

RAAC.IO · SECURITY REVIEW

raac.io

RAAC Core Contracts.

A security review of RAAC Core Contracts.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
raac.io	RAAC Core Contracts	Felipe Gomez, Nicholas Chew, Vlad Zotov, Michael Lett, Ferran Celades	October 2 – October 28, 2026	October 28, 2026

§ 00

Contents.

54 pages: overview, confidence ranking, scope, 30 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 08
	Main Review · 29 findings	P. 10
	Remediation Review · 1 finding	P. 52
§ 05	Disclaimer	P. 53
§ 06	About Guardian	P. 54

§ 01

Overview.

Guardian reviewed RAAC Core Contracts across 2 review rounds. The final report includes 1 Medium, 14 Low, 15 Informational findings.

AUDIT FIRM

Guardian

CLIENT

raac.io

PROJECT

RAAC Core Contracts

PREPARED BY

Felipe Gomez, Nicholas Chew, Vlad Zotov,
Michael Lett, Ferran Celades

AUDIT WINDOW

October 2 – October 28, 2026

FINAL REPORT

October 28, 2026

REPOSITORY

<https://github.com/gundam-foundation/contracts>

METHODOLOGY

Smart contract · Manual · Main Review ·
Remediation Review

MAIN REVIEW COMMIT

7efdd353829d

REMEDIATION REVIEW COMMIT

c864e49254b5

1.1 Confidence Ranking

5 / 5

**Very High
Confidence**

Guardian assigns a Confidence Ranking of 5 out of 5, combining finding severity, remediation outcome, and the reviewer's assessment of latent risk. The full scale is on the next page.



1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
● Critical	0	0	0	0	0
● High	0	0	0	0	0
● Medium	1	0	0	0	1
● Low	14	0	3	1	10
● Informational	15	0	1	1	13
All severities	30	0	4	2	24

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence ASSIGNED	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.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed RAAC Core Contracts in <https://github.com/gundam-foundation/contracts>. The scope includes 2,713 source lines across 20 files.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH
Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM
Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW
Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH
Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM
Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW
Unlikely to ever occur in production.

Files in Scope 1/2.

FILE	NSLOC	TOTAL
MAIN REVIEW		
contracts/core/oracles/CrvUSDToUSDOracle.sol	119	266
contracts/core/oracles/PmUSDToUSDOracle.sol	5	13
contracts/core/pools/LendingPool/ERC20AssetAdapter.sol	109	181
contracts/core/pools/LendingPool/ERC721AssetAdapter.sol	138	215
contracts/core/pools/LendingPool/LendingPool.sol	594	1,145
contracts/core/pools/LendingPool/LendingPoolStorage.sol	45	87
contracts/core/pools/LendingPool/LiquidationProxy.sol	77	152
contracts/core/pools/LendingPool/VaultProxy.sol	198	355
contracts/core/pools/StabilityPool/LiquidationStrategyProxy.sol	93	175
contracts/core/pools/StabilityPool/LiquidationSwap.sol	75	140
contracts/core/pools/StabilityPool/StabilityPool.sol	249	483
contracts/core/pools/StabilityPool/StabilityPoolStorage.sol	40	58
contracts/core/tokens/DebtToken.sol	103	232
contracts/core/tokens/DEToken.sol	148	315
contracts/core/tokens/RToken.sol	166	383
contracts/libraries/math/PercentageMath.sol	29	62
contracts/libraries/math/WadRayMath.sol	97	218
contracts/libraries/pools/LendingPoolFeeLibrary.sol	45	81
contracts/libraries/pools/ReserveLibrary.sol	246	513
contracts/libraries/utils/StringUtils.sol	14	29
REMEDIATION REVIEW		
contracts/core/oracles/CrvUSDToUSDOracle.sol	0	263
contracts/core/oracles/PmUSDToUSDOracle.sol	0	13
contracts/core/pools/LendingPool/ERC20AssetAdapter.sol	0	181
contracts/core/pools/LendingPool/ERC721AssetAdapter.sol	8	226
contracts/core/pools/LendingPool/LendingPool.sol	15	1,141
contracts/core/pools/LendingPool/LendingPoolStorage.sol	0	87
contracts/core/pools/LendingPool/LiquidationProxy.sol	3	155
contracts/core/pools/LendingPool/VaultProxy.sol	8	345
contracts/core/pools/StabilityPool/LiquidationStrategyProxy.sol	1	175
contracts/core/pools/StabilityPool/LiquidationSwap.sol	0	140
contracts/core/pools/StabilityPool/StabilityPool.sol	0	479
contracts/core/pools/StabilityPool/StabilityPoolStorage.sol	0	57
contracts/core/tokens/DebtToken.sol	0	232
contracts/core/tokens/DEToken.sol	0	314

Files in Scope 2/2.

FILE	NSLOC	TOTAL
contracts/core/tokens/RToken.sol	0	383
contracts/libraries/math/PercentageMath.sol	0	62
contracts/libraries/math/WadRayMath.sol	80	242
contracts/libraries/pools/LendingPoolFeeLibrary.sol	6	96
contracts/libraries/pools/ReserveLibrary.sol	2	510
contracts/libraries/utils/StringUtils.sol	0	29

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
M-01	LendingPool.pause() blocks StabilityPool.liquidateBorrower via updateState	DoS	MEDIUM	RESOLVED	p. 10
L-01	repayOnBehalf can grief a rescue	Frontrunning	LOW	ACKNOWLEDGED	p. 12
L-02	Permissionless checkpoints compound the linear supply index and drain reserve spread	economic-accounting	LOW	RESOLVED	p. 13
L-03	Stale NFT appraisals are accepted on the live iRAAC collateral/NAV path	oracle-staleness/economic-accounting	LOW	RESOLVED	p. 15
L-04	StabilityPool liquidation requires idle cash, blocking bad-debt resolution at high utilization	denial-of-service	LOW	RESOLVED	p. 16
L-05	Duplicate same-token adapters alias positions and let an attacker erase debt while draining the StabilityPool	access-control	LOW	RESOLVED	p. 18
L-06	Supply cap bypassed and avoids deposit fee	Configuration	LOW	RESOLVED	p. 20
L-07	Unguarded tier-0 oracle calls bypass fallbacks and freeze valuation-dependent operations	denial-of-service	LOW	ACKNOWLEDGED	p. 22
L-08	LiquidationFinalized reports zero collateral for ERC20 liquidations	logic/event-accounting	LOW	RESOLVED	p. 23
L-09	Unbounded ERC721 position enumeration can exhaust gas for debt reads and borrowing	denial-of-service	LOW	RESOLVED	p. 24
L-10	Blacklisted spender bypasses RToken compliance via transferFrom allowances	compliance/access-control	LOW	ACKNOWLEDGED	p. 25
L-11	Residual RToken cash has no defined claim path	funds-locking	LOW	RESOLVED	p. 27
L-12	Harvest can make vault rebalance revert	DoS	LOW	RESOLVED	p. 28
L-13	Liquidation rebalances the vault before repayment	accounting	LOW	PARTIALLY RESOLVED	p. 29
L-14	Insecure production deployment configuration	Configuration	LOW	RESOLVED	p. 31
I-01	Insurance fee omits accrued interest	Logical Error	INFORMATIONAL	RESOLVED	p. 33
I-02	Documented incident burn of collateralized NFT leaves unsecured, forcibly-unrecoverable debt	collateral-accounting	INFORMATIONAL	RESOLVED	p. 35

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-03	Protocol fees are paid from depositor principal before borrower interest is realised	economic-accounting	INFORMATIONAL	RESOLVED	p. 36
I-04	Large liquidations miss the slippage floor	DoS	INFORMATIONAL	RESOLVED	p. 37
I-05	Public liquidation debt helper reverts for all direct callers	availability	INFORMATIONAL	RESOLVED	p. 38
I-06	Non-receiver contract accounts cannot withdraw ERC721 collateral	availability	INFORMATIONAL	ACKNOWLEDGED	p. 39
I-07	InsuredAsset emits only the pre-existing uninsured balance	data-integrity	INFORMATIONAL	RESOLVED	p. 40
I-08	FeeCollector pause blocks liquidations	DoS	INFORMATIONAL	RESOLVED	p. 41
I-09	NFT flagged while price update is pending	Oracle	INFORMATIONAL	RESOLVED	p. 42
I-10	Incorrect NatSpec and documentation	Documentation	INFORMATIONAL	PARTIALLY RESOLVED	p. 43
I-11	Unused code	Superfluous Code	INFORMATIONAL	RESOLVED	p. 46
I-12	YieldHarvested reports zero priceIncrease on every harvest	data-integrity	INFORMATIONAL	RESOLVED	p. 48
I-13	Admin setters accept out-of-range values	Validation	INFORMATIONAL	RESOLVED	p. 49
I-14	LiquidityRebalanced reports pre-rebalance buffer and tracked principal	Events	INFORMATIONAL	RESOLVED	p. 51

