GA GUARDIAN

Olympus

Convertible Deposits

Security Assessment

September 5th, 2025

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Curiousapple, Vladamir Zotov, Wafflemakr

Client Firm Olympus

Final Report Date September 5, 2025

Audit Summary

Olympus engaged Guardian to review the security of their Olympus V3 Convertible Deposits. From the 28th of July to the 18th of August, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian assigns a Confidence Ranking of 2 to the protocol. Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. Guardian strongly advises that the protocol undergo a full follow-up audit at a finalized and fully remediated commit before any mainnet deployment. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **Ethereum**

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

📊 Code coverage & PoC test suite: https://github.com/GuardianOrg/olympus-v3-olympus-team1

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Invariants Assessed 9

Findings & Resolutions 14

Addendum

Disclaimer 84

About Guardian 85

Project Overview

Project Summary

Project Name	Olympus
Language	Solidity
Codebase	https://github.com/OlympusDAO/olympus-v3
Commit(s)	Initial commit: 08e562e6d4e4f3ef7bdf5553de99574403cfdcb9 Final commit: b598f1b927f869803ef7240dc08cb70924bb27dc

Audit Summary

Delivery Date	September 5, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	8	0	0	2	0	6
Medium	11	0	0	5	0	6
Low	30	0	0	8	0	22
Info	15	0	0	4	0	11

Audit Scope & Methodology

Scope and details:

contract,source,total,comment
olympus-v3/src/bases/BasePeriodicTaskManager.sol,94,192,64
olympus-v3/src/bases/BaseAssetManager.sol,118,245,88
olympus-v3/src/policies/ReserveWrapper.sol,70,140,36
olympus-v3/src/policies/Heart.sol,116,218,60
olympus-v3/src/policies/EmissionManager.sol,326,564,142
olympus-v3/src/libraries/Uint2Str.sol,22,27,4
olympus-v3/src/libraries/Timestamp.sol,33,43,4
olympus-v3/src/libraries/ERC6909Wrappable.sol,125,221,54
olympus-v3/src/libraries/DecimalString.sol,51,87,20
olympus-v3/src/libraries/CloneableReceiptToken.sol,36,90,39
olympus-v3/src/libraries/AddressStorageArray.sol,23,47,14
olympus-v3/src/external/clones/CloneERC20.sol,64,125,26
olympus-v3/src/modules/DEPOS/PositionTokenRenderer.sol,146,184,20
olympus-v3/src/modules/DEPOS/OlympusDepositPositionManager.sol,207,432,143
olympus-v3/src/modules/DEPOS/DEPOS.v1.sol,11,37,16
olympus-v3/src/policies/deposits/YieldDepositFacility.sol,257,426,98
olympus-v3/src/policies/deposits/DepositRedemptionVault.sol,434,764,180
olympus-v3/src/policies/deposits/DepositManager.sol,295,548,146
olympus-v3/src/policies/deposits/ConvertibleDepositFacility.sol,212,379,102
olympus-v3/src/policies/deposits/ConvertibleDepositAuctioneer.sol,362,749,245
olympus-v3/src/policies/deposits/BaseDepositFacility.sol,196,337,75
source count: {
total: 5855,
source: 3198,
comment: 1576,
single: 1564,
block: 12,
mixed: 24,
empty: 1105,
todo: 1,
blockEmpty: 0,
commentToSourceRatio: 0.49280800500312694
}

