	✅	✅	10M+
SP-11	SuperPool.mint() must consume exactly the number of tokens requested	✅	✅	✅	10M+
SP-12	SuperPool.mint() must credit the correct number of shares to the receiver	✅	✅	✅	10M+
SP-13	SuperPool.mint() must credit the correct number of assets to the pools in depositQueue	✅	✅	✅	10M+
SP-14	SuperPool.mint() must credit the correct number of shares to the pools in depositQueue	✅	✅	✅	10M+
SP-15	SuperPool.mint() must credit the correct number of shares to the SuperPool for the pools in depositQueue	✅	✅	✅	10M+
SP-16	SuperPool.mint() must update lastUpdated to the current block.timestamp for the pools in depositQueue	✅	✅	✅	10M+
SP-17	SuperPool.mint() must credit pendingInterest to the totalBorrows asset balance for pools in depositQueue	✅	✅	✅	10M+
SP-18	SuperPool.mint() must transfer the correct number of assets to the base pool for pools in depositQueue	✅	✅	✅	10M+
SP-19	SuperPool.mint() must increase the lastTotalAssets by the number of assets consumed	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SP-20	SuperPool.mint() must always consume less than or equal to the tokens predicted by previewMint()	✅	✅	✅	10M+
SP-21	SuperPool.withdraw() must credit the correct number of assets to the receiver	✅	✅	✅	10M+
SP-22	SuperPool.withdraw() must deduct the correct number of shares from the owner	✅	✅	✅	10M+
SP-23	SuperPool.withdraw() must withdraw the correct number of assets from the pools in withdrawQueue	✅	❌	✅	10M+
SP-24	SuperPool.withdraw() must withdraw the correct number of shares from the pools in withdrawQueue	✅	❌	✅	10M+
SP-25	SuperPool.withdraw() must deduct the correct number of shares from the SuperPool share balance for the pools in withdrawQueue	✅	❌	✅	10M+
SP-26	SuperPool.withdraw() must update lastUpdated to the current block.timestamp for the pools in withdrawQueue	✅	✅	✅	10M+
SP-27	SuperPool.withdraw() must credit pendingInterest to the totalBorrows asset balance for pools in withdrawQueue	✅	✅	✅	10M+
SP-28	SuperPool.withdraw() must transfer the correct number of assets from the base pool for pools in withdrawQueue	✅	❌	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SP-29	SuperPool.withdraw() must decrease the lastTotalAssets by the number of assets consumed	✅	✅	✅	10M+
SP-30	SuperPool.withdraw() must redeem less than or equal to the number of shares predicted by previewWithdraw()	✅	✅	✅	10M+
SP-31	SuperPool.redeem() must credit the correct number of assets to the receiver	✅	✅	✅	10M+
SP-32	SuperPool.redeem() must deduct the correct number of shares from the owner	✅	✅	✅	10M+
SP-33	SuperPool.redeem() must withdraw the correct number of assets to the pools in withdrawQueue	✅	❌	✅	10M+
SP-34	SuperPool.redeem() must withdraw the correct number of shares from the pools in withdrawQueue	✅	❌	✅	10M+
SP-35	SuperPool.redeem() must deduct the correct number of shares from the SuperPool share balance for the pools in withdrawQueue	✅	❌	✅	10M+
SP-36	SuperPool.redeem() must update lastUpdated to the current block.timestamp for the pools in withdrawQueue	✅	✅	✅	10M+
SP-37	SuperPool.redeem() must credit pendingInterest to the totalBorrows asset balance for pools in withdrawQueue	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SP-38	SuperPool.redeem() must transfer the correct number of assets from the pools in withdrawQueue	✅	❌	✅	10M+
SP-39	SuperPool.redeem() must decrease the lastTotalAssets by the number of assets consumed	✅	✅	✅	10M+
SP-40	SuperPool.redeem() must withdraw greater than or equal to the number of assets predicted by previewRedeem()	✅	✅	✅	10M+
SP-41	The lastTotalAssets value before calling accrue should always be <= after calling it	✅	❌	✅	10M+
SP-42	Fee recipient shares after should be greater than or equal to fee recipient shares before	✅	✅	✅	10M+
SP-43	previewDeposit() must not mint shares at no cost	✅	✅	✅	10M+
SP-44	previewMint() must never mint shares at no cost	✅	✅	✅	10M+
SP-45	convertToShares() must not allow shares to be minted at no cost	✅	✅	✅	10M+
SP-46	previewRedeem() must not allow assets to be withdrawn at no cost	✅	✅	✅	10M+
SP-47	previewWithdraw() must not allow assets to be withdrawn at no cost	✅	✅	✅	10M+
SP-48	convertToAssets() must not allow assets to be withdrawn at no cost	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
SP-49	Profit must not be extractable from a convertTo round trip (deposit, then withdraw)	✅	✅	✅	10M+
SP-50	Profit must not be extractable from a convertTo round trip (withdraw, then deposit)	✅	✅	✅	10M+
SP-51	Shares must not be minted for free using deposit()	✅	✅	✅	10M+
SP-52	Shares must not be minted for free using mint()	✅	✅	✅	10M+
SP-53	Assets must not be withdrawn for free using withdraw()	✅	✅	✅	10M+
SP-54	Assets must not be withdrawn for free using redeem()	✅	✅	✅	10M+
SP-55	The vault's share token should have greater than or equal to the number of decimals as the vault's asset token	✅	✅	✅	10M+
SP-56	Share inflation attack possible, victim lost an amount over lossThreshold%	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PO-01	Pool.deposit() must increase poolId assets by assets and pending interest	✅	✅	✅	10M+
PO-02	Pool.deposit() must increase poolId shares by sharesDeposited	✅	✅	✅	10M+
PO-03	Pool.deposit() must consume the correct number of assets	✅	✅	✅	10M+
PO-04	Pool.deposit() must credit the correct number of shares to receiver	✅	✅	✅	10M+
PO-05	Pool.deposit() must transfer the correct number of assets to pool	✅	✅	✅	10M+
PO-06	Pool.deposit() must update lastUpdated to the current block.timestamp	✅	✅	✅	10M+
PO-07	Pool.deposit() must credit pendingInterest to the totalBorrows asset balance for poolID	✅	✅	✅	10M+
PO-08	Pool.redeem() must decrease poolId assets by assetsRedeemed + pendingInterest	✅	✅	✅	10M+
PO-09	Pool.redeem() must decrease poolId shares by shares amount	✅	✅	✅	10M+
PO-10	Pool.redeem() must credit the correct number of assets to receiver	✅	✅	✅	10M+
PO-11	Pool.redeem() must consume the correct number of shares from receiver	✅	✅	✅	10M+
PO-12	Pool.redeem() must transfer the correct number of assets to receiver	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PO-13	Pool.redeem() must update lastUpdated to the current block.timestamp	✅	✅	✅	10M+
PO-14	Pool.redeem() must credit pendingInterest to the totalBorrows asset balance for poolID	✅	✅	✅	10M+
PO-15	The pool.totalAssets.assets value before calling accrue should always be <= after calling it.	✅	✅	✅	10M+
PO-16	The pool.totalBorrows.assets value before calling accrue should always be <= after calling it	✅	✅	✅	10M+
PO-17	Fee recipient shares after should be greater than or equal to fee recipient shares before	✅	✅	✅	10M+
PO-18	User debt value should be equal to 0 or greater than or equal to MIN_DEBT	✅	✅	✅	10M+
PO-19	Min Required Position Asset Value should be greater than total position debt value	✅	✅	✅	10M+
PO-20	The pool.totalAssets.shares values should always equal the sum of the shares of all users	✅	✅	✅	10M+
PO-21	The pool.totalBorrows.shares values should always equal the sum of the borrow share balances of all borrowers	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PM-01	PositionManager.newPosition() should set auth to true for owner	✅	✅	✅	10M+
PM-02	PositionManager.newPosition() should set ownerOf position to owner	✅	✅	✅	10M+
PM-03	PositionManager.deposit() must consume the correct amount of assets	✅	✅	✅	10M+