REMEDIATION REVIEW

I-01	Liquidation still draws on the vault before repayment arrives	Gas Optimization	INFORMATIONAL	RESOLVED	p. 52
------	---	------------------	---------------	----------	-------

MEDIUM **M-01**

RESOLVED

LendingPool.pause() blocks StabilityPool.liquidateBorrower via updateState

CATEGORY	DoS
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/StabilityPool/LiquidationStrategyProxy.sol#L53
REVIEW	Main Review

DESCRIPTION

The only production settlement path is `StabilityPool.liquidateBorrower` → `LiquidationStrategyProxy.liquidateBorrower`. That function always calls `LendingPool.updateState()` before it finalizes, withdraws RToken, or redeposits leftover reserve asset.

`updateState()` is `whenNotPaused`. A LendingPool pause therefore reverts the strategy before any settlement. `finalizeLiquidation()` has no pause guard, and `deposit()` / `withdraw()` explicitly skip pause when `msg.sender` is the Stability Pool so liquidations can finish. Those exemptions never run.

There is no other live route. External `finalizeLiquidation()` calls revert `onlyStabilityPool`. Direct strategy calls revert `onlyProxy`. The Stability Pool only reaches finalize through this function.

The public checkpoint is also unnecessary for settlement. `getPositionScaledDebt` already uses a view index, and `LiquidationProxy.finalizeLiquidation` calls `updateReserveState` before it burns debt.

Alice is flagged and the grace period has ended. An emergency pause freezes user deposit, borrow, repay, and `initiateLiquidation`. The manager then calls `liquidateBorrower`. `updateState()` reverts `EnforcedPause`. Alice's collateral stays on the adapter, her debt stays open, and the backstop cannot close the hole until the pool is unpaused, which also turns user flows back on.

```
// Update lending pool state before liquidation
lendingPool.updateState();
```

Also at: <https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/LendingPool.sol#L650>

PoC: `test/unit/core/pools/StabilityPool/StabilityPool.test.js` (H-01: LendingPool pause reverts `liquidateBorrower` on `updateState`).

M-01 LendingPool.pause() blocks StabilityPool.liquidateBorrower via
updateState

CONTINUED

— **RECOMMENDATION**

Do not gate settlement on the public paused `updateState()`. Let `finalizeLiquidation` refresh reserve state itself, or expose an internal accrue that the Stability Pool can call while the pool is paused, so the existing deposit and withdraw exemptions can complete a liquidation.

— **RESOLUTION**

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/542/changes>

LOW **L-01**

ACKNOWLEDGED

repayOnBehalf can grief a rescue

CATEGORY	Frontrunning
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/LendingPool.sol#L601-L602
REVIEW	Main Review

— DESCRIPTION

Anyone can call `repayOnBehalf` for at least 1% of a position's debt. `_repay` does not cap an ordinary amount to remaining debt. Only `type(uint256).max` is replaced with the live debt. Any other amount goes to `DebtToken.burn`, which reverts `InsufficientBalance` when it is larger than remaining debt.

A borrower who quoted an exact repay can therefore be front run. After the 1% cut, that quoted amount is too large and the rescue reverts. If this happens at the end of insured grace, the next repay is past the deadline and reverts `GracePeriodExpired`. The Stability Pool can then finalize and take the collateral.

```
uint256 actualRepayAmount = (amount == type(uint256).max) ? positionScaledDebt :
    amount;
```

— RECOMMENDATION

Be aware of and document this behavior. A 1% `repayOnBehalf` can void a quoted exact repay at the end of grace, after which the borrower cannot retry and the position can be liquidated.

— RESOLUTION

Status: Acknowledged.

LOW **L-02**

RESOLVED

Permissionless checkpoints compound the linear supply index and drain reserve spread

CATEGORY	economic-accounting
LOCATION	packages/contracts/contracts/libraries/pools/ReserveLibrary.sol:updateReserveInterests:L126-L162
REVIEW	Main Review

— DESCRIPTION

updateReserveInterests applies a linear factor to the already-indexed liquidityIndex and then resets lastUpdateTimestamp. Because updateState is permissionless, callers choose the number of checkpoints in an elapsed interval; repeated daily checkpoints therefore compound a rate documented and implemented as linear. The debt index remains path-independent compound, so this is not a symmetric borrower/supplier calculation.

Impact: The PoC passes against the real LendingPool stack: at 100% utilization, one annual checkpoint pays 75,267 crvUSD interest on 100,000 principal, while 365 permissionless daily checkpoints pay 112,101.56 crvUSD. The extra 36,834.56 is realized on withdrawal and leaves only about 190.56 crvUSD of cash after the pending fee versus about 42,435.51 in the control. Thus a supplier/keeper can make liabilities and retained spread depend on call frequency and can consume the buffer intended for protocol fees and bad debt.

— RECOMMENDATION

Aave V3 uses the same split. Its supply index is a linear factor multiplied into the stored index, and its borrow index is compound. However, Aave's rates do not sit on a flat half-of-prime and then step up to an 83% rate ceiling.

Consider documenting this behavior instead of refactoring the interest rate calculations. Monitor utilization rates constantly, and make sure the manage the protocol caps to avoid a high utilization rate over a long period of time.

Additionally, make sure the protocol docs describes the real behavior of the supply rate, to avoid confusion, as the `ReserveLibrary.md` note states `Interest calculations use different methods for deposits (linear) and borrows (compound)`.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/permissionless-checkpoints-compound-the-linear-supply-index--4c815c6c.js>

L - Permissionless checkpoints compound the linear supply index and drain **CONTINUED**
02 reserve spread

— **RESOLUTION**

Status: Resolved.

LOW **L-03**

RESOLVED

Stale NFT appraisals are accepted on the live iRAAC collateral/NAV path

CATEGORY	oracle-staleness/economic-accounting
LOCATION	packages/contracts/contracts/core/vaults/RAACNFTVaultAdapterOracle.sol:getLatestPrice:L31-L33
REVIEW	Main Review

— DESCRIPTION

RAACNFT.mint enforces a 3600-second freshness rule, but the production RWAVault path receives the same (price,timestamp) from RAACNFTVaultAdapterOracle and discards the timestamp in ERC721VaultAdapter/RAACNFTVaultAdapterV2. The vault therefore accepts an NFT with an expired appraisal, adds its obsolete value to nftsValue, and mints iRAAC against it; RWAIndexTokenOracle then values those shares through the LendingPool's live ERC20 adapter with no age check. The PoC confirms the stale timestamp on both house prices and the live vault oracle, shows the mint path reverting while vault deposit succeeds, and demonstrates subsequent borrowing and NAV dilution of an existing holder when the quote is eventually marked down.

Impact: An oracle outage or delayed appraisal allows a stale-high NFT quote to over-issue iRAAC and make the over-issued shares eligible for borrowing. The cached NAV remains inflated until an external price push; when corrected, pre-existing iRAAC holders lose redeemable value and the lending position becomes undercollateralized, with repeatability across NFTs. The direct ERC721 adapter has the same stale-price omission, but the accepted impact is primarily the deployed iRAAC route.

— RECOMMENDATION

A first approach will suggest to keep the price updated with the same freshness as the NFT minting. However, we recommend to monitor the house prices closely, and trigger updates when there is a significant deviation with the on chain price, avoiding step wise jumps in LTV.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/stale-nft-appraisals-are-accepted-on-the-live-iraac-collater-a9095112.js>

— RESOLUTION

Status: Resolved.