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

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 Olympus, fuzz-testing 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 fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
CDA-01	Day State convertible increase by ohmOut	✅	✅	✅	10M+
CDA-02	Day State convertible should not exceed auction target per deposit period	✅	✅	✅	10M+
CDA-03	Tick price should be above auction min price for each deposit period	✅	✅	✅	10M+
CDA-04	Incorrect Position details check (operator asset remainingDeposit conversionPrice)	✅	✅	✅	10M+
CDA-05	Tick capacity should be less than or equal to current tick size for enabled deposit period	✅	✅	✅	10M+
CDA-06	Tick price should increase or remain the same after successful bid	✅	✅	✅	10M+
CDA-07	Tick capacity should decrease or remain the same after successful bid	✅	✅	✅	10M+
CDA-08	Position conversion price should be greater than or equal to current tick price	✅	✅	✅	10M+
CDA-09	Bid previewed ohmOut should be equal to ohmOut	✅	✅	✅	10M+
CDA-10	Tick size should be 1 when target is 0	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
CDA-11	Deposit period should be enabled if exists in array	✅	✅	✅	10M+
CDF-01	User receipt token received = deposited reserve token amount	✅	✅	✅	10M+
CDF-02	CDF-created position should never be considered a YDF position	✅	✅	✅	10M+
CDF-03	AssetLiabilities should be equal to before + receipt token amount received for deposit	✅	✅	✅	10M+
CDF-04	DepositedSharesInAssets should be equal to before + receipt token amount received for deposit	✅	✅	✅	10M+
CDF-05	User should have correct token balance after conversion (receipt token and ohm token)	✅	✅	✅	10M+
CDF-06	User ohm balance change after convert = preview convert value	✅	✅	✅	10M+
CDF-07	Total conversions should never be more than sold capacity	✅	❌	✅	10M+
DRV-01	Protocol should remain solvent after claimDefaultedLoan (assets > liabilities)	✅	✅	✅	10M+
DRV-02	Loan should be marked as defaulted after claimDefaultedLoan	✅	✅	✅	10M+
DRV-03	Available deposits should not change after defaulted loan	✅	✅	✅	10M+
DRV-04	After claimDefaultedLoan function getAvailableDeposits should return the same value as before	✅	✅	✅	10M+
DRV-05	After claimDefaultedLoan function getAvailableDeposits should return the same value as before (duplicate)	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
DRV-06	Loan details should be equal to previewed	✅	✅	❌	10M+
DRV-07	Interest should be equal to previewed	✅	✅	✅	10M+
DRV-08	User token balance should reflect redemption amount after finishing	✅	✅	✅	10M+
DRV-09	Redemption amount 0 after finishing	✅	✅	✅	10M+
DRV-10	Borrow principal amount always 0 when attempting borrow against finished redemption	✅	✅	✅	10M+
DRV-11	User token balance should reflect redemption cancellation amount	✅	✅	✅	10M+
DRV-12	Redemption object amount should decrease by cancellation amount	✅	✅	✅	10M+
DRV-13	User token balance should reflect start redemption amount	✅	✅	✅	10M+
DRV-14	Redemption object amount should increase by start redemption amount	✅	✅	✅	10M+
DRV-15	User reserve balance should decrease by at least repayment amount	✅	✅	✅	10M+
DRV-16	Either interest or principal should decrease after repayment	✅	✅	✅	10M+
DRV-17	Treasury balance should increase after repayment	✅	✅	✅	10M+
DRV-18	Loan should still exist after extension	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
DRV-19	Loan due date should be strictly later after extension	✅	✅	✅	10M+
DRV-20	Treasury balance should increase after loan extension	✅	✅	✅	10M+
DRV-21	Available deposits should not change post-borrow	✅	✅	✅	10M+
DRV-22	User reserve balance should increase by borrow amount	✅	✅	✅	10M+
GLOB-01	User should never have an active loan without an active redemption request	✅	✅	✅	10M+
GLOB-02	DepositManager solvency check (depositedSharesInAssets + borrowedAmount > assetLiabilities)	✅	❌	❌	10M+
GLOB-03	Asset liabilities should never exceed receipt token supply	✅	✅	✅	10M+
GLOB-04	Receipt supply should be less than reserve token balance in vaults and borrowed	✅	✅	✅	10M+
GLOB-05	Committed deposits should never exceed sharesInAssets	✅	❌	✅	10M+
GLOB-06	Balance of receipt tokens in DRV should be at least sum of all redemption amounts	✅	✅	✅	10M+
GLOB-07	DRV operator committed deposits should equal sum of all redemption amounts	✅	✅	✅	10M+
REVERT-01	Should not panic underflow/overflow	✅	❌	❌	10M+
YDF-01	PositionLastYieldConversionRate should be greater than 0	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
YDF-02	User receipt token received (ERC6909 DepositManager:_mint token) = deposited reserve token amount	✅	✅	✅	10M+
YDF-03	YDF-created position should never be considered a CDF position	✅	✅	✅	10M+
YDF-04	AssetLiabilities should be equal to before + receipt token amount received for deposit	✅	✅	✅	10M+
YDT-05	DepositManager balance should be decreased by total yield amount when no vault configured	✅	✅	✅	10M+
YDT-06	Yield claimed by users should equal reduction in total yield accrued	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Incorrect Scaling Overmints OHM To User	Logical Error	Critical	Resolved
H-01	Reclaims Can Force Insolvent Deposit Managers	Logical Error	High	Resolved
H-02	Yield Claim DoS'ed By Share Inconsistencies	Logical Error	High	Acknowledged
H-03	Split Positions Cannot Claim Yield	Logical Error	High	Resolved
H-04	Facilities Can Steal Yield From Each Other	Logical Error	High	Resolved
H-05	Auctioneer Incompatible	Logical Error	High	Resolved
H-06	Heart Beat DoS'ed On Zero Emissions	DoS	High	Resolved
H-07	User Can Game Rate Received From Vault	Gaming	High	Resolved
H-08	Insolvency Due To Fixed-Amount Withdrawals	Logical Error	High	Acknowledged
M-01	Griefing Receipt Token Conversion	Access Control	Medium	Resolved
M-02	Withdraw Rounding Prevents Yield Claims	Logical Error	Medium	Resolved
M-03	Emissions Not Adjusted	Logical Error	Medium	Acknowledged
M-04	Keepers Not Incentivized To Quickly Beat	Warning	Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-05	Yield Lost For Expired Positions	Unexpected Behavior	Medium	Acknowledged
M-06	Converted Reserves Do Not Increase Backing	Logical Error	Medium	Acknowledged
M-07	First Depositor Attack Affects Redemptions	Logical Error	Medium	Acknowledged
M-08	Capacity Can Grow Incorrectly	Unexpected Behavior	Medium	Resolved
M-09	Capacity Calculation Incorrect	Logical Error	Medium	Resolved
M-10	Target Update Without LastUpdate Adjustment	Logical Error	Medium	Resolved
M-11	Inconsistent Global VS Per-Period Auction Parameters	Logical Error	Medium	Resolved
L-01	Risk Of Underflow With Multiple Operators	Warning	Low	Resolved
L-02	Insolvency When Large Yield	Logical Error	Low	Resolved
L-03	Protocol Risk When Interest Exceeds Buffer	Warning	Low	Acknowledged
L-04	User Can Delay Reclaims	Censoring	Low	Acknowledged
L-05	Default VS Reclaim Gaming	Warning	Low	Resolved
L-06	Delayed Heartbeat Affects Auction Tracking	Warning	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-07	Price Decays Faster Than Expected	Unexpected Behavior	Low	Resolved
L-08	Deposit Period Enabled Before Policy	Warning	Low	Resolved
L-09	Re-enabling Loses Tick Data	Warning	Low	Resolved
L-10	Setting Tracking Period Wipes Existing Results	Warning	Low	Resolved
L-11	Failed Bond Market Creation Derails Auction	Warning	Low	Resolved
L-12	Disabling Policy Loses Auction Results	Warning	Low	Acknowledged
L-13	Allowance Decrease Frontrunning	Warning	Low	Acknowledged
L-14	Task Execution Succeeds	Warning	Low	Resolved
L-15	Max Yield Cannot Be Claimed	Warning	Low	Resolved
L-16	Cross-Contract Reentrancy Risk	Warning	Low	Resolved
L-17	Parameters Should Be Configurable	Warning	Low	Resolved
L-18	Lack Of Validation On Withdrawal	Warning	Low	Resolved
L-19	Missing Validation For previewReclaim	Validation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-20	Initial Auction Parameters Not Validated	Validation	Low	Resolved
L-21	Risk-Free Bids	Warning	Low	Acknowledged
L-22	Asymmetric Yield Across Multiple Claims	Gaming	Low	Resolved
L-23	ERC-4626 Deposit VS PreviewRedeem Invariant	Superfluous Code	Low	Acknowledged
L-24	Vault Validation Warnings	Validation	Low	Acknowledged
L-25	Asymmetric Remaining Deposit	Warning	Low	Resolved
L-26	Lack Of Slippage Protection On Bids	MEV	Low	Resolved
L-27	Split Griefing Increases Gas Cost	Gas Griefing	Low	Resolved
L-28	Potential Superior Loans	Configuration	Low	Resolved
L-29	Handle Loan Functions	Validation	Low	Resolved
L-30	Borrowing More Is Incentivized For Defaults	Warning	Low	Acknowledged
I-01	YDF conversionPrice Should Use Constant	Best Practices	Info	Resolved
I-02	Empty Position Id List Validation Missing	Best Practices	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-03	depositAmount Calculation Rounds	Rounding	Info	Resolved
I-04	Unwrapped Receipt Necessary For Redemption	Documentation	Info	Resolved
I-05	Actual Amount Ignored During Bids	Informational	Info	Resolved
I-06	Redundant Boolean Check	Best Practices	Info	Resolved
I-07	Using Magic Numbers	Best Practices	Info	Resolved
I-08	Superfluous Permission Request	Superfluous Code	Info	Resolved
I-09	Borrows Pay Full Interest Regardless Of Payback	Documentation	Info	Resolved
I-10	Newly Minted OHM Not Counted In Supply	Warning	Info	Acknowledged
I-11	Convertible Deposit Facility (CDF)	Logical Error	Info	Acknowledged
I-12	Unnecessary Day State Storage Read	Gas Optimization	Info	Resolved
I-13	Redundant Setting Of Asset And Period	Logical Error	Info	Acknowledged
I-14	Unnecessary While Loop	Gas Optimization	Info	Acknowledged
I-15	Lack Of SafeTransfer	Best Practices	Info	Resolved

C-01 | Incorrect Scaling Overmints OHM To User

Category	Severity	Location	Status
Logical Error	● Critical	ConvertibleDepositFacility.sol: 221	Resolved

Description PoC

In the conversion logic, `convertedTokenOut = FullMath.mulDiv(amount_, 10 * IERC20(currentAsset).decimals(), position.conversionPrice)` multiplies `amount_` (reserve token amount) by the reserve token's decimals (e.g., 18 for USDS).

However, `position.conversionPrice` is calculated as `depositIn.mulDivUp(_ohmScale, ohmOut)` where `_ohmScale = 1e9` for OHM's 9 decimals.

This introduces an extra factor of `10 * reserveDecimals / 10 * ohmDecimals` (e.g., 10^9 for reserve=18, OHM=9), resulting in `convertedTokenOut` being massively inflated.

For reserve tokens with 18 decimals, users receive 1 billion times more OHM than intended which causes catastrophic OHM supply dilution and violates the principle that each OHM is backed appropriately by treasury assets.

Recommendation

Multiply by `_ohmScale` instead of `10 * IERC20(currentAsset).decimals()` in `_previewConvert`.

Resolution

Olympus Team: The issue was resolved in PR#98.

H-01 | Reclaims Can Force Insolvent Deposit Managers

Category	Severity	Location	Status
Logical Error	High	YieldDepositFacility.sol: 325	Resolved

Description PoC

A user can call function reclaimFor to reclaim their entire deposit, but this does not update the user's DEPOS position, allowing users to claim yield on non-existent assets after reclaiming.

This vector allows for the following scenario:

(1) Alice deposits 1e18 reserve tokens through createPosition
(2) Alice then reclaims her entire deposit, receiving a discounted amount. The DepositManager now holds 0 vault shares, but Alice's yield-earning position remains.
(3) Bob then proceeds to deposit into the vault through createPosition.
(4) Yield accrues and a snapshot is taken.
(5) Alice claims yield (claimYield) with her phantom position, receiving some reserve tokens.
(6) Bob attempts to claim yield, but because Alice already withdrew some assets, Bob's claim yield attempt reverts with DepositManager_Insolvent. Bob cannot claim the yield he has earned with the assets he deposited into the vault and that are generating yield in the vault.

Ultimately, Alice is able to prevent other users from claiming their yield by forcing insolvency through a reclaim, and also claim yield without having actively deposited assets in the vault.

Also note that the smaller the discount is configured (reclaim rate can be as high as 100%), the faster Alice will earn enough yield to overcome any immediate loss.

Recommendation

Adjust the remaining deposit of existing positions to reflect the withdrawal. Furthermore, ensure the reclaim rate is large enough to discourage malicious behavior.

Resolution

Olympus Team: The issue was resolved in PR#86.

H-02 | Yield Claim DoS'ed By Share Inconsistencies

Category	Severity	Location	Status
Logical Error	High	YieldDepositFacility.sol: 321	Acknowledged

Description PoC

Users are able to create a position in YieldDepositFacility in order to claim the yield from an ERC4626 vault strategy, without the ability to convert to OHM.

Furthermore, users holding receipt tokens can start a redemption and borrow against it, withdrawing funds from the ERC4626 vault.