PM-04	PositionManager.transfer() must consume asset amount from position	✅	✅	✅	10M+
PM-05	PositionManager.transfer() must credit asset amount to recipient	✅	✅	✅	10M+
PM-06	PositionManager.borrow() must credit amount of assets to poolId total borrow asset balance	✅	✅	✅	10M+
PM-07	PositionManager.borrow() must credit amount of shares to poolId total borrow share balance	✅	✅	✅	10M+
PM-08	PositionManager.borrow() must credit amount of shares to poolId position share balance	✅	✅	✅	10M+
PM-09	PositionManager.borrow() must credit fee amount to feeRecipient	✅	✅	✅	10M+
PM-10	PositionManager.borrow() must credit the correct number of assets to position	✅	✅	✅	10M+
PM-11	PositionManager.borrow() must add poolId to debtPools	✅	✅	✅	10M+
PM-12	Position debt pools should be less than or equal to max debt pools	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PM-13	PositionManager.repay() must credit assets to pool	✅	✅	✅	10M+
PM-14	PositionManager.repay() must consume asset amount from position	✅	✅	✅	10M+
PM-15	PositionManager.repay() must consume amount of assets from poolId total borrow asset balance	✅	✅	✅	10M+
PM-16	PositionManager.repay() must consume amount of shares from poolId total borrow share balance	✅	✅	✅	10M+
PM-17	PositionManager.repay() must consume amount of shares from poolId position share balance	✅	✅	✅	10M+
PM-18	PositionManager.repay() must delete poolId from debtPools if position has no borrows	✅	✅	✅	10M+
PM-19	PositionManager.addToken() must add asset to position assets list	✅	✅	✅	10M+
PM-20	Position assets length should be less than or equal to max assets	✅	✅	✅	10M+
PM-21	PositionManager.removeToken() must remove asset from position assets list	✅	✅	✅	10M+
PM-22	PositionManager.liquidate() must credit the correct number of debt assets to pool	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
PM-23	PositionManager.liquidate() must credit the correct number of debt assets to poolPositionManager.liquidate() must credit the correct number of assets to liquidator	✅	✅	✅	10M+
PM-24	PositionManager.liquidate() must credit the correct number of fee assets to owner	✅	✅	✅	10M+
PM-25	PositionManager.liquidate() must consume the correct number of position assets from position	✅	✅	✅	10M+
PM-26	Position must be healthy after liquidation	✅	✅	✅	10M+
PM-27	PositionManager.liquidate() must consume the correct number of assets from the pools in debtData	✅	✅	✅	10M+
PM-28	PositionManager.liquidate() must consume the correct number of shares from the pools in debtData	✅	✅	✅	10M+
PM-29	PositionManager.liquidate() must consume amount of shares from poolId position share balance	✅	✅	✅	10M+
PM-30	PositionManager.liquidate() must update lastUpdated to the current block.timestamp for the pools in debtData	✅	✅	✅	10M+
PM-31	PositionManager.liquidate() must delete poolId from debtPools if position has no borrows	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Incorrect Amounts Are Used During Withdrawals	Validation	● Critical	Resolved
C-02	Free Borrowing When Collateral Is The Same Asset	Logical Error	● Critical	Resolved
C-03	DoS Pool By Allowing Excess Borrowing	DoS	● Critical	Resolved
C-04	Drain All Pools With SuperPool As Collateral	Logical Error	● Critical	Resolved
C-05	Attacker Can Drain All Funds from a Pool	Logical Error	● Critical	Resolved
H-01	setFee Honeypot Attack	Validation	● High	Resolved
H-02	Malicious RateModel Attack	Validation	● High	Resolved
H-03	Protocol Fees Are Donations	Logical Error	● High	Resolved
H-04	Reallocating Can Freeze Funds	Validation	● High	Resolved
H-05	DoS On Setting The LTV Of a Token To Zero	Logical Error	● High	Resolved
H-06	_reorderQueue Does Not Work As Expected	Logical Error	● High	Resolved
H-07	DoS In _removePool By Force Feeding	DoS	● High	Resolved
H-08	Reentrancy In SuperPool	Logical Error	● High	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
H-09	Util Ratio Calculation Is Incorrect	Logical Error	● High	Resolved
H-10	Impossible To Repay All Debt In Some Cases	Rounding	● High	Resolved
H-11	Funds Can Be Frozen By Reordering Queues	Validation	● High	Resolved
H-12	Max LTV Can Be Abused On Small Price Changes	Validation	● High	Resolved
H-13	DoS In _supplyToPools If One Deposit Fails	DoS	● High	Resolved
H-14	DoS In _withdrawFromPools Based On Util Ratio	DoS	● High	Resolved
H-15	Lenders can deposit into full SuperPools	Validation	● High	Acknowledged
H-16	Pool Initialized Multiple Times	Logical Error	● High	Resolved
H-17	Users Can Avoid Liquidations	Logical Error	● High	Resolved
H-18	feeRecipient Set To Zero Blocks Functionality	Validation	● High	Resolved
H-19	Missing onlyFactory Check In SuperPool	Validation	● High	Resolved
M-01	Unsafe Use Of transferFrom	Validation	● Medium	Resolved
M-02	Native Ether Can Not Be Deposited	DoS	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-03	superPoolCap Not Implemented	Validation	● Medium	Resolved
M-04	Pausing The PositionManager Disables addToken	DoS	● Medium	Resolved
M-05	Incorrect Key Is Used In PositionManager	Validation	● Medium	Resolved
M-06	Repay & Seize Order Increases Bad Debt Risk	Logical Error	● Medium	Resolved
M-07	Same Heartbeat Assumed For All Price Feeds	Validation	● Medium	Resolved
M-08	Bad Debt Is Not Handled	Logical Error	● Medium	Resolved
M-09	Preview Functions In SuperPool Are Not Accurate	Logical Error	● Medium	Resolved
M-10	Fees Can Be Avoided With Dust Amounts	Validation	● Medium	Resolved
M-11	Check If Asset Is Known In Deposit Flow	Validation	● Medium	Resolved
M-12	Timelock Functionality Is Redundant	Logical Error	● Medium	Resolved
M-13	\`exec\` Should Have \`whenNotPaused\`	Logical Error	● Medium	Resolved
M-14	Rebasing Tokens Are Not Supported	Validation	● Medium	Resolved
M-15	\`setRegistry\` Should Invoke \`updateFromRegistry\`	Logical Error	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-16	Liquidator Can Seize Non-PositionAssets	Logical Error	● Medium	Resolved
M-17	Interest Not Accrued Before Rate Update	Logical Error	● Medium	Resolved
M-18	Pool Cap Can Be Bypassed	Reentrancy	● Medium	Resolved
M-19	Missing Pause Functionality	Logical Error	● Medium	Resolved
M-20	Chainlink Oracles Lack Proper Validation	Oracle	● Medium	Resolved
M-21	Reallocate Can Leave Assets In Contract	Validation	● Medium	Acknowledged
M-22	Missing Storage Gaps	Logical Error	● Medium	Acknowledged
M-23	Reallocate Will Redeem Assets Instead of Shares	Logical Error	● Medium	Resolved
L-01	Accrue Before feeRecipient Update	Logical Error	● Low	Acknowledged
L-02	Allocators Can Bypass Pool Caps	Validation	● Low	Resolved
L-03	approve Race Condition	Logical Error	● Low	Resolved
L-04	Missing Zero Assets Check In redeem	Validation	● Low	Resolved
L-05	Unused Events	Superfluous Code	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-06	Typo	Typo	● Low	Resolved
L-07	Registry Address Can Be Immutable	Optimization	● Low	Resolved
L-08	Mismatch Between Code And Developer Comment	Optimization	● Low	Resolved
L-09	Consider Using Ownable2Step	Optimization	● Low	Acknowledged
L-10	Superfluous Code	Superfluous Code	● Low	Resolved
L-11	Can't Liquidate When Price Is Stale	Oracle	● Low	Acknowledged
L-12	Incorrect Pool Can Be Removed From Queue	Logical Error	● Low	Resolved
L-13	PositionManager Allows Free Flashloans	Logical Error	● Low	Acknowledged
L-14	Superfluous OnlyOwner Modifier	Logical Error	● Low	Resolved
L-15	Liquidation Discount Is Incorrect	Math	● Low	Resolved