LOW **L-04**

RESOLVED

StabilityPool liquidation requires idle cash, blocking bad-debt resolution at high utilization

CATEGORY	denial-of-service
LOCATION	packages/contracts/contracts/core/pools/StabilityPool/LiquidationStrategyProxy.sol:liquidateBorrower:L69-L83
REVIEW	Main Review

— DESCRIPTION

LiquidationStrategyProxy first computes the shortfall in RTokens, then calls LendingPool.withdraw() to redeem those RTokens for crvUSD before transferring the debt amount back to RToken.

LendingPool.withdraw() unconditionally requires equivalent idle RToken cash (or withdrawable vault liquidity). Thus a reserve with ordinary high utilization can hold sufficient RToken backstop while finalization reverts atomically for insufficient pool liquidity; the no-vault path has the same requirement. The submitted five-test real-stack PoC passes, including a control liquidation with cash, rollback on the stressed liquidation, and the independent DEToken withdrawal path that can reduce the RToken backstop below the required shortfall.

Impact: Underwater positions remain latched and their debt continues accruing while managers cannot finalize them; suppliers also cannot redeem against the same unavailable cash. A borrower can temporarily consume the idle buffer before a manager transaction, and an SP depositor can permissionlessly exit through the timelock, causing repeated finalization failures. This can keep bad debt unresolved and leave the reserve/backstop insolvent during stress. This is distinct from the accounting loss after a successful liquidation: the failure occurs before the RToken redemption and does not depend on a DEToken haircut.

— RECOMMENDATION

Burn the stability pool's rToken for the gap, and do not withdraw pmUSD to do it. The liquidation then no longer depends on idle cash. If the stability pool does not hold enough rToken, either keep the liquidation reverting, or complete it and leave the uncovered amount as a shortfall for rToken holders. The treasury can refill that shortfall by transferring pmUSD to the rToken contract without minting new rToken, and can restore the stability pool by transferring rToken to it directly.

L - StabilityPool liquidation requires idle cash, blocking bad-debt resolution at high utilization CONTINUED

04

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/stabilitypool-liquidation-requires-idle-cash-blocking-bad-de-bbc10f3b.js>

— RESOLUTION

Status: Resolved.

raac.io Team: We've updated the liquidations page to cover this: <https://docs.raac.io/stability-pool-liquidations/> (see "Covering a shortfall")

LOW **L-05**

RESOLVED

Duplicate same-token adapters alias positions and let an attacker erase debt while draining the StabilityPool

CATEGORY access-control

LOCATION packages/contracts/contracts/core/pools/LendingPool/ERC20AssetAdapter.sol:ERC20AssetAdapter::getPositionKey:L147-L149

REVIEW Main Review

DESCRIPTION

If a trusted adapter migration registers a second ERC20AssetAdapter for the same collateral token without removing the first, both adapters share `getPositionKey() = keccak256(token)` while maintaining independent collateral balances. `LendingPool.registerAdapter` only rejects duplicate adapter addresses and has no token-uniqueness guard. An unprivileged attacker can deposit dust through one adapter and real collateral through the other, borrow against the real balance, initiate liquidation through the dust adapter because its health check is independently evaluated, and have settlement seize only dust while burning the entire aliased debt. The untouched balance remains withdrawable after settlement.

Impact: Under this configuration prerequisite, any caller can repeatedly cause StabilityPool repayment of the attacker's full debt while only dust collateral is seized, retain the borrowed crvUSD and real collateral, and drain depositor backing/DEToken solvency. The shared liquidation latch also freezes rescue and borrowing operations through the other adapter. The shipped deployment registers only one ERC20 adapter, and registration is `PROTOCOL_ROLE`-only, so this is a configuration/migration hardening issue and is appropriately Medium rather than a default-deployment permissionless High finding.

RECOMMENDATION

Track the underlying token for each registered adapter and reject a second adapter for the same token, or require atomic unregistering of the superseded adapter during migration. Also include the adapter identity in each adapter's position key so accidental duplicate registration cannot alias debt/liquidation state.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/duplicate-same-token-adapters-alias-positions-and-let-an-att-bf84aca0.js>

L - Duplicate same-token adapters alias positions and let an attacker erase **CONTINUED**
05 debt while draining the StabilityPool

— **RESOLUTION**

Status: Resolved.

raac.io Team: The docs have been updated. We mention what type of adapters should be added.

LOW **L-06**

RESOLVED

Supply cap bypassed and avoids deposit fee

CATEGORY	Configuration
LOCATION	LendingPool.sol:307-309
REVIEW	Main Review

— DESCRIPTION

The lending pool accepts deposits as soon as it is deployed. It is not paused. The constructor writes a supply cap of 10,000,000 pmUSD and never writes a deposit fee, so the fee stays 0 (`LendingPool.sol:233-241`).

The launch values are a 100,000 pmUSD cap and a fee of 5, which is 0.05% (`config-global.js:37` and `:40`). They are written later, each in its own transaction, after `RToken.setLendingPool` opens minting (`deployer.js:433` , then `:444` and `:447`). Each call waits for a receipt. On Ethereum that is several blocks before the supply cap changes.

`LendingPool.deposit` charges `parameters.depositFee` , then checks the cap on the amount after the fee (`LendingPool.sol:307-309` and `:332-333`). In that window both use the constructor values. The fee collector is already set. The parameter is 0.

A supplier can deposit 200,000 pmUSD then. They receive 200,000 rToken and pay exactly 200,000. The fee collector receives nothing. The same call can reach 10,000,000.

After the script sets the fee to 0.05% and the cap to 100,000, the 200,000 stays. A later deposit reverts. The protocol never collects the 0.05%. On 200,000 that is 100 pmUSD. On a full 10,000,000 fill it is 5,000 pmUSD, and later suppliers are blocked until the balance falls below 100,000.

— RECOMMENDATION

Call `_pause()` at the end of the LendingPool constructor, and call `unpause()` only after `linkLendingPool` has written the launch parameters. `deposit` already reverts while the pool is paused (`LendingPool.sol:303`).

Alternatively, consider updating the default values in the constructor, and avoid having to send extra config transactions to save gas.

— RESOLUTION

Status: Resolved.

L-06 Supply cap bypassed and avoids deposit fee**CONTINUED**

<https://github.com/gundam-foundation/contracts/pull/546/changes>

LOW **L-07**

ACKNOWLEDGED

Unguarded tier-0 oracle calls bypass fallbacks and freeze valuation-dependent operations

CATEGORY	denial-of-service
LOCATION	packages/contracts/contracts/core/oracles/CrvUSDToUSDOracle.sol:_crvUSDPriceInUSD:L109-L125
REVIEW	Main Review

— DESCRIPTION

The production tier-0 Chainlink path invokes `latestRoundData()`, `decimals()`, and normalization without `try/catch` or saturating arithmetic. A feed revert or normalization overflow therefore bubbles out instead of reaching the configured fallback. The PoC demonstrates this on the production-shaped iRAAC collateral route and LiquidationSwap, with healthy controls; valuation-dependent operations and NFT pricing also revert.

Impact: A feed outage or malformed answer temporarily freezes borrow, collateral valuation and health reads, indebted-position withdrawals, liquidation initiation, swaps used by liquidation routing, and RAACNFT mint pricing. RToken supplier withdrawals, debt repayment, deposits, and debt-free collateral withdrawals remain available. This is a recoverable availability/frozen-liquidation issue; no direct loss is demonstrated.

— RECOMMENDATION

Guard the tier-0 `latestRoundData()` and `decimals()` calls and treat failed or unusable responses as a failed tier so the configured fallback can run. Check decimal-normalization bounds before arithmetic.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/unguarded-tier-0-oracle-calls-bypass-fallbacks-and-freeze-va-e426071f.js>

— RESOLUTION

Status: Acknowledged.

LOW **L-08**

RESOLVED

LiquidationFinalized reports zero collateral for ERC20 liquidations

CATEGORY	logic/event-accounting
LOCATION	packages/contracts/contracts/core/pools/LendingPool/LiquidationProxy.sol:finalizeLiquidation:L135-L150
REVIEW	Main Review

DESCRIPTION

finalizeLiquidation transfers the borrower's entire ERC20 balance before calling getAssetValue for the LiquidationFinalized event. transferAsset zeroes the adapter balance, so every ERC20 liquidation emits collateralLiquidated as zero despite transferring the collateral to the Stability Pool.