The main issue arises when users borrow using the YieldDepositFacility, as it reduces the active shares for the operator, while the facility' user's lastShares will still be using the full deposited value (sum of lastShares of users is greater than the real active shares)

This will impact the claimable yield calculation as users will try to claim more yield than what the max claimable yield allowed in the DepositManager for the YieldDepositFacility , DoS'ing users when calling claimYield.

Although this issue describes only YieldDepositFacility users, the same issue will apply if ConvertibleDepositFacility users redeem and borrow using the YieldDepositFacility.

Recommendation

The solution requires a major refactor, as there is a discrepancy between the individual user shares calculation in YieldDepositFacility vs the actual shares of the facility in the DepositManager.

When funds are borrowed out from the vault, these will not accrue yield, which should be reflected in the _previewClaimYield calculation.

Resolution

Olympus Team: Acknowledged.

H-03 | Split Positions Cannot Claim Yield

Category	Severity	Location	Status
Logical Error	High	OlympusDepositPositionManager.sol: 254	Resolved

Description PoC

When a position is created in `YieldDepositFacility`, `positionLastYieldConversionRate` is set to ensure yield is claimed only from the deposit time onward.

However, when splitting a position in `OlympusDepositPositionManager::split`, the new position's `positionLastYieldConversionRate` is not set, leaving it at 0.

This causes a division-by-zero revert in `claimYield` when calculating `lastShares = mulDiv(remainingDeposit, decimals, lastSnapshotRate = 0)`, blocking yield claims for the new position.

Recommendation

Stamp `positionLastYieldConversionRate` when splitting for YDF positions.

Resolution

Olympus Team: The issue was resolved in PR#90.

H-04 | Facilities Can Steal Yield From Each Other

Category	Severity	Location	Status
Logical Error	High	DepositRedemptionVault.sol: 220	Resolved

Description PoC

Receipt tokens are fungible (receipt tokens created by one operator can be reclaimed through another). Therefore, `ConvertibleDepositFacility` depositors can reclaim their tokens using the `YieldDepositFacility`.

However, reclaiming receipt tokens reduce the amount of yield earned by the facility as it redeems shares from the operator. In extreme cases, the `YieldDepositFacility` can be left with zero operator shares, preventing users to claim yield.

On the other side, claiming yield from the `ConvertibleDepositFacility` will be possible, but it is transferred to the treasury.

Recommendation

Prevent users from the `ConvertibleDepositFacility` to be able to reclaim using a different facility (`YieldDepositFacility`).

Resolution

Olympus Team: The issue was resolved in PR#86.

H-05 | Auctioneer Incompatible

Category	Severity	Location	Status
Logical Error	High	ConvertibleDepositAuctioneer.sol	Resolved

Description PoC

The Auctioneer calculates the OHM output as `convertibleAmount = deposit_.mulDiv(_ohmScale, price_)`; This assumes price has deposit token decimal precision, but the price provided through `PRICE.getCurrentPrice()` in the `EmissionManager` is 18 decimals.

Non-18 decimal deposit tokens are incompatible with the Auctioneer since the resulting `convertibleAmount` is no longer a 9 decimal OHM output amount.

This will lead to the output amount to be larger than expected and absorb more capacity than expected, or truncate down to 0 when dealing with smaller decimal tokens such as USDC and revert.

Recommendation

Scale the price to the deposit token's decimals.

Resolution

Olympus Team: The issue was resolved in PR#99.

H-06 | Heart Beat DoS'ed On Zero Emissions

Category	Severity	Location	Status
DoS	High	ConvertibleDepositAuctioneer.sol: 605	Resolved

Description PoC

The heart beat executes period tasks, including the EmissionManager.execute, where the auction parameters are set.

In case of a low premium (or < 100%, which is the current case based on price and backing per ohm), the emission will be zero.

Therefore, both target and tickSize params are zero during setAuctionParameters, values not allowed in the auctioneer.

Two main issues here:

• heart.beat() reverts, as periodic task execution reverts if one of the tasks fails.

• In case there is a live auction and premium drops for the next execution, ticks won't be updated and auction results not stored.

Recommendation

Make sure periodic tasks do not revert during their execution, preventing complete DoS on the heart beat. Additionally, due to the fact that emissions are zero, auctioneer should be disabled or paused to prevent more bids. Make sure day state and auction results are updated accordingly.

Resolution

Olympus Team: The issue was resolved in PR#104.

H-07 | User Can Game Rate Received From Vault

Category	Severity	Location	Status
Gaming	High	YieldDepositFacility.sol	Resolved

Description PoC

In the claimYield function for expired positions, users can provide a timestampHint_ to select a historical vault conversion rate from vaultRateSnapshots. The only validation is that the hint ≤ expiry, with no check for proximity to expiry or against drops in rate.

If a vault's rate peaks mid-period and then falls, users can hint to the peak, inflating yield calculations beyond the actual value at expiry. Consequently, more assets are withdrawn from the vault than expected, which effectively steals the gain of other users.

Consider the following scenario:

(1) Alice creates a position, the vault accrues yield and a snapshot is taken at time X.

(2) The strategy loses funds, the share value of the vault decreases, and a snapshot is taken at (X + 8) hours.

(3) Bob then creates a position and a snapshot is taken at (X + 16) hours.

(4) Time passes so Bob's position is now past expiry.

(5) Bob claims yield with a hint at time X, which is before the he even made a position, effectively stealing any yield Alice earned in her duration during the vault.

Recommendation

Enforce timestampHint_ is as close to the expiry as possible.

Resolution

Olympus Team: The issue was resolved in PR#84.

H-08 | Insolvency Due To Fixed-Amount Withdrawals

Category	Severity	Location	Status
Logical Error	High	BaseAssetManager.sol: 127-128	Acknowledged

Description

Olympus intends to support vaults that can realize losses — in other words, vaults where the value of a user's deposit may decrease after the time of deposit. However, the current code is not equipped to handle such scenarios.

Example – Base Asset Manager

• A user deposits x amount into the vault. Olympus records the deposited asset amount and mints recipient tokens equivalent to that amount.

• When the user withdraws, Olympus allows them to redeem shares equivalent to their initial deposit amount.

• If the vault has incurred losses, this logic allows the user to withdraw more than their proportional share of the vault's current value. This could lead to a bank run and insolvent positions.

Recommendation

We expect that there may be multiple areas in the code where loss-incurring vaults can cause similar issues. Addressing this in a single place may not be sufficient. A broader review and multiple code changes are likely required to ensure proper handling of loss scenarios.

Resolution

Olympus Team: Acknowledged.

M-01 | Griefing Receipt Token Conversion

Category	Severity	Location	Status
Access Control	Medium	ERC6909Wrappable.sol: 167	Resolved

Description PoC

In order to convert a receipt token into OHM, users first need to approve the DepositManager to spend the tokens, enforced during _burn. This approval is also needed for wrap or unwrap tokens from ERC6909 to ERC20.

However, malicious users can grief the convert transaction by front running and wrapping/unwrapping receipt tokens. The convert will now fail with insufficient allowance as it was spent by the malicious actor.

This is possible due to the fact that wrap and unwrap do not have access control, and allow any user to perform the action on behalf of other, as long as the allowance was given. The most impact occurs when the attacker repeats this process, until the user's position expires.

In case of a smart contract user, deployed with max approval to DepositManager to save gas, receipt tokens can be permanently locked if the ERC6909 tokens are suddenly wrapped into ERC20, without a way to handle these tokens or approval functionality.

Recommendation

A combination of better documentation on approvals and stricter access control for wrap/unwrap functionality is recommended to mitigate this risk. Alternatively, consider using msg.sender instead of onBehalfOf for the wrap and unwrap functions.

Resolution

Olympus Team: The issue was resolved in PR#89.

M-02 | Withdraw Rounding Prevents Yield Claims

Category	Severity	Location	Status
Logical Error	Medium	DepositManager.sol	Resolved

Description PoC

When claiming yield, the assets are withdrawn from the vault through the deposit manager: `(, uint256 actualAmount) = _withdrawAsset(asset_, recipient_, amount_);`

`ERC4626` `withdraw` ends up withdrawing the exact number of assets requested, but it also rounds up the number of shares that are burnt.

Because more shares are burnt, `depositedSharesInAssets` is decreased slightly more than expected, making it more likely that the vault is insolvent: `_assetLiabilities[assetLiabilitiesKey] > depositedSharesInAssets + borrowedAmount`.