C-01 | Incorrect Amounts Are Used During Withdrawals

Category	Severity	Location	Status
Validation	● Critical	SuperPool.sol: 441	Resolved

Description PoC

When assets are withdrawn from SuperPool, the `redeem` function in the base pool is invoked. This function requires the share amount as an input. However, during this call, the asset amount is provided instead, leading to significant accounting issues and potential loss of funds.

Recommendation

To address this issue, it is recommended to convert the user-provided asset amount to the corresponding share amount before proceeding with the redemption process.

Resolution

Sentiment Team: The issue was resolved in PR#222.

C-02 | Free Borrowing When Collateral Is The Same Asset

Category	Severity	Location	Status
Logical Error	● Critical	Global	Resolved

Description PoC

Users deposit collateral and borrow assets using the PositionManager contract, and every action that changes a Position’s balance requires a health check via RiskEngine and RiskModule.

The health check is done by comparing total debt of a position and total asset of a position. However, this health check is inaccurate when a borrowed asset and the collateral asset are the same. The RiskEngine accounts newly borrowed assets as user provided collateral, which causes the check to be incorrect.

A pool owner can
• Set the `ltv` to 1 for the borrow asset.
• Add the borrow asset as collateral asset to his position with `addToken`
• Borrow all assets from the pool without adding any collateral.
Resulting in all funds to be frozen for regular depositors.

Recommendation

Do not allow borrow asset and the collateral asset to be the same.

Resolution

Sentiment Team: Resolved.

C-03 | DoS Pool By Allowing Excess Borrowing

Category	Severity	Location	Status
DoS	● Critical	Pool.sol: 325	Resolved

Description PoC

Users can borrow from the `Pool` only by using the `PositionManager`.
Although users must deposit enough collateral to pass the health check after borrowing, the `Pool` does not check if the `poolId` has enough liquidity to support the amount borrowed.

Any amount borrowed that exceeds the `poolId` liquidity, will effectively steal this liquidity from other pools, and total borrows will be greater than total assets.

There are multiple impacts with this issue:
• pool redeems are DoS'ed when calculating:
`uint256 assetsInPool = pool.totalAssets.assets - pool.totalBorrows.assets`
• lenders won't be able to redeem from other pools, as there is not enough assets in balance
• SuperPool `maxWithdraw` reverts as `getLiquidityOf` calculation will underflow

Recommendation

Prevent positions from borrowing more assets than the liquidity of the `poolId`.

Resolution

Sentiment Team: The issue was resolved in PR#243.

C-04 | Drain All Pools With SuperPool As Collateral

Category	Severity	Location	Status
Logical Error	● Critical	SuperPool.sol: 367	Resolved

Description PoC

SuperPool vault shares should have the same decimals as the underlying ASSET. The decimal value is set in the constructor. However, this is not the case, as depositing 1e18 assets will give you 1e36 shares.

The issue relies on _convertToShares when the first user deposits, where lastTotalAssets and totalSupply are 0, but it will multiply by 10 ** DECIMALS:

```
shares = assets.mulDiv(totalSupply() + 10 ** DECIMALS, lastTotalAssets + 1, rounding);
```

Although users will be able to redeem the shares for the correct amount of tokens, there is a discrepancy between the decimals() of the vault and the minted share units. Users will be able to use this vault token as collateral to borrow assets against. When calculating the asset value of the vault token in RiskModule, the value returned will be 1e18 times greater than expected.

Therefore, users will be able to drain pools by borrowing all assets, as the collateral value is basically infinite: `vaultTokens(1e36) * priceInEth(1e18) / decimals(1e18) = 1e36 ether value`

Recommendation

Implement the following:
• ~~shares = assets.mulDiv(totalSupply() + 10 ** DECIMALS, lastTotalAssets + 1, rounding);~~
• shares = assets.mulDiv(totalSupply() + 1, lastTotalAssets + 1, rounding);
• ~~assets = shares.mulDiv(lastTotalAssets + 1, totalSupply() + 10 ** DECIMALS, rounding);~~
• assets = shares.mulDiv(lastTotalAssets + 1, totalSupply() + 1, rounding);

Resolution

Sentiment Team: The issue was resolved in PR#240.

C-05 | Attacker Can Drain All Funds from a Pool

Category	Severity	Location	Status
Logical Error	● Critical	RiskModule.sol: 211	Resolved

Description PoC

The _getMinReqAssetValue function in the RiskModule contract determines the minimum required asset value for a position to be considered healthy. This function is called by the isPositionHealthy function, which performs a health check after every action or series of actions taken by users to ensure their position remains healthy.

The issue arises because the _getMinReqAssetValue function relies on the length of the position's positionAssets array for the inner loop when calculating the required asset value for a position to be healthy. However, a position does not need to have any assets added to its positionAssets list in order to perform a borrow.

As a result, a user could perform a borrow with no funds, and the health check performed at the end would still pass. This occurs because _getMinReqAssetValue would return zero, given that the position's positionAssets is empty, despite the position having open debt from the borrow. Consequently, a user could borrow all funds from a pool and transfer them to personal accounts, effectively draining the pool.

Recommendation

Update the _getMinReqAssetValue function to revert if the resulting minReqAssetValue is zero. This function is only called when debt exceeds zero, so minReqAssetValue should almost always be greater than zero.

One exception is when a position's value falls to zero, causing insolvency. In such cases, this fix could block the liquidation of bad debt, so an admin function should be implemented to handle these scenarios.

Resolution

Sentiment Team: The issue was resolved in PR#271.

H-01 | setFee Honeypot Attack

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 309-313	Resolved

Description PoC

SuperPool owners can adjust the fee anytime instantly and no boundaries for the fee are set. This enables a honeypot attack:

- Attacker creates a SuperPool with a 1% fee
- Users deposit into the pool
- The attacker sets the fee way above 100%
- A few seconds pass and the pending interest for the attacker is >= all assets in the contract
- Attacker withdraws all funds of the SuperPool

Recommendation

Implement min/max values and a timelock for critical parameter updates like fees.

Resolution

Sentiment Team: The issue was resolved in PR#225.

H-02 | Malicious RateModel Attack

Category	Severity	Location	Status
Validation	● High	Pool.sol: 425	Resolved

Description PoC

• Anyone can create a BasePool with an arbitrary contract as RateModel
• Anyone can create a SuperPool
• The owner of a SuperPool can add a new BasePool to a SuperPool and reallocate all funds to it any time

These conditions allow the following attack:

• SuperPool owner creates a BasePool with a malicious RateModel and mints some shares
• SuperPool owner adds the BasePool to the queue of the SuperPool
• SuperPool owner reallocates all funds to the malicious BasePool
• The malicious actor calls accrue on the BasePool and the malicious RateModel returns that a lot of tokens in interest were accrued
• The malicious actor withdraws all funds in the BasePool with the few shares minted in the beginning and the users of the SuperPool lose everything

Recommendation

Do not allow arbitrary addresses as RateModel.

Resolution

Sentiment Team: The issue was resolved in PR#264.

H-03 | Protocol Fees Are Donations

Category	Severity	Location	Status
Logical Error	● High	Pool.sol: 428-429	Resolved

Description

• The interestFee and originationFee are fees that go to the protocol.
• Anyone can create a new BasePool and set the interestFee and originationFee while doing so.

Therefore the protocol fees are not enforced and act like donations instead. As it makes no economic sense for the BasePool owner to set these fees above 0 the protocol will probably lose a lot of money.

Recommendation

Enforce the protocol fees.

Resolution

Sentiment Team: The issue was resolved in PR#224.

H-04 | Reallocating Can Freeze Funds

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 345-360	Resolved

Description PoC

In the reallocate function it is not checked if the given pools to deposit to are part of the deposit/withdraw queue.
Therefore malicious SuperPool owners can deposit funds into a pool that is not in the queue, which will freeze funds for the users of the SuperPool.
The owner could then spread on social media that users need to pay X amount of funds to unfreeze it (this could be even done with a contract).
When the users pay the amount, the owner can unfreeze the funds by adding the new pool to the queue or reallocating the funds back to the original pool.