Impact: The canonical LiquidationFinalized event misreports seized ERC20 collateral as zero. Indexers, monitoring, and reconciliation systems that derive liquidation amounts from this event cannot account for Stability Pool collateral intake. No direct on-chain fund loss is shown.

RECOMMENDATION

Read and cache the collateral value before transferAsset, then emit the cached value (or otherwise compute it from the pre-transfer balance).

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/liquidationfinalized-reports-zero-collateral-for-erc20-liqui-fcbe0303.js>

RESOLUTION

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/547/changes>

LOW **L-09**

RESOLVED

Unbounded ERC721 position enumeration can exhaust gas for debt reads and borrowing

CATEGORY	denial-of-service
LOCATION	packages/contracts/contracts/core/pools/LendingPool/ERC721AssetAdapter.sol:getPositionKeys:L159-L166
REVIEW	Main Review

DESCRIPTION

ERC721AssetAdapter records every NFT deposited by a user in an uncapped array, and getPositionKeys materializes and hashes the entire array. LendingPool.getPositionsScaledDebt enumerates those keys; DebtToken.balanceOf delegates to that aggregate calculation, and DebtToken.mint calls balanceOf on a new borrow. The PoC shows approximately linear gas growth and an account with 1,000 deposited NFTs exceeding the 16,777,216 per-transaction gas cap for the aggregate debt read and a subsequent borrow.

Impact: An account with an exceptionally large NFT portfolio can make its own aggregate DebtToken.balanceOf and further borrowing run out of gas. The effect is self-scoped and recoverable: another user cannot enlarge the list, repayment and liquidation remain position-specific, and debt-free NFTs can be withdrawn individually. The PoC reaches this count using privileged zero-price minting; buying roughly 1,000 priced property NFTs would be expensive. NFT lending is not planned for the current launch. No lender loss, liquidation failure, trapped funds, or protocol-wide denial of service was demonstrated.

RECOMMENDATION

Document how aggregate debt reads and new borrowing scale with a user's deposited NFT count, including the supported portfolio size. If large NFT portfolios are expected, avoid the full-account enumeration in the borrow path and provide a paginated position-read API.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/unbounded-erc721-position-enumeration-can-exhaust-gas-for-de-9288f20d.js>

RESOLUTION

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/558/changes>

LOW **L-10**

ACKNOWLEDGED

Blacklisted spender bypasses RToken compliance via transferFrom allowances

CATEGORY	compliance/access-control
LOCATION	packages/contracts/contracts/core/tokens/RToken.sol:_update:L324-L331
REVIEW	Main Review

— DESCRIPTION

RToken._update() (RToken.sol:324-329) and DEToken._update() (DEToken.sol:281-286) validate only from and to addresses. ERC20.transferFrom spends the caller's allowance but invokes `_update(sender, recipient, amount)`, so a blacklisted caller is never checked. Neither token checks the spender in approve or in the inherited ERC20Permit.permit, so a blacklisted spender can also obtain a fresh allowance from a holder's off-chain signature after being blacklisted and spend it in the same transaction. The PoC blacklists the spender after a clean holder approves it, shows that a direct transfer reverts, and shows that transferFrom(victim, clean, fullBalance) succeeds.

Impact: A sanctioned/blacklisted operator with a pre-existing allowance can continue moving the approving holder's RToken to arbitrary un-blacklisted destinations via transferFrom, bypassing ComplianceRegistry's sanctions enforcement on outflows. This is a material compliance-control bypass enabling sanctions evasion, not direct theft beyond the allowance the clean holder granted. isBlacklisted() is not only the protocol's own list: on mainnet, ComplianceRegistry is wired to the Chainalysis sanctions oracle, and OFAC-listed addresses such as `0x7F367cC41522cE07553e823bf3be79A889DEbe1B` and `0x098B716B8Aaf21512996dC57EB0615e2383E2f96` return true today. Those addresses are blocked from holding and transferring RToken/DEToken, but not from spending them as an approved operator. Regulated tokens close this path explicitly: USDC's FiatToken applies notBlacklisted(msg.sender) on transferFrom (alongside from and to) and on approve.

— RECOMMENDATION

Check complianceRegistry.isBlacklisted(msg.sender) in transferFrom before spending the allowance in both RToken and DEToken (or pass the spender into _update and validate it); consider applying the same compliance policy to approve/permit, while still allowing approve(spender, 0) so holders can revoke allowances.

L - Blacklisted spender bypasses RToken compliance via transferFrom **CONTINUED**
10 allowances

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/blacklisted-spender-bypasses-rtoken-compliance-via-transferf-560dbc97.js>

— **RESOLUTION**

Status: Acknowledged.

LOW **L-11**

RESOLVED

Residual RToken cash has no defined claim path

CATEGORY	funds-locking
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/tokens/RToken.sol#L353-L375
REVIEW	Main Review

— DESCRIPTION

After borrowers repay, protocol fees are claimed, and suppliers withdraw, reserve asset can remain in RToken. `calculateDustAmount()` displays cash above RToken `totalSupply()`, but `rescueToken()` rejects the reserve asset and the normal fee claim transfers only the recorded pending fee. The remaining cash has no defined beneficiary or routine claim path.

The owner can temporarily set itself as RToken's lending pool and use the uncapped `transferAsset()` to move funds. That requires changing a live authorization pointer and manually deciding how much is safe to take. `calculateDustAmount()` is not a safe withdrawal limit while claims remain: with 100 cash, 80 RToken supply, and 10 pending fees, it reports 20 as dust although only 10 is unallocated after the fee claim.

— RECOMMENDATION

Be aware of and document whether residual cash is meant to remain as extra backing or be allocated to a recipient. If recovery is intended, provide a bounded path based on all outstanding RToken and fee claims. Do not authorize a sweep using `calculateDustAmount()` alone.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/rtoken-lacks-a-dedicated-dust-recovery-path-surplus-is-only--f320931d.js>

— RESOLUTION

Status: Resolved.

LOW **L-12**

RESOLVED

Harvest can make vault rebalance revert

CATEGORY DoS

LOCATION <https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/VaultProxy.sol#L293-L302>

REVIEW Main Review

— DESCRIPTION

When a pmUSD savings vault is configured, `VaultProxy.rebalanceLiquidity()` chooses `withdrawAmount = min(shortage, maxWithdraw)` before calling `withdrawFromVault()`. The latter first harvests yield by redeeming vault shares, then rechecks the now lower `maxWithdraw` against the original amount. A routine deposit or protocol-fee claim can therefore revert during the optional buffer rebalance even when nearly all of the requested amount remains withdrawable.

In a controlled replay with a 100% buffer, the requested withdrawal exceeded the post-harvest limit by two wei and the entire transaction reverted. This requires a configured vault whose withdrawal limit falls when yield shares are redeemed. The launch plan leaves the savings vault unset; the replay uses a simplified vault mock, so the exact rounding behavior has not been verified against a future pmUSD vault.

— RECOMMENDATION

Be aware of and document this behavior before enabling a pmUSD savings vault. For optional buffer rebalancing, determine the withdrawable amount after harvesting and cap the rebalance to that updated amount. Retain strict liquidity checks for exact user payouts.

— RESOLUTION

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/548/changes>

LOW **L-13**

PARTIALLY RESOLVED

Liquidation rebalances the vault before repayment

CATEGORY	accounting
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/LiquidationProxy.sol#L148-L149
REVIEW	Main Review

— DESCRIPTION

LiquidationProxy credits reserve.totalLiquidity with the liquidation debt before the StabilityPool transfers reserve assets. LendingPool.finalizeLiquidation immediately rebalances against that higher total, so a funded vault with a buffer below 100% is asked to release part of its principal before the repayment arrives. In the real-stack PoC, a reserve whose buffer was on target withdraws 30,000 from the vault before a 150,000 debt repayment at a 20% buffer. A separate yield-bearing state shows that the pre-harvest versus post-harvest maxWithdraw mismatch can make this withdrawal revert.

Impact: In the demonstrated RToken-funded liquidation, the extra vault withdrawal is an intermediate movement of cash within the same transaction. The StabilityPool then redeems RTokens and transfers repayment to RToken, restoring the target buffer. For the stated strategy-free vault with no exit penalty, this movement does not realize a withdrawal fee, slippage, or loss of depositor principal. Its relevant consequence is an additional vault call that may hit a withdrawal limit; the post-harvest limit failure is covered by the separate 'Harvest can make vault rebalance revert' draft. No independent lasting fund loss is shown.