Ultimately a user attempts to claim yield but is prevented by as little as 1 wei of insolvency due to the extra drop in `depositedSharesInAssets`.

This issue also occurs with `borrowAgainstRedemption` as it triggers `withdraw` as well, and the deficit will continue to compound as more users submit borrows.

Recommendation

Consider adjusting the withdrawal mechanism such that the user's requested amount to withdraw is rounded down with existing `ERC4626` functionality.

The user may now receive less than than requested for redemption and this must be appropriately documented (redemptions are not exactly 1:1 anymore).

Resolution

Olympus Team: The issue was resolved in PR#91.

M-03 | Emissions Not Adjusted

Category	Severity	Location	Status
Logical Error	Medium	EmissionManager.sol	Acknowledged

Description

The EmissionManager calculates emissions based on supply (getSupply() = gohm.totalSupply() * gohm.index() / 10 * _gohmDecimals) and premium, assuming a single reserve token.

If multiple EmissionManagers are deployed for different reserves (e.g. USDC through separate auctioneers), each independently computes and mints OHM emissions without aggregating across all managers.

This duplicates emissions, as getNextEmission() in one manager doesn't account for emissions from others, over-minting OHM and diluting supply. For example 2 EmissionManagers/Auctioneers can double the expected OHM emission.

Recommendation

Consider accounting for multiple managers within getNextEmission. Otherwise, utilize only one EmissionManager and Auctioneer at a time.

Resolution

Olympus Team: Acknowledged.

M-04 | Keepers Not Incentivized To Quickly Beat

Category	Severity	Location	Status
Warning	Medium	Heart.sol	Acknowledged

Description

Within the Heart, the `currentReward()` function pays 0 at the exact beat boundary (`current time = lastBeat + frequency()`), then ramps linearly to `maxReward` over `min(auctionDuration, frequency())`.

Since `beat()` reverts before the boundary and is valid at the boundary with 0 reward, rational keepers are incentivized to wait past the boundary to earn a positive payout.

Especially if transaction gas fees are higher, it only makes sense for keepers to wait to get a positive reward that exceeds the gas fee. This leads to execution delay, delayed price moving average updates, etc.

Recommendation

Consider a base reward at the valid time boundary or run protocol keepers.

Resolution

Olympus Team: Acknowledged.

M-05 | Yield Lost For Expired Positions

Category	Severity	Location	Status
Unexpected Behavior	Medium	YieldDepositFacility.sol: 326	Acknowledged

Description PoC

Creating positions in `YieldDepositFacility` gives user the ability to claim yield from the vault, until the position expires. Once expired, users will only be able to claim up to the expiring timestamp. These expired positions will still have assets deposited in the vault, accruing yield.

However, the yield generated is not distributed to existing users, even after expired positions redeem or reclaim (exit). This pending yield will be left in the `DepositManager` even after all facility positions exit, with no way to claim it. `YieldDepositFacility` will end up in an invalid state:

- operator liabilites = 0
- shares in assets > 0
- max claim yield > 0

Recommendation

Consider adding an admin function to claim yield during emergencies or when operator liabilities becomes 0 (no user deposits and available yield to claim).

Resolution

Olympus Team: Acknowledged.

M-06 | Converted Reserves Do Not Increase Backing

Category	Severity	Location	Status
Logical Error	Medium	ConvertibleDepositFacility.sol: 289	Acknowledged

Description

Teller will make a `callBack` to `EmissionManager` in order to update the backing price based on new supply and reserves added.

On the other side, the `ConvertibleDepositFacility.convert` function will mint OHM to user, after withdrawing reserves from the vault into the treasury. These reserves will later be deposited into the vault during a periodic task execution in `ReserveWrapper`.

However, this convert flow does not update the backing price before depositing the received reserves and minting the output amount of OHM, compared to teller purchases during `callback` execution.

Recommendation

Consider updating backing OHM price when converting notes in the `ConvertibleDepositFacility`.

Resolution

Olympus Team: Acknowledged.

M-07 | First Depositor Attack Affects Redemptions

Category	Severity	Location	Status
Logical Error	Medium	DepositManager.sol	Acknowledged

Description PoC

Users interacting with a facility are potentially vulnerable to the first depositor inflation attack depending on the underlying vault.

Consider a vault that allows for donations (using `balanceOf`):

• Bob deposits 1 wei USDS into CDF and gets 1 wei `rUSDS`
• Alice deposits 20,000 USDS into CDF, expecting 20,000 `rUSDS`
• Bob frontruns and transfers 10,000 USDS directly to the vault
• Alice is minted 15,000 rUSDS but only 1 wei of shares is minted for the `DepositManager` for the CDF's operator shares
• CDF operator shares is only 2 wei now
• Alice redeems 15,000 `rUSDS` and withdraws 15,000 USDS, and shares for the CDF operator goes from 2 wei to 1 wei.
• Bob redeems his 1 wei USDS and shares is now 0, but the Vault contains 15,000 USDS.
• Bob deposits 15,000 USDS again, receiving 30,000 `rUSDS` due to the vault's sitting 15,000 USDS
• Bob redeems 30,000 `rUSDS`, withdrawing 30,000 USDS, recovering the vault's leftover assets.
• Bob invested 1 wei + 10,000 USDS (donation) + 15,000 USDS (final deposit) and walked away with 30,000 USDS, making 5,000 USDS in profit while Alice lost $5,000 USDS
• Consequently, the `actualAmount` receipt token setup does not prevent the first depositor inflation attack but adds extra steps necessary to execute it.

Note that the severity is of course subject to the length of redemption periods and whether the underlying vault is susceptible to the typical first depositor inflation attack.

Recommendation

Ensure the underlying vault uses the OZ virtual offset or dead shares are minted initially.

Resolution

Olympus Team: Acknowledged.

M-08 | Capacity Can Grow Incorrectly

Category	Severity	Location	Status
Unexpected Behavior	Medium	ConvertibleDepositAuctioneer.sol	Resolved

Description

The `_getCurrentTick` function computes the `capacityToAdd` amount accounting for the number of enabled deposit periods such that the configured aggregate auction target is met per day in capacity additions.

However in the `enableDepositPeriod` and `disableDepositPeriod` functions there is no update to ensure that the latest capacity changes have been reflected up to the current timestamp.

This allows for technically incorrect capacity changes to occur around the enabling and disabling of deposit period.

For example, consider the following scenario:

- There are 2 deposit periods, A & B, enabled for the auctioneer
- The daily target for all auctions in aggregate is 100 tokens
- it has been 3 hours since the last capacity update
- 3 * 100 / 24 = 12.5 total capacity which should be added to auctions A & B through an increase of 6.25 capacity to both
- A third deposit period is enabled, C
- A deposit occurs for deposit period A right after the deposit period C is enabled
- Now the 12.5 total capacity which has accrued is split amongst deposit periods A, B, and C — giving A only a 4.167 token allocation when it should have received 6.25

Recommendation

In the `enableDepositPeriod` and `disableDepositPeriod` functions consider updating the capacities for the existing deposit periods with the `_updateTicks` function before a new period is added or removed to reflect the state of the capacities of each deposit period at that time.

Resolution

Olympus Team: The issue was resolved in PR#96.

M-09 | Capacity Calculation Incorrect

Category	Severity	Location	Status
Logical Error	Medium	ConvertibleDepositAuctioneer.sol: 397-398	Resolved

Description

```
uint256 capacityToAdd = (_auctionParameters.target * timePassed) /
1 days /
_depositPeriodsCount;
```

This formula assumes all deposit periods are active from the beginning. If a deposit period is enabled or disabled mid-auction:

• Enabling a new period causes under-selling (less OHM minted).

• Disabling a period causes overselling (more OHM minted).

Recommendation

Restrict enabling/disabling deposit periods to parameter updates (`setAuctionParams`) before tick recalculations. Only allow emergency toggling in exceptional cases.

Resolution

Olympus Team: The issue was resolved in PR#114.

M-10 | Target Update Without LastUpdate Adjustment

Category	Severity	Location	Status
Logical Error	Medium	ConvertibleDepositAuctioneer.sol: 723-724	Resolved

Description

When `setAuctionParams` is called to set a new target, `lastUpdate` is not updated. The new target applies retroactively from the last update, affecting both past and future periods, which may result in artificially discounted or inflated prices.

Recommendation

Update `lastUpdate` within `setAuctionParams` by setting `setLastUpdate_ = true`.

Resolution

Olympus Team: The issue was resolved in PR#112.