Recommendation

Add a check to see if the pools are part of the deposit/withdraw queue.

Resolution

Sentiment Team: The issue was resolved in PR#231.

H-05 | DoS On Setting The LTV Of a Token To Zero

Category	Severity	Location	Status
Logical Error	● High	RiskModule.sol: 228	Resolved

Description PoC

• The owner of a BasePool can set the LTV of a token to zero.
• The isPositionHealthy function will revert if a position holds a token in the asset list and this token has an LTV of zero.

Therefore if the owner updates the LTV of a token to zero and any position holds this token any interaction with it through the PositionManager will revert.

This has multiple bad consequences:
• Borrowers debt increases but they are not able to repay
• Borrowers can not be liquidated
• Borrowers are not able to get their collateral back
• The owner of a base pool can create unliquidatable positions
Unliquidatable positions:
• Owner sets the LTV of a token above 0
• The owner creates a position adds this token as collateral, borrows funds, and does something risky with them
• The owner sets the LTV for this token to 0
• Now the position is not liquidatable till the owner sets the LTV back to a value above 0

Recommendation

Do not allow to set the LTV of a token to zero.

Resolution

Sentiment Team: The issue was resolved in PR#237.

H-06 | _reorderQueue Does Not Work As Expected

Category	Severity	Location	Status
Logical Error	● High	SuperPool.sol: 488	Resolved

Description PoC

SuperPool contract has deposit and withdrawal queues. Order of these queues are important since deposits and withdrawals are done based on the order, which is supposed to be arranged by the owner.

However, due to incorrect implementation of the _reorderQueue function, the owner can not change queue orders. This function takes a new order as an indexes array, and is supposed to rearrange the order based on indexes. But it only copies the previous order as is with the newQueue[i] = queue[i] line.

Recommendation

Use the inputted indexes array to determine new order. Change newQueue[i] = queue[i] to newQueue[i] = queue[indexes[i]].

Resolution

Sentiment Team: The issue was resolved in PR#220.

H-07 | DoS In _removePool By Force Feeding

Category	Severity	Location	Status
DoS	● High	SuperPool.sol: 467	Resolved

Description

• The _removePool function reverts if the SuperPool still owns assets in the given pool.
• Anyone can deposit into a BasePool and set any SuperPool as the receiver of the assets.

This enables a malicious actor the possibility to front run a _removePool call and deposit one wei of assets into the SuperPool to DoS the call.

Recommendation

Implement a function that reallocates and removes the pool in one transaction.

Resolution

Sentiment Team: The issue was resolved in PR#266.

H-08 | Reentrancy In SuperPool

Category	Severity	Location	Status
Logical Error	● High	SuperPool.sol: 396-398	Resolved

Description PoC

The last three actions in the withdrawal flow of the SuperPool are:
• Burning the share tokens
• Transferring the funds to the user
• Reducing the lastTotalAssets value

Therefore if the attacker reenters on receiving the tokens the total amount of shares is already reduced, but the total amount of assets is not.

The first action in withdraw/deposit is accruing interest calculated based on the saved lastTotalAssets value and the current total amount. Therefore the given difference on reentering (as lastTotalAssets is not reduced yet) is seen as interest and the owner of the pool receives fees on this interest.

This enables the following attack path:
• SuperPool owner deposits tokens into the own pool
• SuperPool owner withdraws the tokens and reenters on receiving them over and over again
• Every time the difference is seen as interest and the owner receives fees
• Owner withdraws the gained fees
• The owner repeats this process over and over again till the SuperPool is drained completely

Recommendation

• safeTransfer should be the last action in this flow.
• Use a reentrancy guard on critical functions.

Resolution

Sentiment Team: The issue was resolved in PR#244.

H-09 | Util Ratio Calculation Is Incorrect

Category	Severity	Location	Status
Logical Error	● High	LinearRateModel.sol: 65	Resolved

Description PoC

The LinearRateModel calculates the totalAssets amount by summing up the totalBorrows and the idleAssetAmt. But the idleAssetAmt already is the totalAssets amount (pool.totalAssets.assets from the BasePool). This leads to an incorrect calculation of the utilization ratio.

For example:
Unborrowed Amount = 500
Borrowed Amount = 500
Therefore util ratio = 50%
Calculation in the LinearRateModel:
idleAssetAmt = pool.totalAssets.assets = 1000
totalAssets = totalBorrows + idleAssetAmt = 500 + 1000 = 1500
util = totalBorrows / totalAssets = 500 / 1500 = 33.33%

Therefore the calculated utilization ratio is 33.33% when 50% of the funds are borrowed. As the utilization ratio is smaller than it should be the lender receives less interest than they should.

Recommendation

Do not calculate the totalAssets amount as the idleAssetAmt already is the totalAssets amount.

Resolution

Sentiment Team: The issue was resolved in PR#227.

H-10 | Impossible To Repay All Debt In Some Cases

Category	Severity	Location	Status
Rounding	● High	Pool.sol: 181, 387	Resolved

Description PoC

Users can pay their Position’s whole debt by passing type(uint256).max value as the repayment amount. In that case, PositionManager contract will calculate Position’s total debt and make a call to the underlying pool for the payment.

However, the getBorrowsOf function uses convertToAssets, which rounds down by default, and this causes amount to be paid to round down. This amount is later used in the repay function. repay in the Pool contract also rounds down the borrowShares amount to burn, which is done to ensure excess debt isn't pushed to other users.

As a result of rounding down twice during the repay flow, the Position will always have 1 borrowShare left even though the user tries to pay whole amount unless the amount is exactly the multiple of asset:share ratio.

This would cause repayment to fail due to MIN_DEBT requirement. Also, even if the Position has other debts that cover MIN_DEBT, the repaid debt pool can not be removed from debtPools array from the Position contract due to remaining 1 share.

Recommendation

Firstly, getBorrowsOf function should round up. Then, ensure that all borrowShares are burned when all debt is paid.

Resolution

Sentiment Team: The issue was resolved in PR#245.

H-11 | Funds Can Be Frozen By Reordering Queues

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 480-491	Resolved

Description

The SuperPool owner has the capability to reorder deposit and withdrawal queues. However, there is a lack of duplicate entry check in the `_reorderQueue` function, which could potentially enable a malicious owner to freeze funds.

For instance:
• withdrawQueue = [1, 2, 3]
• A malicious owner reorders with indices [0, 0, 0]
• Resulting withdrawQueue = [1, 1, 1]
As a result, the funds deposited in pools 2 and 3 become unwithdrawable.

Recommendation

It is recommended to implement a duplicate check within the function to prevent this issue from occurring. Verify the `indexes` param in the reorder queue functions contain all pool ids in the queue array.

Resolution

Sentiment Team: The issue was resolved in PR#232.

H-12 | Max LTV Can Be Abused On Small Price Changes

Category	Severity	Location	Status
Validation	● High	RiskEngine.sol: 181-186	Resolved

Description PoC

The given maxLtv value provided by the protocol as a deployment parameter is 1e18 which equals a 1:1 ratio LTV. This means users can borrow assets for $100 by providing $100 collateral if the LTV for the given collateral token is set to 1e18. Therefore only a small price change not updated in the oracle because of deviation, or not yet updated as the attacker front runs the oracle update can be abused to make risk-free profit and put the pool into bad debt.

Here is a possible attack scenario:
• The pool owner adds a new asset to the BasePool with a 1e18 LTV (1:1 ratio)
• The price of this asset is currently outdated as chainlinks deviation threshold is not reached (for example 0.5% deviation and 0.4% change)
• The attacker takes a flash loan of the given asset and borrow all funds with it
• The attacker sells the funds from the BasePool for a 0.4% - trading fees profit and pays back the flashloan
• The attacker made a large profit depending on the size of the pool and the pool will be in bad debt as soon as the oracle price is updated

Recommendation

Use a lower maxLtv value.