— RECOMMENDATION

Be aware of and document that finalization can call the yield vault before the Stability Pool returns reserve cash, even when the subsequent RToken-funded settlement restores the buffer in the same transaction. Keep the liquidation settlement independent of optional vault rebalancing if the selected vault's withdrawal limit can change during harvest.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/phantom-liquidation-credit-prematurely-drains-vault-and-can--8c1c85cb.js>

— RESOLUTION

Status: Partially resolved.

L-13 Liquidation rebalances the vault before repayment**CONTINUED**

Guardian: The rebalance was removed from `finalizeLiquidation`, but the `StabilityPool`'s `withdraw` still rebalances against the repayment before it arrives, so the vault is still drawn on early.

<https://github.com/gundam-foundation/contracts/pull/557/changes> , <https://github.com/gundam-foundation/contracts/pull/563>

LOW **L-14**

RESOLVED

Insecure production deployment configuration

CATEGORY Configuration

LOCATION `deploy_aws.js:16`

REVIEW Main Review

DESCRIPTION

Independent defects in the mainnet deploy scripts:

1. **Sepolia oracle config.** `deploy_aws.js:16` builds the oracle deployer from `configs.network.sepolia`. On mainnet it sets a feed with no code, hands ownership to the Safe, then reverts in its own `getLatestPrice` check.
2. **Oracle deployed instead of failing.** With `TOKEN_TO_USD_ORACLE` empty, `deployer.js:118-122` treats network `hardhat` as a local run. The documented command has no `--network`, yet transactions go to mainnet through the script's own `RPC_URL` signer. It deploys a feedless `PmUSDToUSDOracle` returning \$1, which `transferOwnership` skips as "already owned by multisig"; `LiquidationSwap` and `RWAIndexTokenOracle` use it.
3. **Fees to a destroyed key.** The process destroys the deploy key, yet `config-global.js:62-66` makes it the repair, burn, veRAAC, RAAC Corp and other recipient: 75% of lending yield, all NFT sale proceeds and LP rewards. Fees distributed before the Safe repoints them are lost.
4. **Not the KMS key.** `deploy_beta_aws.js:14-26` signs with a raw `PRIVATE_KEY` from Secrets Manager, so disabling the KMS key does not retire it.
5. **Exchange-rate feed.** Feed `0x922a...` ("PMUSD / USD Exchange Rate") was exactly 1.0 in all 210 daily rounds, so the 0.99-1.01 breaker is unlikely to see a depeg.
6. **Minor.** `precheck` funds 25M gas; the deploy uses at least 31M. The multisig note names `setPmUSDToUSDOracle`, which does not exist (`RAACHousePrices.sol:191`).

RECOMMENDATION

1. Load `configs.network.mainnet` in the oracle script.
2. Stop when `TOKEN_TO_USD_ORACLE` is empty instead of deploying one.
3. Refuse to run while the owner is not the Safe or a fee recipient is the deployer.
4. Sign with the KMS signer, as the oracle script does.

L-14 Insecure production deployment configuration

CONTINUED

5. Use a market-price pmUSD feed, or document that depegs are not detected.
6. Raise the gas budget and fix the setter name in the multisig note.

— RESOLUTION

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/549/changes>

INFORMATIONAL I-01

RESOLVED

Insurance fee omits accrued interest

CATEGORY	Logical Error
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/LendingPool.sol#L495-L508
REVIEW	Main Review

DESCRIPTION

`borrowWithInsurance` quotes the premium from `getUninsuredBalance` plus the new principal before `_borrow`. That helper is `rawDebtBalance - rawInsuredBalance`. It does not include pending interest.

`_borrow` then mints debt. `DebtToken.mint` capitalizes `scaledDebt - rawDebt` as `userIncrease` and the pool adds that amount to `rawDebtBalance`. The wrapper still charges the old quote, adds only `uninsuredBalance + amount` to `rawInsuredBalance`, and sets `isInsured` to true. Liquidation grace, close, and repay while flagged all check that flag, not the insured ledger, so the uncharged interest receives the same 3 day repayment window as the paid notional.

The path needs existing debt and time since the last mint or burn. A first insured borrow has no `userIncrease`. The caller must still borrow at least `MIN_BORROW_AMOUNT` and stay inside `borrowThreshold`.

Docs say this call insures the entire outstanding debt. After accrual it does not. The fee collector misses `userIncrease * insuranceFee`.

Raw uninsured debt 1000 with a 10% index rise is scaled 1100. Borrowing 10 more charges $(1000 + 10) * 3\% = 30.3$ instead of $(1100 + 10) * 3\% = 33.3$. After the call, raw debt is 1110, raw insured is 1010, and `isInsured` is true.

```
uint256 uninsuredBalance = getUninsuredBalance(adapter, msg.sender, data);
uint256 insuranceFeeAmount = calculateInsuranceFee(adapter, msg.sender, data,
    amount);
...
_borrow(adapter, data, amount);
...
position.isInsured = true;
position.rawInsuredBalance += uninsuredBalance + amount;
```

RECOMMENDATION

When `isInsured` is set, the collected premium and `rawInsuredBalance` must cover current outstanding debt, including pending interest. Include scaled uninsured debt, or the mint

I-01 Insurance fee omits accrued interest

CONTINUED

`userIncrease`, in the fee and in `rawInsuredBalance` so the premium matches the debt that receives grace.

— RESOLUTION

Status: Resolved.

INFORMATIONAL I-02

RESOLVED

Documented incident burn of collateralized NFT leaves unsecured, forcibly-unrecoverable debt

CATEGORY	collateral-accounting
LOCATION	packages/contracts/contracts/core/pools/LendingPool/ERC721AssetAdapter.sol:deposit:L64-L78
REVIEW	Main Review

DESCRIPTION

RAACNFT.burn is the documented security-incident burn operation, but it destroys an adapter-custodied token without notifying ERC721AssetAdapter or clearing its valuation/custody state. Existing debt remains validated, while liquidation and withdrawal reach safeTransferFrom on the nonexistent token and revert. The PoC also confirms voluntary repayment remains possible in the burn-only path, but forcible seizure cannot complete; zeroing the quote does not repair the stuck debt. If reminting is allowed, redeposit can additionally overwrite the reverse owner, freeze the prior position, and double-count the ID.

Impact: An incident burn of an indebted collateral NFT leaves an unsecured loan that cannot be forcibly recovered through normal on-chain liquidation or withdrawal; only optional borrower repayment can clear it. On default, reserve/RTOKEN suppliers bear the debt, and a stale nonzero quote can permit further borrowing. The remint/redeposit path can freeze voluntary repayment and create two debts against one NFT. Recovery requires privileged state/custody repair.

RECOMMENDATION

Consider documenting this scenario in the burn function natspec, so the admin role addresses do not burn NFTs held by the ERC721 adapter.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/documented-incident-burn-of-collateralized-nft-leaves-unsecured-a4ba842e.js>

RESOLUTION

Status: Resolved.

INFORMATIONAL I-03

RESOLVED

Protocol fees are paid from depositor principal before borrower interest is realised

CATEGORY	economic-accounting
LOCATION	packages/contracts/contracts/libraries/pools/LendingPoolFeeLibrary.sol:claimProtocolFees:L42-L45
REVIEW	Main Review

DESCRIPTION

Interest accounting creates a pending protocol fee from the outstanding debt index even though no borrower cash has entered the pool. A fee claim then ensures liquidity by withdrawing from the RToken buffer or vault and transfers that amount to the collector. With a zero buffer, this directly burns vault shares and removes depositor principal while the debt remains unpaid.

Impact: The production-entry-point PoC shows a 1108.59 crvUSD claim funded entirely by vaulted depositor cash while DebtToken supply is unchanged. After the same external collateral-price collapse and borrower non-payment, full supplier withdrawal reverts in both controls, but the world that claimed first recovers exactly 1108.59 crvUSD less than the unclaimed world. Thus routine fee collection is senior to suppliers and crystallizes protocol cash before an accrued receivable is realized; suppliers absorb the corresponding loss when the receivable defaults.