M-11 | Inconsistent Global VS Per-Period Auction Parameters

Category	Severity	Location	Status
Logical Error	Medium	ConvertibleDepositAuctioneer.sol: 291	Resolved

Description

The auction design maintains global variables for:
• tickSize
• target
• dayState

But maintains per-period state for:
• depositPeriodPreviousTicks[depositPeriod] (price, capacity, lastUpdate).
This creates asymmetry:
• When tick size is reduced globally (after hitting the daily limit), all deposit periods inherit the smaller tick size.
• However, each deposit period's previousTick (price, capacity, lastUpdate) is only updated when a bid occurs in that specific period.

If a new bid occurs in a different deposit period right after tick size reduction, the global tick size reduction applies instantly, but the local previousTick state for that period is stale. This leads to:
• getCurrentTick running for significantly more iterations (capacity ÷ smaller tick size, plus longer timePassed), pushing the price downward excessively.
• previewBid also using the reduced tick size, so while the final loop ramps the price upward, the first tick capacity is sold at a direct discount, and later ones start going up from the outdated starting price, resulting in total discount.

Example Scenario
• Two deposit periods (A and B).
• Period A hits its daily limit → tick size halved globally.
• Period B has not been updated since earlier in the day (large capacity, stale price, older lastUpdate).
• A user bids in Period B:
• Loop runs longer (capacity ÷ smaller tick size).
• Starting price is based on the outdated previousTick, which is lower.
• Price steps down further with each iteration, creating unintended discounts.
Net effect: the bidder in Period B acquires OHM cheaper than intended.

Recommendation

• Revisit the architecture:
• Either make all auction parameters (tick size, targets, day state) per-period, OR
• Keep them global but synchronize state updates across all deposit periods whenever tick size is reduced.

Resolution

Olympus Team: The issue was resolved in PR#113.

L-01 | Risk Of Underflow With Multiple Operators

Category	Severity	Location	Status
Warning	Low	BaseDepositFacility.sol	Resolved

Description

Because function handleBorrow does not decrement the appropriate _assetOperatorCommittedDeposits, if there are multiple authorized operators for a facility a malicious operator can prevent another operator from withdrawing entirely.

Consider the following scenario:
Operator A commits 50 tokens and Operator B commits 50 tokens, total tokens committed is 100
-> _assetOperatorCommittedDeposits[A] reads 50
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 100
Operator A borrows 50 tokens (handleBorrow)
-> _assetOperatorCommittedDeposits[A] still reads 50
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 50
Operator A can now handleCommitWithdraw
-> _assetOperatorCommittedDeposits[A] reads 0
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 0

Operator B cannot withdraw anything since _assetCommittedDeposits is 0 and underflow reverts on any operation. Note that for this attack vector to be performed there have to be multiple operators, however only the DepositRedemptionVault appears to be be a valid operator currently.

Recommendation

Update assetOperatorCommittedDeposits accordingly when borrowing.

Resolution

Olympus Team: The issue was resolved in PR#94.

L-02 | Insolvency When Large Yield

Category	Severity	Location	Status
Logical Error	Low	ConvertibleDepositFacility.sol: 358	Resolved

Description PoC

In DepositManager.claimYield, withdrawing a large yield relative to a small deposit burns all operator shares via vault.withdraw, setting depositedSharesInAssets to 0 while liabilities remain. This triggers DepositManager_Insolvent, blocking subsequent yield claims.

Consider the following scenario:

• Deposit: 100 assets, 100 shares, 100 receipts.

• Yield: +1e6 assets, vault = 1000100 assets, 100 shares, rate = 10001.0.

• Claim: Withdraws 999999 assets, burns all 100 shares.

• Post-claim: borrowed = 0, shares = 0, depositedSharesInAssets = 0, liabilities = 100 hence 100 > 0 + 0 which triggers DepositManager_Insolvent on claim.

Recommendation

Consider enforcing a minimum deposit amount as this scenario is more prevalent for small deposits.

Resolution

Olympus Team: The issue was resolved in PR#115.

L-03 | Protocol Risk When Interest Exceeds Buffer

Category	Severity	Location	Status
Warning	Low	DepositRedemptionVault.sol	Acknowledged

Description

In `DepositRedemptionVault`, the loan interest is not capped to ensure it remains below the buffer (`redemption.amount - loan.initialPrincipal`).

Governance can set `_assetAnnualInterestRates` and `_assetMaxBorrowPercentages` such that interest exceeds the buffer, incentivizing users to default rather than repay or extend loans.

Consider the following scenario:

• start redemption 100

• borrow 90 (principal:90 interest:45)

• if you repay -> pay 45, claim 100, net = 100 - 45 = 55

• if you default -> do not pay anything, lose everything, but walk away with 90

Recommendation

Clearly document and ensure the interest rate and max borrow is set appropriately so that the amount committed is greater.

Resolution

Olympus Team: Acknowledged.

L-04 | User Can Delay Reclaims

Category	Severity	Location	Status
Censoring	Low	DepositRedemptionVault.sol	Acknowledged

Description

When a user starts a redemption their assets are committed and the available deposits (`getAvailableDeposits`) is reduced by that amount.

The smaller the available deposits, the less users can `reclaimFor` due to `_validateAvailableDeposits` validation: `if (amount_ > availableDeposits) revert DepositFacility_InsufficientDeposits(amount_, availableDeposits);`

Because there is no penalty for starting a redemption and then cancelling (`cancelRedemption`) to receive their committed tokens back, malicious users can continuously commit their receipt tokens to delay user reclaims.

Consider the following example:

• User A has 100 receipt tokens, deposited 100 tokens

• User B has 100 receipt tokens, deposited 100 tokens

• User A commits 100 to start redemption

• `getAvailableDeposits = 100` (shares in assets assume 1:0.5 lossy vault) - 100 = 0

• User B cannot reclaim until User B cancels their redemption

Recommendation

Consider adding a fee for cancellation.

Resolution

Olympus Team: Acknowledged.

L-05 | Default VS Reclaim Gaming

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Depending on the configured parameters, it may be more favorable for a user to borrow, wait the loan's term, and self-default rather than reclaim. Consequently, the protocol treasury's earning may potentially be negatively impacted with the user incentives.

Recommendation

Document and take this into consideration when setting protocol parameters.

Resolution

Olympus Team: The issue was resolved in PR#93.

L-06 | Delayed Heartbeat Affects Auction Tracking

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Everyday the auction stores the convertible OHM in the `_dayState`, and this is recorded within the `_auctionResults`.

The expectation is that the `EmissionManager` will trigger the auction every 24 hours, and every 24 hours there will be an accurate representation of convertible amount to decide whether a bond market needs to be created.

The Heart is not required to beat every 8 hours, so waiting for 3 beats may take longer than 24 hours if there are outages, block congestion, etc.

In this case, more convertible amount may be attributed to a day than was actually sold in that day, preventing the creation of the bond market even if there was underselling.

Recommendation

Consider documenting this behavior.

Resolution

Olympus Team: The issue was resolved in PR#105.

L-07 | Price Decays Faster Than Expected

Category	Severity	Location	Status
Unexpected Behavior	Low	ConvertibleDepositAuctioneer.sol	Resolved

Description

In the `ConvertibleDepositAuctioneer` contract the size of each tick decays as the daily expected volume of purchases is surpassed by an incremental multiple. This allows the price of the auction to move up faster as more convertible deposits are purchased.

However it also allows the price to decay more rapidly due to the logic inside of `_getCurrentTick` which decays the price in correspondence with the number of ticks which would be crossed by the `capacityToAdd` which is added over time.

The outcome of this behavior is that the auction simply allows users to purchase more tokens at a lower average price within a day than may be expected when there is a large volume purchased, which pushes the amount to a multiple of the daily allotment, at the beginning of the day.

Recommendation

Be aware of this unexpected increase in the rate at which the auction price can drop after a large volume has been purchased. To negate this effect, use the initial tick size rather than the `_currentTickSize` in the decay `while` loop in the `_getCurrentTick` function.

Resolution

Olympus Team: The issue was resolved in PR#97.

L-08 | Deposit Period Enabled Before Policy

Category	Severity	Location	Status
Warning	Low	ConvertibleDepositAuctioneer.sol	Resolved

Description

Enabling a deposit period before the Auctioneer policy is initialized through `_enable` sets the previous tick's price and capacity to 0, as `_auctionParameters.minPrice` and `_auctionParameters.tickSize` are uninitialized. This invalid state persists until re-enabled or updated.

Recommendation

Add a check in `enableDepositPeriod` to revert if policy is not enabled or clearly document this behavior.

Resolution

Olympus Team: The issue was resolved in PR#103.