Resolution

Sentiment Team: The issue was resolved in PR#281.

H-13 | DoS In _supplyToPools If One Deposit Fails

Category	Severity	Location	Status
DoS	● High	SuperPool.sol: 415	Resolved

Description

Every deposit call could fail if the BasePool is paused, or if the cap in the BasePool is reached.
As this call is not wrapped into a try-catch block, the transaction will revert and therefore the user is not able to deposit into the other pools of the queue. For example, if the first BasePool in the queue is paused the whole SuperPool deposit function is not usable.

Recommendation

Wrap the deposit call into a try-catch block as it is done with withdraws.

Resolution

Sentiment Team: The issue was resolved in PR#234.

H-14 | DoS In _withdrawFromPools Based On Util Ratio

Category	Severity	Location	Status
DoS	● High	SuperPool.sol: 435	Resolved

Description

The _withdrawFromPools function uses the getAssetsOf function of the given BasePool to calculate the maximum that can be withdrawn. But the getAssetsOf function returns the amount that the SuperPool owns in the BasePool, not the amount that the SuperPool can withdraw right now.

This can lead to reverts and using the queue in a suboptimal way:
• The SuperPool has 100 tokens in the BasePool
• The user tries to withdraw 15 tokens
• The util ratio of the BasePool is 95% (Only 10 tokens can be withdrawn right now as the rest is borrowed)
• Therefore the SuperPool could withdraw 10 tokens from this pool and 5 tokens from the next one
• Instead, it tries to withdraw 15 tokens from this pool, the call fails and it will try to withdraw 15 from the next one
• This reorders the queue in a suboptimal way

Recommendation

Reduce the borrowed funds from the assetsInPool, or set the assetsInPool to the balance of the BasePool if it's smaller. Do not withdraw more than available liquidity.

Resolution

Sentiment Team: The issue was resolved in PR#235.

H-15 | Lenders can deposit into full SuperPools

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 404-421	Acknowledged

Description

The _supplyToPools function loops through all BasePools in the queue and supplies funds to them as long as the cap is not reached. But this function does not revert if all caps were reached and it was impossible to supply funds to any pool. Therefore lenders are still able to deposit funds into a full SuperPool and mint shares which is very capital inefficient and will reduce the yield per share of all lenders.

Example:
• X lenders deposit a sum of Y USDC into pools with 5% APY over the SuperPool and reach the cap
• These lenders now receive 5% APY on their deposits
• More lenders deposit into the SuperPool which is already full
• The new lender's funds are not put to work but they still receive shares of the SuperPool and therefore a share of the yield from the BasePools
• Every lender now receives less than 5% APY on their deposits

Recommendation

Revert at the end of the _supplyToPools function.

Resolution

Sentiment Team: Acknowledged.

H-16 | Pool Initialized Multiple Times

Category	Severity	Location	Status
Logical Error	● High	Pool.sol: 410	Resolved

Description PoC

The initializePool function allows users to initialize a new pool, which users can then deposit the pool's assets into as well as access other functionalities. The initializePool function enforces a check to ensure that an already initialized pool can't be reinitialized, as this would reset the pool's totalAssets and totalBorrows back to zero, wiping out all previous users' deposits and debt.

This check is done by ensuring that ownerOf[poolId] is zero, which implies the newly created pool does not currently exist. The problem is that the initializePool function allows the owner address to be set to zero. Therefore, a pool could be created with the ownerOf[poolId] as address zero, and since this check is used to ensure a pool is not being reinitialized, this will be bypassed in this case, and the pool can be reinitialized, resetting all values.

Recommendation

Since the zero address is used as a check to ensure a pool ID does not already exist in the initializePool function, restrict users from being able to set the owner parameter as the zero address to avoid the problem above.

Resolution

Sentiment Team: The issue was resolved in PR#238.

H-17 | Users Can Avoid Liquidations

Category	Severity	Location	Status
Logical Error	● High	RiskModule.sol: 70	Resolved

Description PoC

Users can borrow funds from pools using the PositionManager contract, as long as the position remains healthy, verified by riskEngine.isPositionHealthy(position).

The isPositionHealthy function has a flaw, as it has a condition where it can revert:
if (totalDebtValue != 0 && totalDebtValue < MIN_DEBT) revert RiskModule_DebtTooLow(position, totalDebtValue);

The liquidate function uses isPositionHealthy to avoid liquidating healthy positions, but if the position is not healthy and the totalDebtValue is less than MIN_DEBT, the liquidation fails. Although isPositionHealthy is checked when a position borrows assets, the eth value of the debt can decrease below MIN_DEBT.

Recommendation

Consider removing the MIN_DEBT check from RiskModule and add it only to the borrow and repay functions in PositionManager

Resolution

Sentiment Team: The issue was resolved in PR#270.

H-18 | feeRecipient Set To Zero Blocks Functionality

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 259	Resolved

Description

The feeRecipient is initially set during the deployment of a SuperPool and can also be changed using the setFeeRecipient function in the SuperPool contract. This address receives fees when interest is accrued, which are minted as SuperPool shares to the feeRecipient.

The issue is that there is no restriction on the address to which the feeRecipient can be set. As a result, a SuperPool owner could inadvertently or maliciously set the feeRecipient to the zero address. This will cause the accrue function, which is called in most operations, to revert, as the SuperPool _mint function will revert when the recipient address is zero.

Recommendation

To prevent this issue, add a validation check in the deploySuperPool and setFeeRecipient functions to ensure that the feeRecipient address is not set to the zero address when the fee is greater than zero.

Resolution

Sentiment Team: The issue was resolved in PR#246.

H-19 | Missing onlyFactory Check In SuperPool

Category	Severity	Location	Status
Validation	● High	SuperPool.sol: 101-117	Resolved

Description PoC

All pools exist within the singleton pool contract, with one singleton contract per SuperPool factory. Developer comments indicate that SuperPools should only be deployed through the factory contract to ensure they all point to the same singleton pool implementation.

However, the constructor of the SuperPool lacks a check to confirm it is being called by the factory. This opens the possibility for users to deploy a SuperPool that points to a malicious base pool. With multiple SuperPools containing the same asset, distinguishing between malicious SuperPools and regular ones becomes challenging for ordinary users.

Recommendation

Implement an `onlyFactory` check to restrict SuperPool deployment to the factory only, preventing unauthorized users from deploying potentially harmful SuperPools.

Resolution

Sentiment Team: The issue was resolved in PR#268.

M-01 | Unsafe Use Of transferFrom

Category	Severity	Location	Status
Validation	● Medium	PositionManager.sol: 424	Resolved

Description PoC

Some ERC-20 tokens return a boolean instead of reverting therefore using `transferFrom` will not revert when the transfer fails.

This can be abused in the liquidate function as the flow looks like the following:
• The liquidator repays the debt of the borrower:
• The funds are transferred from the liquidator to the position (if the transfer fails with such a token nothing happens here)
• The debt of the borrower is reduced in the pool
• The liquidator seizes funds from the position as reward

Therefore if the `transferFrom` call failed without reverting the debt of the borrower is reduced but the funds in the pool stayed the same and the liquidator receives rewards from the position for free.

Recommendation

Use `safeTransferFrom` instead of `transferFrom`.

Resolution

Sentiment Team: The issue was resolved in PR#219.

M-02 | Native Ether Can Not Be Deposited

Category	Severity	Location	Status
DoS	● Medium	Position.sol: 86	Resolved

Description

The exec function of the Position contract can transfer native ether, as this might be needed to interact with some protocols (for example to pay gas for a 2-step flow).

But it is not possible to use this feature as ether can not be deposited into the Position contract:
• The deposit function can not be used to deposit native ether
• The exec function is not payable
• And there is no receive or fallback function in the Position contract

Recommendation

Implement the possibility to deposit native ether and/or make the exec functions payable and forward the msg.value from the PositionManager to the Position contract.

Resolution

Sentiment Team: The issue was resolved in PR#247.

M-03 | superPoolCap Not Implemented

Category	Severity	Location	Status
Validation	● Medium	SuperPool.sol: 43	Resolved

Description

The superPoolCap variable is initialized and can be updated, but is never checked in the deposit flows (only in the maxDeposit view function).