RECOMMENDATION

First impression will be to make protocol fees withdrawable from interest actually collected, but this requires quite a few changes in the logic. Assuming a liquidity buffer below 100%, a simpler solution will be to only allow protocol fee claiming when there is enough available liquidity, instead of using the depositor's funds from the vault.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/protocol-fees-are-paid-from-depositor-principal-before-borrower-interest-is-realised-7a2ecbea.js>

RESOLUTION

Status: Resolved.

INFORMATIONAL I-04

RESOLVED

Large liquidations miss the slippage floor

CATEGORY	DoS
LOCATION	LiquidationSwap.sol:71-75
REVIEW	Main Review

— DESCRIPTION

Liquidation sells the seized collateral in one trade. There is no partial close. The stability pool burns 2.5% of the index token, then sells 75%. The slippage protection makes the floor 1% under the vault price per share, and that price is read after the burn.

The burn raises the share price, so the floor moves up. A large sale also moves the pool due to price impact. Those two effects stack. At the deploy setting of 100 bps, a small sale in the liquidation flow still clears. A sale that is large relative to the pool liquidity does not. The exchange reverts, and the liquidation reverts with it. Most of the gap is price impact. The burn is a smaller push in the same direction.

One single position can borrow a significant piece borrow cap. Nothing else limits that debt, apart from the position's own collateral. The result is a position big enough that can't be liquidated

— RECOMMENDATION

Add a maximum debt per position, separate from the global borrow cap. Set it so a full liquidation of that position still clears the 1% floor after the 2.5% burn and the 75% sale. One position can borrow the whole global cap, so that cap alone does not bound the trade.

— RESOLUTION

Status: Resolved.

INFORMATIONAL I-05

RESOLVED

Public liquidation debt helper reverts for all direct callers

CATEGORY	availability
LOCATION	packages/contracts/contracts/core/pools/LendingPool/LiquidationProxy.sol:getPositionScaledDebtProxy:L37-L39
REVIEW	Main Review

DESCRIPTION

getPositionScaledDebtProxy is exposed as a public view function without onlyProxy, yet forwards via ILendingPool(address(this)).getPositionScaledDebt(...). A standalone LiquidationProxy has no such selector, so every direct ABI invocation reverts instead of returning a value or the descriptive guard error used by other proxy entrypoints.

Impact: The helper function is unusable for direct integrations, monitoring and simulation callers, and its failure mode is an opaque revert. The PoC deploys the real stack and confirms the pool implementation works while direct proxy calls revert. No current on-chain flow loses funds, so impact is limited to low-severity API availability/correctness.

RECOMMENDATION

Make the helper function internal. If a direct integration API is intended, forward to a stored LendingPool address rather than address(this) or create a separate helper public function away from the proxy contract.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/public-liquidation-debt-helper-reverts-for-all-direct-caller-231fc7b8.js>

RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. A fix has been applied.

<https://github.com/gundam-foundation/contracts/pull/550/changes>

INFORMATIONAL I-06

ACKNOWLEDGED

Non-receiver contract accounts cannot withdraw ERC721 collateral

CATEGORY	availability
LOCATION	packages/contracts/contracts/core/pools/LendingPool/ERC721AssetAdapter.sol:withdraw:L81-L86
REVIEW	Main Review

DESCRIPTION

ERC721AssetAdapter.deposit accepts a contract account as the depositor and pulls its NFT with transferFrom, without checking IERC721Receiver support. On withdrawal, the adapter always calls safeTransferFrom back to that same account, and LendingPool.withdrawAsset offers no alternate recipient. A non-receiver account therefore cannot use the ordinary debt-free withdrawal path; the transaction reverts and the NFT remains in the adapter. An upgradeable account could later add the receiver callback. NFT lending is not planned for the current launch.

Impact: This is a self-scoped compatibility risk for a contract account that chooses to deposit an NFT it cannot safely receive back. No third-party theft or protocol-wide loss is demonstrated.

RECOMMENDATION

Document that contract accounts depositing ERC721 collateral must implement IERC721Receiver and that ordinary withdrawals return the NFT only to the depositor. If non-receiver accounts are intended to be supported, add a withdrawal recipient chosen by the depositor.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/erc721-withdrawal-permanently-locks-nfts-deposited-by-non-re-3cf7fa44.js>

RESOLUTION

Status: Acknowledged.

raac.io Team: Hello, thanks for this finding. It is the responsibility of the depositor to ensure that the account is a valid NFT receiver. We don't believe that the adapter should not be responsible for self sabotaging actions of external actors. The current implementation will remain as is.

INFORMATIONAL I-07

RESOLVED

InsuredAsset emits only the pre-existing uninsured balance

CATEGORY	data-integrity
LOCATION	packages/contracts/contracts/core/pools/LendingPool/LendingPool.sol:borrowWithInsurance:L508-L510
REVIEW	Main Review

DESCRIPTION

`borrowWithInsurance` increases `rawInsuredBalance` by `uninsuredBalance + amount` but emits `InsuredAsset` with `uninsuredBalance` only. The first insured borrow therefore emits amount 0, and later insured borrows omit the newly borrowed insured principal.

Impact: Consumers that build insurance/risk/accounting state from `InsuredAsset` logs under-report newly insured debt. The on-chain position is correct and no direct fund loss is shown, so the impact is limited to off-chain data integrity.

RECOMMENDATION

Document that `InsuredAsset.amount` reports only the previously uninsured balance converted by `borrowWithInsurance`; the separate `Borrow` event reports new principal. If the event is intended to report all debt newly insured by the call, emit `uninsuredBalance + amount` instead.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/insuredasset-emits-only-the-pre-existing-uninsured-balance-bea5e86d.js>

RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. A fix will be applied to ensure the `InsuredAsset` event emits correct data.

<https://github.com/gundam-foundation/contracts/pull/551/changes>

INFORMATIONAL I-08

RESOLVED

FeeCollector pause blocks liquidations

CATEGORY	DoS
LOCATION	LiquidationStrategyProxy.sol:153-160
REVIEW	Main Review

DESCRIPTION

Every liquidation sends 20% of the minted iREET, the liquidity reward leg, through

`IFeeCollector(feeCollector).collectFee(...)` with no `try/catch` (`LiquidationStrategyProxy.sol:153-160`, split in `config-global.js:10-15`).

`collectFee` is `whenNotPaused` (`FeeCollector.sol:318`). It also reverts if the iREET token or the `RWA_INDEX_TOKEN_LIQUIDITY_REWARDS` fee type is not registered (`:320-321`). Any of these reverts the whole `liquidateBorrower`.

`EMERGENCY_ROLE` can pause the FeeCollector at any time, and `emergencyWithdraw` only works while paused (`:430`). The collector's own incident flow keeps liquidations down for as long as the incident lasts.

The docs say "liquidateBorrower deliberately stays available while paused". The LendingPool lets the StabilityPool bypass its pause "for liquidations" (`LendingPool1.sol:301`, `:342`). The FeeCollector pause has no such exception, and the docs never mention a FeeCollector call on the liquidation path.

The fix for "LendingPool.pause() blocks StabilityPool.liquidateBorrower via updateState" does not cover this. To liquidate, the team must unpause, liquidate and re-pause in one transaction, or change the split before each liquidation. Neither is documented.

RECOMMENDATION

Wrap the `collectFee` call in `try/catch`. On failure, send the leg to the Treasury or keep it in the StabilityPool for a later sweep, so a collector-side pause or configuration cannot block a liquidation. Document the behavior next to the existing pause carve-outs.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-09

RESOLVED

NFT flagged while price update is pending

CATEGORY Oracle
LOCATION LiquidationProxy.sol:48-68
REVIEW Main Review

DESCRIPTION

A house is marked pending only when its on-chain price differs from the valuation the CRE will write. A pending price is known to be outdated.

`initiateLiquidation` values NFT collateral through `RAACHousePrices.getLatestPrice` (`ERC721AssetAdapter.sol:117-121`), which ignores `isPendingRequest` (`RAACHousePrices.sol:201-209`). Anyone can flag a position on the outdated price while the pending valuation is healthy.

An uninsured borrower then cannot repay or close (`LendingPool.sol:592`), and the flag is never re-evaluated. The house can be seized after the update lands. The docs recommend a settlement guard in consuming protocols for pending prices.

RECOMMENDATION

Revert `initiateLiquidation` for ERC721 collateral while `isPendingRequest(tokenId)` is true. If this is the intended behavior, consider documenting the liquidation scenario.