L-09 | Re-enabling Loses Tick Data

Category	Severity	Location	Status
Warning	Low	ConvertibleDepositAuctioneer.sol	Resolved

Description

Disabling and re-enabling a deposit period with functions `disableDepositPeriod` and `enableDepositPeriod` resets the previous tick to the auction's `minPrice` and `tickSize`, losing prior capacity and price data.

New capacity from time passed during disablement is not added, leading to lost OHM conversion potential.

Recommendation

Be aware and clearly document this behavior.

Resolution

Olympus Team: The issue was resolved in PR#103.

L-10 | Setting Tracking Period Wipes Existing Results

Category	Severity	Location	Status
Warning	Low	ConvertibleDepositAuctioneer.sol	Resolved

Description

Calling `setAuctionTrackingPeriod` resets `_auctionResults` to a new array of length `days_`, wiping prior results and setting `_auctionResultsNextIndex=0`.

This loses historical data, potentially misinforming `EmissionManager` on underselling and triggering unnecessary bond markets.

Note that the number of days can even be set to the same amount, and the historical data will be wiped.

Recommendation

Be aware and clearly document this behavior.

Resolution

Olympus Team: The issue was resolved in PR#103.

L-11 | Failed Bond Market Creation Derails Auction

Category	Severity	Location	Status
Warning	Low	EmissionManager.sol	Resolved

Description

The `EmissionManager` tightly couples auction parameter updates with bond market creation. If `bondAuctioneer.createMarket` fails (e.g. `allowNewMarkets = false`), the entire emission execution reverts.

The auction will continue to operate with stale parameters - minimum auction price, current tick size will not be adjusted to reflect latest emission rate and OHM price. Furthermore, the auction results will not be stored for the day.

Recommendation

Consider having governance safety mechanisms in-place to set auction parameters and data in the case of failure.

Resolution

Olympus Team: The issue was resolved in PR#105.

L-12 | Disabling Policy Loses Auction Results

Category	Severity	Location	Status
Warning	Low	ConvertibleDepositAuctioneer.sol	Acknowledged

Description

Disabling the Auctioneer before the 3rd Heartbeat (24 hours) skips `_storeAuctionResults` in `setAuctionParameters`, losing the daily convertible data. Re-enabling the policy starts the tracking from scratch, potentially triggering unnecessary bond markets due to perceived underselling.

Recommendation

Be aware and clearly document this behavior.

Resolution

Olympus Team: Acknowledged.

L-13 | Allowance Decrease Frontrunning

Category	Severity	Location	Status
Warning	Low	CloneERC20.sol	Acknowledged

Description

The `approve` function simply resets the allowance to the new value: `allowance[msg.sender][spender] = amount;` This allows a spender to utilize the prior approval before the new one is set.

For example, if the spender has a previous allowance of 100e18 and the owner wants to decrease this to 10e18, the spender can frontrun this call and consume the full 100e18 allowance, while getting an additional allowance of 10e18 granted.

Recommendation

Clearly document this risk with the `approve` function.

Resolution

Olympus Team: Acknowledged.

L-14 | Task Execution Succeeds

Category	Severity	Location	Status
Warning	Low	BasePeriodicTaskManager.sol	Resolved

Description

A low-level call is used to trigger a custom selector on a periodic task address. If the address does not have contract code, the call will return with `success = true` even if no task was truly executed.

Recommendation

Ensure only valid contract addresses are added for tasks.

Resolution

Olympus Team: The issue was resolved in PR#101.

L-15 | Max Yield Cannot Be Claimed

Category	Severity	Location	Status
Warning	Low	YieldDepositFacility.sol	Resolved

Description

The maxClaimYield function aims to "return the maximum yield that can be claimed for an asset and operator pair", but it does not represent the true amount that can be claimed from within the YieldDepositFacility.

A portion of the yield will be unclaimable through the YDF because the last snapshot rate for a position is adjusted by 1 wei, as well as precision loss when calculating a position's current value.

Therefore, the calculated yield across multiple positions and users will not sum up to the maxYield returns for the YDF operator. This may be unexpected for integrators and off-chain systems relying on maxClaimYield.

Recommendation

Clearly document that maxClaimYield is the theoretical max and may not be reached by all users claiming their current yield.

Resolution

Olympus Team: The issue was resolved in PR#109.

L-16 | Cross-Contract Reentrancy Risk

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Throughout the contracts, `safeTransferFrom` is not performed at the very top of the function. This opens up attack vectors for `ERC777` deposit tokens to trigger a cross-contract reentrancy since state updates are performed before the tokens are received.

Recommendation

Move `safeTransferFrom` to the top of functions and take into consideration which tokens are allowed within the protocol.

Resolution

Olympus Team: The issue was resolved in PR#102.

L-17 | Parameters Should Be Configurable

Category	Severity	Location	Status
Warning	Low	DepositRedemptionVault.sol	Resolved

Description

Parameters such as `_assetMaxBorrowPercentages` are dependent solely on the deposit token, and are the same across all facilities. However, some facilities may have different risk profiles, hence the borrow percentages, reclaim rates, etc. should be adjustable according to the facility as well.

Recommendation

Consider allowing for more granularity in parameter adjustments by also accounting for the facility.

Resolution

Olympus Team: The issue was resolved in PR#111.

L-18 | Lack Of Validation On Withdrawal

Category	Severity	Location	Status
Warning	Low	BaseAssetManager.sol: 114	Resolved

Description

There are multiple entry points in the `DepositManager` that access function `_withdrawAsset` such as `claimYield`, `borrowingWithdraw`, and `withdraw`.

Currently only `claimYield` validates that the `DepositManager` is solvent, but it would be advisable that `_withdrawAsset` itself would have the validation as a chokepoint pattern to ensure all withdrawals leave the `DepositManager` solvent and if there are any issues the Olympus team is swiftly aware.

Recommendation

Consider adding solvency validation to `_withdrawAsset`.

Resolution

Olympus Team: The issue was resolved in PR#107.

L-19 | Missing Validation For previewReclaim

Category	Severity	Location	Status
Validation	Low	BaseDepositFacility.sol: 300	Resolved

Description

During reclaims, the amount being withdrawn is compared to the available deposits, accounting for commitments, to verify if there are enough assets to reclaim.

However, deposit manager will only allow to withdraw up to the asset liabilities of the given facility, which only increase during deposits.

Given the fact that YDF users can reclaim using CDF, withdrawing funds may underflow as the amount was not validated against the facility liabilities:

```
_assetLiabilities[_getAssetLiabilitiesKey(params_.asset, msg.sender)] = params_.amount;
```

Recommendation

During `previewReclaim`, verify that the `amount` param is lower or equal to the facility liabilities in the `DepositManager`, not the available deposits (which include yield).

Resolution

Olympus Team: The issue was resolved in PR#108.

L-20 | Initial Auction Parameters Not Validated

Category	Severity	Location	Status
Validation	Low	ConvertibleDepositAuctioneer.sol: 605	Resolved

Description

The EmissionManager is responsible for periodically tuning the auction parameters, which are calculated based on the emissions and current OHM price. The tickSize_ is calculated as a % of the target_.

However, during the ConvertibleDepositAuctioneer policy enable flow, there is no validation on tickSize_ < target_. If the tick size is greater than the target , it will open unexpected behaviors, as the auctioneer will decrease the tick size faster due to higher values of the multiplier.

Recommendation

Consider not only validating non zero values for the auction params, but ensuring tickSize_ < target_, in line with the EmissionManager. This can be done in the _enable as the issue only appears during policy enabling.

Resolution

Olympus Team: The issue was resolved in PR#106.

L-21 | Risk-Free Bids

Category	Severity	Location	Status
Warning	Low	Global	Acknowledged

Description

Based on the current Auction status, users can bid for convertible notes and ideally purchase a position with the lowest conversion price to make a profit from longing OHM.

However, even if price does decrease, users can still start a redemption and receive their entire deposit back 1:1.

Consequently, users have no direct asset-risk for utilizing convertible notes. Olympus should confirm this is intended mechanics for convertible notes, especially since most auctions will begin at an OHM price below market, effectively creating a risk-free money printer for some.

Recommendation

Consider if this is intended behavior. If so, consider extra fees upon redemptions or bids. Furthermore, consider implementing the possibility for a time delay before conversion.

Resolution

Olympus Team: Acknowledged.

L-22 | Asymmetric Yield Across Multiple Claims

Category	Severity	Location	Status
Gaming	Low	YieldDepositFacility.sol: 333	Resolved

Description

Function claimYield updates the position's last snapshot rate after each yield claim, setting the lastSnapshotRate to a higher value for subsequent calculations as yield is earned.