Recommendation

Check if the superPoolCap is reached.

Resolution

Sentiment Team: The issue was resolved in PR#228.

M-04 | Pausing The PositionManager Disables addToken

Category	Severity	Location	Status
DoS	● Medium	PositionManager.sol: 384	Resolved

Description

The `addToken` function is paused when the `PositionManager` is paused. This prevents borrowers from adding new tokens as collateral to their position, which could result in the borrowers not being able to keep their position healthy.

Here is an example of such a scenario:
• A borrower opens a position with a collateral token (for example a BasePool or SuperPool share token) and borrows funds
• Something bad happens in the system and the `PositionManager` as well as the pool of the collateral token is paused
• The collateral of the borrower losses value
• As the pool is paused the borrower is not able to get more tokens and increase the collateral of the position
• Also as the `addToken` function is paused the borrower is not able to add a new collateral token to the position
The same could happen with another token that is for any reason not available at the given moment.

Recommendation

Remove the `whenNotPaused` modifier from the `addToken` function.

Resolution

Sentiment Team: The issue was resolved in PR#267.

M-05 | Incorrect Key Is Used In PositionManager

Category	Severity	Location	Status
Validation	● Medium	PositionManager.sol: 83-84	Resolved

Description

Module addresses are fetched from the Registry contract and these addresses are stored with key to address mappings. Address keys are constant variables in contracts and they are determined using keccak hashes.

SENTIMENT_POSITION_BEACON_KEY in the PositionManager contract is stated as "0xc77ea3242ed8f193508dbbe062eaeef25819b43b511cbe2fc5bd5de7e23b9990". However, the correct hash result of keccak(SENTIMENT_POSITION_BEACON_KEY) is "0x6e7384c78b0e09fb848f35d00a7b14fc1ad10ae9b10117368146c0e09b6f2fa2".

If the Registry contract owner uses the correct key while setting addresses, positionBeacon address in the PositionManager contract will be retrieved incorrectly from the Registry contract.

Recommendation

Use the correct hash for constant keys.

Resolution

Sentiment Team: The issue was resolved in PR#226.

M-06 | Repay & Seize Order Increases Bad Debt Risk

Category	Severity	Location	Status
Logical Error	● Medium	PositionManager.sol: 418-445	Resolved

Description

In the current liquidate flow the liquidator has to first repay debt from the own pocket and receive funds from the liquidated borrower's position after that. If the unhealthy position is very big, fewer users (or no one) might be able to repay it from their own wallet.

Reversing the order of this flow (seize before repay) could enable more users to have enough funds to repay the debt. This leads to more liquidations on time and therefore decreases the likelihood of bad debt.

Recommendation

Reverse the order of the repay and seize loop in the liquidate function.

Resolution

Sentiment Team: The issue was resolved in PR#248.

M-07 | Same Heartbeat Assumed For All Price Feeds

Category	Severity	Location	Status
Validation	● Medium	ChainlinkEthOracle.sol: 34	Resolved

Description

The same heartbeat (one hour) is assumed for all chainlink price feeds, but there are assets with different heartbeats for example 24 hours this will result in all operations with a 24-hour heartbeat asset to only work 1 hour a day and DoS the rest of the time.

Recommendation

Implement the possibility to set the heartbeat for each price feed individually.

Resolution

Sentiment Team: The issue was resolved in PR#239.

M-08 | Bad Debt Is Not Handled

Category	Severity	Location	Status
Logical Error	● Medium	PositionManager.sol: 405-451	Resolved

Description

At the moment the system does not handle bad debt, the liquidator always has to repay the full loan. If the loan to repay is higher than the assets in the Position it makes no economic sense to call liquidate as the liquidator would lose money. This leads to no one calling liquidate if bad debt occurs and the bad debt probably increases even further.

The missing opportunity to take bad debt would lead to major problems in the system when a black swan event occurs.

Recommendation

Handle bad debt by repaying the maximum amount possible and either reducing the amount owned by the lenders or increasing the debt of the borrowers.

Resolution

Sentiment Team: The issue was resolved in PR#272.

M-09 | Preview Functions In SuperPool Are Not Accurate

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

Description

SuperPool is a contract that is compatible with ERC4626. It includes multiple preview functions that, in the end, call internal functions `_convertToShares` or `_convertToAssets`.

These internal functions utilize the `lastTotalAssets` variable in their calculations. However, `lastTotalAssets` does not represent the most updated asset amount, as it does not account for accrued interest since the last update. Therefore, the results of the preview functions are not accurate.

Some more examples of non-compliance include:
• `previewDeposit` does not simulate accrue, so `deposit` might mint less shares than previewed.
• `previewMint` does not simulate accrue, so `mint` might consume more assets than previewed.
• `previewRedeem` does not simulate accrue, so `redeem` might withdraw less assets than previewed.
• `previewWithdraw` does not simulate accrue, so `withdraw` might burn more shares than previewed.
• `maxDeposit/maxMint` do not correctly return the amount that can be deposited, as the cap can be bypassed.
• `maxDeposit/maxMint` do not return 0 if pools are paused for deposits
• `maxWithdraw/maxRedeem` return more assets than the real amount available, as `pool.getLiquidityOf` adds interest accrued.
• `deposit` should revert if all of `assets` cannot be deposited (due to poolCap limit)

Recommendation

To ensure accurate preview calculations, it is suggested to call the `simulateAccrue` function and utilize the returned `newTotalAssets` value. This will provide a more precise calculation of assets including accrued interest. Furthermore, consider making the Superpool EIP-4626 compliant.

Resolution

Sentiment Team: The issue was resolved in PR#240.

M-10 | Fees Can Be Avoided With Dust Amounts

Category	Severity	Location	Status
Validation	● Medium	Pool.sol: 350-352	Resolved

Description

The origination fee (borrowing fee) rounds down (in favor of the borrower) and no minimum borrowing amount is enforced. Therefore borrowers can avoid paying borrowing fees by borrowing dust amounts multiple times.

For example:
originationFee = 0.01e18 (1%)
Amount to borrow = 99
fee = amt _ originationFee / 1e18
99 _ 0.01e18 = 0.99e18
fee = 0.99e18 / 1e18 = 0.99 = 0

This will likely lead to a loss (because of gas fees) for the borrower on most tokens (1e18 precision) but it could be profitable with low-precision tokens.

Recommendation

Implement a minimum borrow amount or round up (against the borrower).

Resolution

Sentiment Team: The issue was resolved in PR#269.

M-11 | Check If Asset Is Known In Deposit Flow

Category	Severity	Location	Status
Validation	● Medium	PositionManager.sol: 308	Resolved

Description

The PositionManager contract contains two mappings, isKnownAddress and isKnownFunc, which define the universe of the protocol. isKnownAddress specifies recognized addresses that a Position can interact with.

The transfer function verifies whether the token is known as expected. However, the deposit function lacks this validation. Borrowers can deposit any asset to their position as collateral, causing these tokens to become locked since users are unable to transfer them afterwards.
An analogous issue is present in the addToken function.

Recommendation

Ensure that the address is verified in these functions.

Resolution

Sentiment Team: The issue was resolved in PR#250.

M-12 | Timelock Functionality Is Redundant

Category	Severity	Location	Status
Logical Error	● Medium	Global	Resolved

Description

Some updates in the protocol require a 24-hour timelock period. These updates are requested initially and then accepted or rejected after the timelock period has elapsed.

However, the requester, accepter, and rejecter are all the same person. A malicious owner could request an update days or weeks before it is actually needed and simply wait for the opportune moment to accept it, rendering the timelock feature ineffective.

Recommendation

Consider implementing a deadline, such as 12 or 24 hours, for accepting a request after the timelock period has elapsed. Do not allow a pending request to be accepted after this deadline.

Resolution

Sentiment Team: The issue was resolved in PR#251.