RESOLUTION

Status: Resolved.

INFORMATIONAL **I-10**

PARTIALLY RESOLVED

Incorrect NatSpec and documentation

CATEGORY	Documentation
LOCATION	LendingPool.sol:736-757
REVIEW	Main Review

DESCRIPTION

Each item below is text that contradicts what the code does. None changes on-chain behaviour, but integrators and reviewers who rely on it get the wrong units, callers or behaviour.

Wrong on the docs site (and in the code's NatSpec):

- `calculateHealthFactor` returns "RAY, 1e27" (docs `lending-pool-functions` ; contract NatSpec `LendingPool.sol:740`). Indebted positions return 1e18-scaled values. Only the zero-debt value is RAY.
- The liquidity leg is deposited "into the Curve Gauge" (docs `stability-pool-liquidations` , `stability-pool-functions` ; contract NatSpec `LiquidationStrategyProxy.sol:151`). The code calls `FeeCollector.collectFee` .
- `tokenOfOwnerByIndex` documents both params as "The NFT tokenId" and the return as "Address of the user" (docs `lending-pool-asset-adapter-interface` ; interface NatSpec `IERC721Adapter.sol:20-25`). It takes a user and an index and returns a tokenId.

Wrong in code comments only:

- [contract NatSpec] `setRWAVault` and `setLiquidationVaultTokenSplit` say "Sets the IndexToken contract address" (`StabilityPool.sol:272,297`). The split struct says "Index token minting fee" (`StabilityPoolStorage.sol:28`).
- [contract NatSpec] `getPendingProtocolFee` is documented as the normalized income (`LendingPool.sol:852-853`).
- [contract NatSpec] `DebtToken.decimals` is documented as the reserve asset address (`DebtToken.sol:224-225`).
- [interface NatSpec] `rawBalanceOf` and `rawTotalSupply` say "scaled" (`IRToken.sol:62-73`). They return non-scaled values, as the contract NatSpec says (`RToken.sol:227-241`).
- [interface NatSpec] `mint` and `burn` say the first return value is scaled (`IRToken.sol:19,37`). It is the underlying amount (`RToken.sol:122,165`).

I-10 Incorrect NatSpec and documentation

CONTINUED

- [contract NatSpec] `RToken.transfer` and `transferFrom` "use scaled amounts" (`RToken.sol:200,210`). Both call `super` unchanged.
- [contract NatSpec] `RToken.rescueToken` is "Only callable by the Reserve Pool" (`RToken.sol:366`). It is `onlyOwner` .
- [interface NatSpec] `setParameter` is "Only callable by contract owner" (`ILendingPool.sol:177`). The pool has no owner and gates by role.
- [contract NatSpec] `claimCollectorRewards` "Claims CRV rewards from the vault" (`LendingPool.sol:1112`). It transfers booked reserve-asset yield.
- [contract NatSpec] `LiquidationStrategyProxy.liquidateBorrower` is callable by "a manager or the owner when the contract is not paused" (`:36`). It is `onlyProxy` , and the StabilityPool entry has no pause check.
- [contract NatSpec] `finalizeLiquidation` says "If there is no debt, it will revert" (`LiquidationProxy.sol:112`). A zero-debt burn returns early (`DebtToken.sol:155-157`).
- [inline comment] `_repay` says amounts above the debt are capped (`LendingPool.sol:600`). Only `type(uint256).max` is capped.
- [contract NatSpec] `setLiquidityPool` sets an "address and fee", with "10000 for 100%" protection (`LiquidationSwap.sol:93,95`). No fee is set, and 10000 gives `minDy = 0` .
- [contract and interface NatSpec] `withdraw` mentions RAAC rewards (`StabilityPool.sol:124,200` , `IStabilityPool.sol:8,20`). None exist.
- [contract NatSpec] `ReserveData` stores "all values in RAY" (`ReserveLibrary.sol:33`). Liquidity and usage are in asset units.
- [contract NatSpec] `calculateLinearInterest` returns "the new liquidity index" (`ReserveLibrary.sol:177-181`). It returns an interest factor and ignores `lastIndex` .
- [contract NatSpec] `calculateBorrowRate` works "within the maxRate and baseRate window" (`ReserveLibrary.sol:298`). Below the kink the rate is flat at `optimalRate` .
- [inline comment] `_mintUserInterest` says `_mint` updates `_rawTotalDeposits` (`DEToken.sol:266`). It does not.
- [contract NatSpec] `CrvUSDToUSDOracle.decimals` mentions "USD 8 decimals" (`:96`). It always returns 18.
- [inline comment] The `withdrawAll` header sends yield "to the reserve" and leftover shares "to the manager" (`VaultProxy.sol:73,75`). The code books yield as collector rewards and deposits the shares to the Treasury.

I-10 Incorrect NatSpec and documentation

CONTINUED

- [interface NatSpec] `transferAsset` is callable by "privileged modules" (`IAssetAdapter.sol:17`). Both adapters allow only the lending pool.

- [contract NatSpec] `withdrawFromVault` and `ensureLiquidity` are an "Internal function" (`VaultProxy.sol:229,310`). Both are public or external.

— RECOMMENDATION

Update each listed NatSpec, comment and docs-site entry to describe what the code does.

— RESOLUTION

Status: Partially resolved.

raac.io Team: Hello, thanks for this finding. A fix has been applied

Guardian: 16 of 26 items fixed. Still wrong in code: `tokenOfOwnerByIndex` return (`IERC721Adapter.sol:17`), `getPendingProtocolFee` "normalized income" (`LendingPool.sol:856`), `IRToken` "scaled total supply" (:70), `RToken` transfer "scaled amounts" (:200), `SP` withdraw "RAAC rewards" (`StabilityPool.sol:124`, `IStabilityPool.sol:8`), `IAssetAdapter` "privileged modules" (:18). Still wrong on the live docs: "RAY, 1e27" (`lending-pool-functions`), "Curve Gauge" (`stability-pool-liquidations`, `stability-pool-functions`), "Address of the user" (`lending-pool-asset-adapter-interface`).

<https://github.com/gundam-foundation/contracts/pull/556> , <https://github.com/gundam-foundation/contracts/pull/562/changes>

INFORMATIONAL I-11

RESOLVED

Unused code

CATEGORY	Superfluous Code
LOCATION	StabilityPool.sol:142-152
REVIEW	Main Review

DESCRIPTION

The items below have no production reader or caller. Each was searched across `contracts/`, `deployment/`, `library/` and `test/`. They cost gas or mislead readers.

Storage written but never read:

- `WithdrawTimelock.expireAt` (`StabilityPool.sol:150`). Expiry uses `readyAt + withdrawTimelockDelay` (`:169`). The docs still describe `expireAt` as the end of the window.
- `WithdrawTimelock.userIndexAtRequest` (`StabilityPool.sol:146`), which also costs an external call per request.
- `vaultRewards.lastSharePrice` (`VaultProxy.sol:86,133`), only ever set to 0.
- `vaultRewards.lastDepositTime`, `lastWithdrawalTime` and `lastClaimTime` (`VaultProxy.sol:87,225,268`, `LendingPoolFeeLibrary.sol:59`), visible only through the public getter.

Code that never runs:

- `LendingPool.sol:350` comments out a call to `_executeWithdrawTimelock`, which is not defined anywhere. The comment on `:349` still describes a timelock.
- `LendingPool.sol:413` sets the health factor to RAY when `positionDebt < 1`, inside `if (positionDebt > 0)` (`:408`). It is unreachable.
- `DEToken.setUserIndex` (`DEToken.sol:95-97`) and `DEToken.transferAsset` (`:163-165`) have no caller outside tests.
- `onlyManager` (`StabilityPool.sol:78-81`) is never applied.

Declarations never used:

- Errors never thrown: `CircuitBreakerTriggered` (`CrvUSDToUSDOracle.sol:278`, the breaker clamps instead), `TimeDeltaIsZero` (`ReserveLibrary.sol:110`), `NoDust` (`IRToken`), and 27 in `ILendingPool`, `ILendingPoolStorage` and `IStabilityPool`, such as `NFTAlreadyDeposited`, `MarketNotFound`, `NoRewardsToClaim` and `RepayAmountTooHigh`.