This increases the denominator in the lastShares computation for remaining growth periods, resulting in fewer effective shares (lastShares) and lower total yield when claiming multiple times compared to a single claim over the same period and end rate.

Ultimately, users who claim yield multiple times during a position’s period will receive less total yield than those who claim once.

Recommendation

Clearly document this behavior to users.

Resolution

Olympus Team: The issue was resolved in PR#84.

L-23 | ERC-4626 Deposit VS PreviewRedeem Invariant

Category	Severity	Location	Status
Superfluous Code	Low	BaseAssetManager.sol: 90-91	Acknowledged

Description

In yield deposit facilities, Olympus first deposits tokens into the vault and then calls `previewRedeem` to determine the owed assets, minting equivalent receipt tokens.

However, there is no guarantee that, `previewRedeem(deposit(amount x)) ≈ x` would always holds true for all kinds of vaults.

This may deviate if the vault charges withdrawal fees or considers available liquidity when calculating redemptions.

Recommendation

Before onboarding any vault, review its behavior to ensure this invariant holds. Specifically, confirm that `previewRedeem(deposit(x)) ≈ x` for the chosen vault, otherwise deposits may miscalculate user entitlements.

Resolution

Olympus Team: Acknowledged.

L-24 | Vault Validation Warnings

Category	Severity	Location	Status
Validation	Low	DepositManager.sol	Acknowledged

Description

Each asset has a specified deposit cap (`assetConfiguration.depositCap`) which should not be exceeded when depositing assets through the `DepositManager`. Each asset also has an arbitrarily configured ERC4626 vault which was set during the `addAsset` function call by a manager or admin.

Because the vault can require a minimum deposit amount which is not currently enforced directly in the `DepositManager`, there is potential for the minimum deposit to be greater than the amount of capacity leftover before exceeding the `assetConfiguration.depositCap` and users will be unable to deposit even when there is available space.

Another potential asymmetry is if the vault has its own deposit cap, and users may attempt deposits through the `DepositManager` that will simply fail when the deposit into the vault is attempted and the vault's cap is exceeded, although there is still available space from the `DepositManager`'s perspective.

Recommendation

Try to maintain alignment between the vault's configuration and `DepositManager`'s configuration to avoid unexpected behavior and/or document this risk.

Resolution

Olympus Team: Acknowledged.

L-25 | Asymmetric Remaining Deposit

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

Function `YieldDepositFacility:createPosition` uses `remainingDeposit: params_.amount` but this may slightly differ from the `actualAmount` returned by the deposit due to `ERC4626` rounding.

This `remainingDeposit` value is asymmetric with `ConvertibleDepositFacility:createPosition` which sets it as `remainingDeposit: actualAmount`.

Consequently, the `currentValue` during yield calculations within the `YieldDepositFacility` may be just slightly larger than intended.

Recommendation

Consider using `remainingDeposit: actualAmount`.

Resolution

Olympus Team: The issue was resolved in PR#87.

L-26 | Lack Of Slippage Protection On Bids

Category	Severity	Location	Status
MEV	Low	ConvertibleDepositAuctioneer.sol	Resolved

Description

As users call function `bid()` to purchase convertible deposit tokens, the tick prices will continue to increase as demand grows. However, there is no slippage protection mechanism in the function, such that a user may receive a much lower `output.ohmOut` than expected.

Recommendation

Consider adding slippage-protection on the output amount in function `bid()`.

Resolution

Olympus Team: The issue was resolved in PR#88.

L-27 | Split Griefing Increases Gas Cost

Category	Severity	Location	Status
Gas Griefing	Low	OlympusDepositPositionManager.sol	Resolved

Description

The `split` function in `OlympusDepositPositionManager` allows a position owner to split a position into smaller positions (e.g., 1 wei) and assign them to an arbitrary receiver, appending new position IDs to the receiver's `_userPositions` array.

A malicious user can repeatedly split tiny amounts to a recipient, bloating their `_userPositions` array. This increases gas costs for the receiver in `transferFrom`, which loops over `_userPositions` to remove the transferred ID.

While testing it was found an attacker would expend more gas than the recipient would lose with the bloated list of positions, making DoS scenarios unlikely.

Recommendation

Consider enforcing a minimum amount for the `split` function and/or clearly document this risk.

Resolution

Olympus Team: The issue was resolved in PR#92.

L-28 | Potential Superior Loans

Category	Severity	Location	Status
Configuration	Low	DepositRedemptionVault.sol	Resolved

Description

In the DepositRedemptionVault, depending on the interest rate and _claimDefaultRewardPercentage a borrower may be able to borrow and pay a lower total amount than compared to the interest rate by defaulting themselves.

For example:

• Consider an interest rate of 12% annually

• A _assetMaxBorrowPercentages of 80%

• A _claimDefaultRewardPercentage of 50%

• The interest for a 12 month period deposit of 100 tokens is 12 tokens

• The principal for the 12 month period deposit is 80 tokens, leaving 20 as a buffer

• If the user pays off their loan before the end of the 12 month period, they will pay back the 80 token principal + the 12 token interest

• If the user instead waits until the end of their loan period and defaults themselves, they will lose the 20 token buffer, but gain half of that as a default reward

In the case where the user defaults themselves they keep the 80 token borrowed amount and only lose the 10 token half of the buffer, rather than paying the 12 tokens in interest.

Recommendation

Be aware of this superior loan strategy and consider this when configuring the _assetMaxBorrowPercentages, _claimDefaultRewardPercentage, and interest rate.

Resolution

Olympus Team: The issue was resolved in PR#93.

L-29 | Handle Loan Functions

Category	Severity	Location	Status
Validation	Low	BaseDepositFacility.sol	Resolved

Description

The handleBorrow and handleLoanRepay functions neglect to update the _assetOperatorCommittedDeposits mapping values, while they do update the _assetCommittedDeposits for the asset, allowing the total committed deposits for an asset to disagree with the sum of all operator committed deposits.

Furthermore, this allows one operator to “steal” committed deposits from another. Consider, ConvertibleDepositFacility with Operators A and B

Operator A commits 50 tokens and Operator B commits 50 tokens, total tokens committed is 100
-> _assetOperatorCommittedDeposits[A] reads 50
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 100
Operator A borrows 50 tokens
-> _assetOperatorCommittedDeposits[A] still reads 50
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 50
Operator A can now handleCommitWithdraw
-> _assetOperatorCommittedDeposits[A] reads 0
-> _assetOperatorCommittedDeposits[B] reads 50
-> _assetCommittedDeposits reads 0

Operator B cannot withdraw anything since _assetCommittedDeposits is 0 and underflow reverts on any operation. Currently there is no issue related to this as there is only one planned operator, the DepositRedemptonVault, and the logic within the DepositRedemptonVault does not allow for this to be exploited.

Recommendation

Consider updating the _assetOperatorCommittedDeposits to correspond with the _assetCommittedDeposits update in the handleBorrow and handleLoanRepay functions

Resolution

Olympus Team: The issue was resolved in PR#94.

L-30 | Borrowing More Is Incentivized For Defaults

Category	Severity	Location	Status
Warning	Low	DepositRedemptionVault.sol: 688	Acknowledged

Description

Upon loan default, the `retainedCollateral` which serves as the buffer between user collateral value and borrow amount is withheld from the user: `uint256 retainedCollateral = redemption.amount - loan.initialPrincipal`.

Borrowers are incentivized to borrow more if defaulting since the `retainedCollateral` is smaller when the `initialPrincipal` is larger.

This may encourage users who do not have intention of paying the interest to maximally borrow funds and reduce available deposits, which may affect features such as reclaims which require available deposits.

This creates a counterintuitive outcome where borrowers taking larger loans are penalized less, incentivizing higher borrowing. Assets with lower borrowing capacity could result in disproportionate losses to users.

Recommendation

Consider aligning with the standard liquidation practices:

• Keeper receives a cut proportional to borrowed amount.

• Protocol receives a liquidation penalty proportional to borrowed amount.

• Remainder is returned to the position holder.

Resolution

Olympus Team: Acknowledged.

I-01 | YDF conversionPrice Should Use Constant

Category	Severity	Location	Status
Best Practices	Info	YieldDepositFacility.sol: 131	Resolved

Description

In `YieldDepositFacility:createPosition`, the `conversionPrice` for a yield-only position is hardcoded to `type(uint256).max`.

The DEPOS module already defines a `constant NON_CONVERSION_PRICE = type(uint256).max` for non-convertible positions.