M-13 | exec Should Have whenNotPaused

Category	Severity	Location	Status
Logical Error	● Medium	PostiionManager.sol: 271	Resolved

Description

Some functions such as `borrow`, `addToken`, and `removeToken` in the PositionManager contract have the `whenNotPaused` modifier. The `exec` function should also have this modifier to prevent any unwanted actions from being executed when paused.

Recommendation

Include the `whenNotPaused` modifier in the `exec` function. Additionally, reassess other functions like `transfer` to determine if they should be permitted when paused.

Resolution

Sentiment Team: The issue was resolved in PR#267.

M-14 | Rebasing Tokens Are Not Supported

Category	Severity	Location	Status
Validation	● Medium	Pool.sol	Resolved

Description PoC

The `BasePool` uses internal variables to keep track of the pool balance as well as the user's balances and does not use the actual balance of the contract. Therefore rebalance and fee on transfer tokens will not work properly in the system.

Here is an example of how a fee on transfer token would act in the system:
• User1 deposits 1000 tokens to the pool
• User2 deposits 1000 tokens to the pool
• User1 withdraws 1000 tokens from the pool
• User2 tries to withdraw 1000 tokens from the pool, but the call will revert as the pools balance is lower than 1000 tokens because of the fee on every transfer

Recommendation

Update the system to support rebasing and fee on transfer tokens, or do not allow them by not setting oracles for these tokens and explicitly state these tokens are not supported.

Resolution

Sentiment Team: The issue was resolved in PR#265.

M-15 | setRegistry Should Invoke updateFromRegistry

Category	Severity	Location	Status
Logical Error	● Medium	Global	Resolved

Description

The Pool and PositionManager contracts utilize the Registry contract to retrieve crucial module addresses, and both contracts feature an `updateFromRegistry` function to update these addresses. In addition, both contracts are equipped with a `setRegistry` function that enables the owner to modify the Registry contract address.

It is advisable to invoke the `updateFromRegistry` function when setting a new Registry address in order to avoid potential mismatches. Failing to do so may result in the utilization of outdated module addresses until the `updateFromRegistry` function is manually called.

Recommendation

Always call the `updateFromRegistry` function when implementing a new Registry address.

Resolution

Sentiment Team: The issue was resolved in PR#252.

M-16 | Liquidator Can Seize Non-PositionAssets

Category	Severity	Location	Status
Logical Error	● Medium	PositionManager.sol: 405	Resolved

Description PoC

The liquidate function in the PositionManager contract allows anyone to liquidate an unhealthy position. Specifically, a user can select the amount and assets of debt to repay from the position and the amount and assets to seize from the position in return. The requirements being that the position must initially be unhealthy and end up being healthy after the liquidation, while also enforcing a maximum limit on the asset value that can be seized by the liquidator.

The issue is that, with the current implementation, the liquidator can seize assets not included in the position's positionAssets list, as long as they are known assets. This should not be allowed, as the health check performed on the position only considers the assets in the position's positionAssets list. Consequently, a user could unfairly lose assets that they did not intend to risk in a position by leaving them out of the asset list.

Recommendation

Modify the liquidate function to ensure that liquidators can only seize assets from a position's positionAssets list.

Resolution

Sentiment Team: The issue was resolved in PR#253.

M-17 | Interest Not Accrued Before Rate Update

Category	Severity	Location	Status
Logical Error	● Medium	Pool.sol: 471	Resolved

Description

The acceptRateModelUpdate function allows the pool owner to change the pool's rate model to the pending one after the timelock duration (one day). This rate model is used in the simulateAccrue function to calculate the interest accrued in the pool for the duration since it was last called (pool.lastUpdated) until the current block.timestamp.

The issue is that since the acceptRateModelUpdate function doesn't call accrue first, the next time a function that calls accrue is executed, the calculated interest will be based on the new rate model using the duration since the pool.lastUpdated, which could be a long time before the rate model was updated.

Recommendation

The acceptRateModelUpdate function should call accrue before updating the rate model to ensure that the interest calculation accurately reflects the old rate model up to the point of the update.

Resolution

Sentiment Team: The issue was resolved in PR#223.

M-18 | Pool Cap Can Be Bypassed

Category	Severity	Location	Status
Reentrancy	● Medium	Pool.sol: 225	Resolved

Description PoC

Users can deposit the pool's respective asset by calling the deposit function in the pool contract. The function allows the pool owner to enforce a pool cap; however, when an ERC777 asset is used, this limit can be bypassed.

This issue arises because pool.totalAssets.assets is updated after the asset transfer, which is the point of reentrancy. A user can reenter with pool.totalAssets.assets not yet updated, so the limit check will use the old total asset value.

For example, if the totalAssets are 10 tokens away from the cap, a user could initially deposit 10 tokens. During the asset transfer, the user can reenter the function and make a second deposit of 10 tokens. As a result, totalAssets will end up being 10 tokens over the limit.

Recommendation

The asset transfer should be the initial step in the function to avoid a malicious state where tokens have not been transferred to the pool yet.
Additionally, a nonReentrant modifier can be added to further secure the function.

Resolution

Sentiment Team: The issue was resolved in PR#236.

M-19 | Missing Pause Functionality

Category	Severity	Location	Status
Logical Error	● Medium	PositionManager.sol: 73	Resolved

Description

The contract PositionManager inherits from PausableUpgradeable but does not implement the onlyOwner functions required to enable this functionality. As a result, the owner is unable to pause/unpause functions that have the whenNotPaused modifier.

Additionally, SuperPool contract inherits from Pausable contract, does not use the whenNotPaused or contains the onlyOwner functions.

Recommendation

To address this issue, it is recommended that pause and unpause functions be added to the PositionManager contract.

If SuperPool contract is not meant to have pause functionality, consider removing the inheritance from Pausable.

Resolution

Sentiment Team: The issue was resolved in PR#242.

M-20 | Chainlink Oracles Lack Proper Validation

Category	Severity	Location	Status
Oracle	● Medium	ChainlinkEthOracle.sol: 95	Resolved

Description

The `_getPriceWithSanityChecks` function is used in the ChainlinkEthOracle and ChainlinkUsdOracle contracts to validate the fetched price updates from the Chainlink feed. However, in the current implementation, the validation only reverts when the price is less than zero, meaning a price of zero would be considered valid.

This price is used to calculate position values and determine if positions can be liquidated. Consequently, positions could be incorrectly liquidated if a price of zero is returned instead of the oracle reverting.

Recommendation

Modify the price check to revert if the returned price is less than or equal to zero.

Resolution

Sentiment Team: The issue was resolved in PR#254.

M-21 | Reallocate Can Leave Assets In Contract

Category	Severity	Location	Status
Validation	● Medium	SuperPool.sol: 345	Acknowledged

Description

The `reallocate` function will move funds from one pool to another one. However, there is no check that the total amount of assets redeemed as effectively deposited into the new pools. Assets not deposited will not earn interest, so SuperPool user's earnings will be affected.

Recommendation

Verify that the total amount redeemed from pools matches the total amount deposited.

Resolution

Sentiment Team: Acknowledged.

M-22 | Missing Storage Gaps

Category	Severity	Location	Status
Logical Error	● Medium	ERC6909	Acknowledged

Description

Pool is an upgradeable contract that inherits from ERC6909 contract. However, this contract don't use storage gaps, which will result in a corrupted storage if a variable is added/removed.

Recommendation

Consider adding storage gaps in the ERC6909 contract.

Resolution

Sentiment Team: Acknowledged.

M-23 | Reallocate Will Redeem Assets Instead of Shares

Category	Severity	Location	Status
Logical Error	● Medium	SuperPool.sol: 352	Resolved

Description

Owner can reallocate funds by redeeming from certain pools and depositing into different pools. The issue arises when executing `POOL.redeem` as it uses assets instead of shares. Not only owner will redeem more assets than expected, but assets might not be fully deposited during the second loop as total assets redeemed will be greater that `deposits` total assets.

Recommendation

The `withdraws` array should contain shares instead of assets amount.
Alternatively, calculate the shares that need to be redeemed with the asset amount passed and use that value in `POOL.redeem()`.

Resolution

Sentiment Team: The issue was resolved in PR#222.