I-11 Unused code

CONTINUED

- Events never emitted: `RewardDistributed` (`VaultProxy.sol:354`), the market and allocation events (`IStabilityPool.sol:150-209`), `LiquidityIndexUpdated`, `BalanceTransfer`, `BurnerSet`, `DustTransferred` (`IRToken`), `UsageIndexUpdated` (`IDebtToken`), and `ProtocolFeesCollected`, `ProtocolFeesWithdrawn`, `RewardTokensDeposited`, `VaultWithdrawalFailed` (`ILendingPool`).
- Unused imports: `Address`, `IBaseCollector`, `IComplianceRegistry`, `DebtToken` and `RToken` in `LendingPool.sol`; `ILendingPool` and `IAssetAdapter` in `VaultProxy.sol`; `SafeERC20` in `LiquidationProxy.sol`; `using SafeERC20 for IDToken` in `StabilityPool.sol`.
- Library helpers with no caller: `percentDiv`, `wadDiv`, `rayPow` and `WadRayMath.SECONDS_PER_YEAR`.

— RECOMMENDATION

Remove the unused imports, errors, events and helpers, the unreachable branch at `LendingPool.sol:413` and the commented-out timelock call. Keep the unused storage fields, because `StabilityPool` is upgradeable and the proxy modules share storage through `delegatecall`. Mark them as unused in the `NatSpec`, and correct the docs entry for `expireAt`.

— RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. The unused elements raised will be assessed and removed where necessary.

<https://github.com/gundam-foundation/contracts/pull/555/changes>

INFORMATIONAL I-12

RESOLVED

YieldHarvested reports zero priceIncrease on every harvest

CATEGORY	data-integrity
LOCATION	packages/contracts/contracts/core/pools/LendingPool/VaultProxy.sol:_harvestYield:L144-L189
REVIEW	Main Review

DESCRIPTION

VaultProxy._harvestYield emits YieldHarvested(yieldHarvested, 0, _getPricePerShare()) after a successful harvest. The event documents priceIncrease as the share-price change, but this field is always zero, and lastSharePrice is never updated after its initialization. A consumer that relies on priceIncrease alone will miss share-price appreciation. The event still provides yieldAmount and newSharePrice, and this mismatch does not alter on-chain balances. The PoC triggers two harvests with positive yield and rising newSharePrice; both report priceIncrease == 0.

RECOMMENDATION

Document that YieldHarvested.priceIncrease is currently always zero, so consumers should derive a share-price change from successive newSharePrice values if needed. Alternatively, populate priceIncrease from a tracked prior share price.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/yieldharvested-emits-zero-priceincrease-on-every-harvest-531b4ceb.js>

RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. A fix has been applied

<https://github.com/gundam-foundation/contracts/pull/552/changes>

INFORMATIONAL I-13

RESOLVED

Admin setters accept out-of-range values

CATEGORY	Validation
LOCATION	LendingPool.sol:134-176
REVIEW	Main Review

DESCRIPTION

`LendingPool.setParameter` stores the buffer ratio, insurance fee, deposit fee, protocol fee rate, optimal utilization, prime-rate step limit and vault opening fee unchecked, and bounds `LiquidationThreshold` only by `borrowThreshold`. `StabilityPool.setWithdrawTimelockDelay` and the stale-time and `K` setters of `CrvUSDToUSDOracle` and its child `PmUSDToUSDOracle` are unchecked too.

A bad value breaks or distorts a core flow. A deposit fee above 100% underflows `amount - depositFeeAmount` (`LendingPool.sol:308`). An insurance fee above 100% charges more than the borrow (`LendingPool.sol:536`). A buffer ratio above 100% asks for more than the vault holds after harvest, so `withdrawFromVault` reverts (`VaultProxy.sol:245`) and every pool operation that rebalances reverts with it. A withdraw delay of 0 expires every `StabilityPool` withdraw request (`StabilityPool.sol:169`). A stale time of 0 pins both oracles to the fixed fallback.

No setter relates `liquidationThreshold` to the liquidation split. Debt is repaid only by swapping a share of the seized index tokens (`LiquidationStrategyProxy.sol:65-83`), so a threshold above swap share $\times (1 - \text{slippage})$ makes every liquidation a `StabilityPool` shortfall, not only bad debt. Deployed 60% is safe (limit 74.6%); the 80% default is not.

RECOMMENDATION

Reject the values that break operations: a deposit fee or buffer ratio above `100_00`, a protocol fee rate above `RAY`, and a zero withdraw delay. Document the operating ranges for the other fees, the oracle stale times and `K`, and how `liquidationThreshold` relates to the liquidation split's swap share.

RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. A fix has been applied

Guardian: Not bounded: `StabilityPool.setWithdrawTimelockDelay` accepts 0

I-13 Admin setters accept out-of-range values

CONTINUED

<https://github.com/gundam-foundation/contracts/pull/553/changes>

INFORMATIONAL I - 14

RESOLVED

LiquidityRebalanced reports pre-rebalance buffer and tracked principal

CATEGORY	Events
LOCATION	https://github.com/gundam-foundation/contracts/blob/7efdd353829d289f4650da5292578076f03e5d6b/contracts/core/pools/LendingPool/VaultProxy.sol#L281-L307
REVIEW	Main Review

DESCRIPTION

VaultProxy.rebalanceLiquidity reads currentBuffer before it deposits into or withdraws from the vault, then emits that old value in LiquidityRebalanced after the operation. Its vaultAmount field is vaultRewards.totalDeposits, a tracked principal value rather than the vault's current redeemable asset value. For example, moving 800,000 of a 1,000,000 deposit to the vault still emits a 1,000,000 buffer. A consumer treating the event as a post-rebalance allocation can therefore be misled; the event does not affect on-chain balances, and no dependent consumer is identified.

RECOMMENDATION

Be aware of and document that LiquidityRebalanced currently reports a pre-operation buffer and tracked vault principal. Consumers needing the final allocation should read actual balances and vault conversion state. If the event is intended to report final allocation, emit post-operation cash and current vault asset value.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-RAAC-Core-Contracts--Round-1-01a0fd596c0970e183821ee528114a91/blob/refinery/poc-01a0fd59-846d-7e81-bae6-19cc643b2734/test/refinery-poc/liquidityrebalanced-emits-stale-buffer-and-principal-instead-1b2bdb13.js>

RESOLUTION

Status: Resolved.

raac.io Team: Hello, thanks for this finding. A fix has been applied

Guardian: The buffer is now post-rebalance. vaultAmount is still vaultRewards.totalDeposits

<https://github.com/gundam-foundation/contracts/pull/554/changes>

INFORMATIONAL I-01

RESOLVED

Liquidation still draws on the vault before repayment arrives

CATEGORY Gas Optimization
LOCATION LendingPool.sol:364
REVIEW Remediation Review

DESCRIPTION

The fix removed the rebalance from `finalizeLiquidation` and added a final `rebalanceLiquidity()` (`LiquidationStrategyProxy.sol:92`). The early vault draw it targeted still happens, one step later.

`finalizeLiquidation` still credits `reserve.totalLiquidity` with the burned debt before any cash arrives (`LiquidationProxy.sol:152`). In an RToken-funded liquidation, the `StabilityPool` then calls `lendingPool.withdraw` (`LiquidationStrategyProxy.sol:78`). `withdraw` ends with `_rebalanceLiquidity()` (`LendingPool.sol:364`), which sizes the buffer against the inflated total and pulls the difference from the vault. The repayment then lands in the RToken (`:81`), and the final rebalance deposits the same amount back.

As a result, every RToken-funded liquidation withdraws from the vault and then deposits the same amount back within one transaction, each leg with its own harvest. No funds are lost with a vault that has no exit fee, and the withdrawal no longer reverts. With a vault that charges on exit or limits withdrawals, the round trip pays that fee or consumes that limit for nothing.

RECOMMENDATION

Skip the rebalance in `withdraw` when the caller is the `StabilityPool`, and let the final `rebalanceLiquidity()` in `LiquidationStrategyProxy` settle the buffer once. Liquidation is the `StabilityPool`'s only call to `withdraw`, and every liquidation path already ends with a rebalance.

RESOLUTION

Status: Resolved.

<https://github.com/gundam-foundation/contracts/pull/563/changes>

§ 05

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.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.