Hardcoding risks misalignment if the module is updated with a different `NON_CONVERSION_PRICE` in the future, even if this is an unlikely scenario.

Recommendation

Consider using the existing `NON_CONVERSION_PRICE` constant in `YieldDepositFacility:createPosition`.

Resolution

Olympus Team: The issue was resolved in PR#109.

I-02 | Empty Position Id List Validation Missing

Category	Severity	Location	Status
Best Practices	Info	ConvertibleDepositFacility.sol: 289	Resolved

Description

The `convert` function in `ConvertibleDepositFacility` does not explicitly check if `positionIds_.length > 0`, allowing calls with empty arrays.

While this results in `receiptTokenIn` and `convertedTokenOut` being 0, the transaction reverts safely during `MINTR.mintOhm` (which disallows zero mints). However, this lacks an early revert, potentially wasting gas and reducing user clarity.

Recommendation

Consider reverting `if (positionIds_.length = 0)` to be symmetrical with `YieldDepositFacility` validations.

Resolution

Olympus Team: The issue was resolved in PR#109.

I-03 | depositAmount Calculation Rounds

Category	Severity	Location	Status
Rounding	Info	ConvertibleDepositAuctioneer.sol	Resolved

Description

In the `_previewBid` function the `depositAmount = convertibleAmount.mulDiv(output.tickPrice, _ohmScale)` calculation which occurs when the `depositAmount` must be derived from the total available capacity at the current tick uses round down division.

Therefore the resulting `depositAmount` is rounded down from what the user has to put into the conversion from their total deposit. This however rounds against the favor of the protocol and in favor of the user.

Recommendation

Out of an abundance of caution, to follow best practices of rounding this calculation should round against the user and estimate a larger `depositAmount` that should be deducted from the user's total deposits using `mulDivUp`.

Notice that this may result in cases where the `remainingDeposit` would be subtracted such that it underflows and causes a panic revert due to additional wei being removed. In this case the `remainingDeposit` value should be minimized to zero.

Resolution

Olympus Team: The issue was resolved in PR#109.

I-04 | Unwrapped Receipt Necessary For Redemption

Category	Severity	Location	Status
Documentation	Info	DepositRedemptionVault.sol	Resolved

Description

Only unwrapped receipt tokens are supported to start redemptions due to `_pullReceiptToken` requiring the `ERC6909` function.

Furthermore, function `reclaimFor` requires unwrapped tokens due to `isWrapped: false`. This is safe from a security perspective, but should be clearly documented that users must unwrap before beginning redemptions/reclaiming.

Recommendation

Clearly document this behavior.

Resolution

Olympus Team: The issue was resolved in PR#110.

I-05 | Actual Amount Ignored During Bids

Category	Severity	Location	Status
Informational	Info	ConvertibleDepositAuctioneer.sol: 235	Resolved

Description

The `YieldDepositFacility.createPosition` returns the `actualAmount` which represents the amount of receipt tokens minted.

Even though the `ConvertibleDepositFacility.createPosition` returns this same parameter, this function is only callable by the `ROLE_AUCTIONEER` which is granted to the `ConvertibleDepositAuctioneer`.

The `ConvertibleDepositAuctioneer.bid` does not return the `actualAmount`. Therefore, integrations will need to rely on the receipt token balance delta to calculate this amount.

Recommendation

Return the `actualAmount` parameter during bids

Resolution

Olympus Team: The issue was resolved in PR#109.

I-06 | Redundant Boolean Check

Category	Severity	Location	Status
Best Practices	Info	DepositRedemptionVault.sol: 678	Resolved

Description

Within function `claimDefaultedLoan`, the expression `loan.isDefaulted = true` is not necessary and can be shorted to `loan.isDefaulted`.

Recommendation

Consider changing the expression to `loan.isDefaulted`.

Resolution

Olympus Team: The issue was resolved in PR#110.

I-07 | Using Magic Numbers

Category	Severity	Location	Status
Best Practices	Info	Global	Resolved

Description

The codebase uses magic numbers to represent certain values. For example:

• `100e2` instead of `ONE_HUNDRED_PERCENT`, used in `_previewBorrowAgainstRedemption`, `setMaxBorrowPercentage`, `setAnnualInterestRate`, `setClaimDefaultRewardPercentage`

• `12` instead of creating `TWELVE_MONTHS` constant

• `30 days` instead of creating `ONE_MONTH` constant

Recommendation

Consider using constants instead of magic numbers, to avoid mistakes.

Resolution

Olympus Team: The issue was resolved in PR#91.

I-08 | Superfluous Permission Request

Category	Severity	Location	Status
Superfluous Code	Info	ConvertibleDepositFacility.sol: 79	Resolved

Description

The `ConvertibleDepositFacility` policy has a requests permission for the `MINTR.decreaseMintApproval`, but the contract never calls this function.

Recommendation

Remove superfluous permission.

Resolution

Olympus Team: The issue was resolved in PR#109.

I-09 | Borrows Pay Full Interest Regardless Of Payback

Category	Severity	Location	Status
Documentation	Info	DepositRedemptionVault.sol	Resolved

Description

If a user borrows deposit tokens for a short period of time (e.g. a single block) and proceeds fully repay their loan, the user still owes the entire interest as if they had an active loan for the entire deposit period.

This may be unexpected for some users, especially who are accustomed to time based interest mechanisms.

Recommendation

Clearly document this behavior to users.

Resolution

Olympus Team: The issue was resolved in PR#110.

I-10 | Newly Minted OHM Not Counted In Supply

Category	Severity	Location	Status
Warning	Info	EmissionManager.sol	Acknowledged

Description

Function `getSupply()` return supply as `(gohm.totalSupply() * gohm.index()) / 10 * _gohmDecimals;`, but this does not consider any newly minted OHM. Consequently, the emission may be lower than intended.

Recommendation

Clarify if this is intended behavior.

Resolution

Olympus Team: Acknowledged.

I-11 | Convertible Deposit Facility (CDF)

Category	Severity	Location	Status
Logical Error	Info	BaseDepositFacility.sol: 314-315	Acknowledged

Description

Within the Convertible Deposit Facility, both redemptions and conversions can occur against the same position.

The user just needs to acquire more receipt tokens through a `deposit` or `createPosition` to have both be in parallel.

While no immediate issue was identified, this is a feature/inconsistency protocol team should be aware of.

Recommendation

Be aware of this duality during future modifications.

Resolution

Olympus Team: Acknowledged.

I-12 | Unnecessary Day State Storage Read

Category	Severity	Location	Status
Gas Optimization	Info	ConvertibleDepositAuctioneer.sol: 298	Resolved

Description

The `_previewBid` will loop until `remainingDeposit = 0`, increasing the tick price and size.

If there is not enough capacity in the current tick, the size is calculated based on `_getNewTickSize` which uses `_dayState.convertible` to account for daily ohm bids.

However, this daily value is constant throughout the while loop, creating unnecessary storage reads.

Recommendation

Consider caching the `_dayState.convertible` before the while loop, and using the memory variable instead.

Resolution

Olympus Team: The issue was resolved in PR#109.

I-13 | Redundant Setting Of Asset And Period

Category	Severity	Location	Status
Logical Error	Info	ConvertibleDepositFacility.sol: 316-317	Acknowledged

Description

Asset and deposit period values are expected to remain consistent across all entries of a position. Currently, these values are set repeatedly within the loop for each iteration, which is redundant.

Recommendation

Optimize by setting `asset` and `periodMonths` once for the first position and reuse them, rather than resetting on each loop iteration.

Resolution

Olympus Team: Acknowledged.

I-14 | Unnecessary While Loop

Category	Severity	Location	Status
Gas Optimization	Info	ConvertibleDepositAuctioneer.sol: 397	Acknowledged

Description

The `_getCurrentTick` function uses a `while (newCapacity > _currentTickSize)` loop to iteratively reduce capacity and decay the tick price until capacity fits within the tick size or hits `minPrice`.

This is unnecessary, as the number of decay steps and final price can be computed directly without looping.

Recommendation

Consider replacing the loop with a direct calculation.

Resolution

Olympus Team: Acknowledged.

I-15 | Lack Of SafeTransfer

Category	Severity	Location	Status
Best Practices	Info	YieldDepositFacility.sol	Resolved

Description

In the `YieldDepositFacility` contract, the `transfer` function is used when claiming yield, but best practices is to use `safeTransfer` to handle the return values of the various assets that can be configured.

Recommendation

Consider using `safeTransfer` instead from `SafeERC20`.

Resolution

Olympus Team: The issue was resolved in PR#95.

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

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian 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