L-01 | Accrue Before feeRecipient Update

Category	Severity	Location	Status
Logical Error	● Low	SuperPool.sol: 325-326	Acknowledged

Description

If the owner of the SuperPool lost access to the feeRecipient wallet and tries to change it with the setFeeRecipient function it first accrues interest. This would lead to further loss in this case.

Recommendation

Update the feeRecipient before calling the accrue function.

Resolution

Sentiment Team: Acknowledged.

L-02 | Allocators Can Bypass Pool Caps

Category	Severity	Location	Status
Validation	● Low	SuperPool.sol: 345-360	Resolved

Description

The pool caps are not checked in the reallocate function. Therefore allocators can bypass the pool caps set by the owner of the SuperPool.

Recommendation

Check the pool caps in the reallocate function.

Resolution

Sentiment Team: The issue was resolved in PR#231.

L-03 | approve Race Condition

Category	Severity	Location	Status
Logical Error	● Low	ERC6909.sol: 57	Resolved

Description

The ERC6909 contract is vulnerable to a well-known race condition in the approve function:
• User approves 100 tokens to a spender
• The user wants to increase the allowance to by 50 tokens and calls approve with 150 tokens
• Spender front runs the call and spends 100 tokens
• The user's approve call goes through and the allowance is set to 150 tokens
• Spender spends 150 tokens
Therefore the user wanted to allow the spender to spend 150 tokens but the spender was able to spend 250 tokens.

Recommendation

Implement increaseAllowance and decreaseAllowance functions.

Resolution

Sentiment Team: The issue was resolved in PR#256.

L-04 | Missing Zero Assets Check In redeem

Category	Severity	Location	Status
Validation	● Low	SuperPool.sol: 242	Resolved

Description

In the `redeem` function of the `SuperPool` contract, it is not checked if the calculated `assets` amount from `previewRedeem` is 0. As `previewRedeem` rounds down it could be possible that a non-zero share amount is burned from the user but the user does not receive any assets in return.

Recommendation

Revert if the calculated `assets` amount is 0.

Resolution

Sentiment Team: The issue was resolved in PR#241.

L-05 | Unused Events

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Resolved

Description

The PoolOwnerSet event in the Pool contract and the PoolAdded event in the SuperPool contract are not being emitted.

Recommendation

It is recommended to either remove the unused events or to use them in appropriate functions.

Resolution

Sentiment Team: The issue was resolved in PR#257.

L-06 | Typo

Category	Severity	Location	Status
Typo	● Low	Global	Resolved

Description

There are some typos in the codebase.
PositionManager contract L30: “…liqudiator…” should be “…liquidator…”
PositionManager contract L82: “SENIMENT” should be “SENTIMENT”
RiskEngine contract L143: “…witihin…” should be “…within…”

Recommendation

We recommend updating the mentioned words in the codebase.

Resolution

Sentiment Team: The issue was resolved in PR#258.

L-07 | Registry Address Can Be Immutable

Category	Severity	Location	Status
Optimization	● Low	RiskEngine: 48	Resolved

Description

The registry variable in the RiskEngine contract is assigned in the constructor and there isn't any function for updating the 'registry' address in the contract.

Recommendation

Consider making the variable immutable.

Resolution

Sentiment Team: The issue was resolved in PR#259.

L-08 | Mismatch Between Code And Developer Comment

Category	Severity	Location	Status
Optimization	● Low	Pool.sol: 337-338	Resolved

Description

In the borrow function of the Pool contract, the developer's comments state that minted shares should round up. However, the actual code rounds down because it uses `convertToShares`.

As rounding down is in favour of the borrower in this case. The excess debt is socialized among all other borrowers:
• Borrower borrowers X amount of funds
• The `convertToShares` function calculates that the given amount to borrow equals "Y + 0.9" debt shares
• As solidity rounds down the user only gets Y debt shares
• Therefore the borrowed amount that equals 0.9 debt shares is socialized among all borrowers

Recommendation

Round up (against the borrower).

Resolution

Sentiment Team: The issue was resolved in PR#245.

L-09 | Consider Using Ownable2Step

Category	Severity	Location	Status
Optimization	● Low	Global	Acknowledged

Description

The protocol utilizes the Ownable library from OpenZeppelin, which does not include a 2-step ownership change implementation. This could potentially lead to undesired situations such as contracts being left without an owner after an incorrect update.

Recommendation

It is advisable to consider utilizing the Ownable2Step library.

Resolution

Sentiment Team: Acknowledged.

L-10 | Superfluous Code

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Resolved

Description

In the accrue function of the Pool contract, the following lines are used twice:

```
// Store a timestamp for this accrue() call
// Used to compute the pending interest next time accrue() is called
pool.lastUpdated = uint128(block.timestamp);
```

Additionally, calling OwnableUpgradeable.__Ownable_init() right before the internal _transferOwnership(owner_) in the Pool contract and PositionManager contract is unnecessary, as the latter call will override the initialization calls.

Recommendation

Consider removing superfluous code to optimize the efficiency of the contracts.

Resolution

Sentiment Team: The issue was resolved in PR#260.

L-11 | Can't Liquidate When Price Is Stale

Category	Severity	Location	Status
Oracle	● Low	ChainlinkEthOracle.sol: 100	Acknowledged

Description

Liquidations use the health check to validate if the position can be liquidated. If the price is stale, health check reverts, preventing the liquidation. Protocol might want to liquidate positions, no matter if price is stale, to avoid bad debt.

Recommendation

If this is not the intended behavior, consider bypassing the stale price check only for the liquidation process

Resolution

Sentiment Team: Acknowledged.

L-12 | Incorrect Pool Can Be Removed From Queue

Category	Severity	Location	Status
Logical Error	● Low	SuperPool.sol: 497	Resolved

Description

The pool owner can remove a pool from the queue as long as there are no assets deposited. The `_removeFromQueue` function contains a loop that searches for the index of the pool to remove, and then removes the first item, shifting all subsequent items down.

While it is unlikely that the first loop will not find the poolId in the queue, `toRemoveIdx` may remain uninitialized, resulting in the removal of the first pool from the queue.

Recommendation

Implement an early return inside the first loop if the index is not found.

Resolution

Sentiment Team: The issue was resolved in PR#261.

L-13 | PositionManager Allows Free Flashloans

Category	Severity	Location	Status
Logical Error	● Low	PositionManager.sol: 227	Acknowledged

Description

`PositionManager` allows users to batch actions using `processBatch`, with the condition that the position must be healthy when the transaction ends. This opens up the possibility of free Flash Loans, where users can borrow, transfer out the funds, use them outside of the protocol, and finally repay them to make sure the position is healthy.

Recommendation

Although this will considerably increase the gas cost of the transaction, health can be checked in every action to ensure users can't borrow without depositing collateral first.

Resolution

Sentiment Team: Acknowledged.

L-14 | Superfluous OnlyOwner Modifier

Category	Severity	Location	Status
Logical Error	● Low	SuperPool.sol: 466	Resolved

Description

_removePool has an onlyOwner modifier, although this is an internal function that can only be called from the setPoolCap which already has the access control.

Recommendation

Remove the onlyOwner modifier from the _removePool function.

Resolution

Sentiment Team: The issue was resolved in PR#261.

L-15 | Liquidation Discount Is Incorrect

Category	Severity	Location	Status
Math	● Low	RiskModule.sol: 116	Resolved

Description

Liquidators can get borrowers collateral at a discounted price. According to deployment script it will be 20%. Maximum seized collateral amount is calculated with this formula: `debtRepaidValue.mulDiv((1e18 + LIQUIDATION_DISCOUNT), 1e18)`

However, the correct way to implement this discount should be `debtRepaidValue.mulDiv(1e18, (1e18 - LIQUIDATION_DISCOUNT))` For example, getting $100 worth of asset with 20% discount means that the user should pay 80$. But in the current implementation, the user gets $120 worth of asset by paying $100, effectively making the discount 16.66%.

Recommendation

Consider implementing the discount by decreasing the denominator instead of increasing the numerator. Otherwise, document this behavior as accepted.

Resolution

Sentiment Team: The issue was resolved in PR#262.

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