GA GUARDIAN

Strata

Tranches

Security Assessment

October 10th, 2025

Summary

Audit Firm Guardian

Prepared By Robert Reigada, Cosine, Ciphky, Michael Lett

Client Firm Strata

Final Report Date October 10, 2025

Audit Summary

Strata engaged Guardian to review the security of their Strata Tranches Contracts. From the 22nd of September to the 29th of September, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report. **Note:** Fixes to the findings uncovered in remediations (included starting on page 42) have not been reviewed by Guardian.

Confidence Ranking

Given the number of High and Critical issues detected as well as additional code changes made after the main review, and to address findings from the remediation review. Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. The Strata team has accepted this recommendation and is conducting a follow up review with another firm.

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

📊 PoC test suite: https://github.com/GuardianOrg/contracts-tranches-team1-1758218379250

Table of Contents

Project Overview

Project Summary

Project Name	Strata
Language	Solidity
Codebase	https://github.com/Strata-Money/contracts-tranches
Commit(s)	Initial commit: 7ebd6eb19a9d7e0b1747ff4bee1ac55d41fc3d1a Final commit: 1057bdedbcecf9a3b0724228ce007c2544fa80b9

Audit Summary

Delivery Date	October 10, 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	5	0	0	0	0	5
Medium	14	0	0	0	0	14
Low	5	0	0	0	0	5
Info	8	0	0	3	0	5

Audit Scope & Methodology

Scope and details:

contract,source,total,comment
contracts/tranches/StrataCDO.sol,79,106,8
contracts/tranches/StrataCDOStorage.sol,56,129,55
contracts/tranches/YieldAccounting.sol,206,286,22
contracts/governance/AccessControlManager.sol,32,93,53
contracts/governance/AccessControlled.sol,47,91,27
contracts/governance/StrataMasterChef.sol,8,13,1
contracts/tranches/utils/UD60x18Extra.sol,24,35,5
contracts/tranches/strategies/Strategy.sol,9,13,1
contracts/tranches/oracles/AprTupleFeed.sol,111,143,2
contracts/tranches/base/CDOComponent.sol,16,23,2
contracts/tranches/base/Tranche.sol,158,209,24
contracts/tranches/strategies/ethena/IsUSDe.sol,8,12,1
contracts/tranches/strategies/ethena/sUSDeCooldownRequestImpl.sol,40,52,1
contracts/tranches/strategies/ethena/sUSDeStrategy.sol,101,130,7
contracts/tranches/base/cooldown/ERC20Cooldown.sol,75,95,6
contracts/tranches/base/cooldown/UnstakeCooldown.sol,107,141,11
source count: {
  total: 1571,
  source: 1077,
  comment: 226,
  single: 121,
  block: 105,
  mixed: 5,
  empty: 274,
  todo: 4,
  blockEmpty: 1,
  commentToSourceRatio: 0.20984215413184773
}

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 Strata, 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
GLOB-01	Total NAV should equal sum of tranche NAVs and reserves	✅	✅	✅	10m+
GLOB-02	Strategy total assets should equal sum of tranche assets	✅	✅	✅	10m+
GLOB-03	ERC20 cooldown SUSDe balance should equal or greater than total ERC20 cooldown amount	✅	✅	✅	10m+
TRANCHE-01	After mint or deposit, the total assets should increase and the total supply should increase	✅	✅	✅	10m+
TRANCHE-02	After withdraw or redeem, the total assets should decrease and the total supply should decrease	✅	✅	✅	10m+
COOLDOWN-01	After cooldown request, the strategy total assets should decrease and the erc20CooldownSUSDe balance should increase	✅	✅	✅	10m+
COOLDOWN-02	After cooldown finalize, the erc20CooldownSUSDe balance should decrease	✅	✅	✅	10m+
COOLDOWN-03	After cooldown finalize, the unstakeCooldownTotalAmount should decrease	✅	✅	✅	10m+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
COOLDOWN-04	After cooldown request, the strategy total assets should decrease and the unstakeCooldownTotalAmount should increase	✅	✅	✅	10m+
STRATEGY-01	srtTargetIndex must strictly increase when dt>0 and aprSrt>0	✅	✅	✅	10m+
STRATEGY-02	Junior tranche NAV should be within 2 wei of jrtTotalAssets	✅	✅	✅	10m+
STRATEGY-03	Senior tranche NAV should be within 2 wei of srtTotalAssets	✅	✅	✅	10m+
STRATEGY-04	Total NAV should be within 2 wei of strategy total assets	✅	✅	✅	10m+
STRATEGY-05	Total assets should decrease after reduce reserve	✅	❌	✅	10m+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Reserve Withdrawal Unit Mismatch	Logical Error	Critical	Resolved
H-01	Withdraw Griefing DoS	DoS	High	Resolved
H-02	Withdrawals Flip Token And Base Assets	Logical Error	High	Resolved
H-03	Juniors Do Not Profit On Loss	Logical Error	High	Resolved
H-04	JRT totalSupply Will Grow Constantly	DoS	High	Resolved
H-05	MEV APR Front-Run Via onAprChanged	MEV	High	Resolved
M-01	setMinimumJrtSrtRatio Mutates	Logical Error	Medium	Resolved
M-02	grantCall Gives DEFAULT_ADMIN Role	Validation	Medium	Resolved
M-03	JRT maxWithdraw Redemption DoS	DoS	Medium	Resolved
M-04	Missing updateAccounting Call	Rewards	Medium	Resolved
M-05	Redeem Mistreats Shares As Assets	Logical Error	Medium	Resolved
M-06	Old Data Can Overwrite Newer Data	Validation	Medium	Resolved
M-07	JRT maxMint Reverts When Share Price < 1	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-08	Cooldown Removal Ignores Pending Requests	Validation	Medium	Resolved
M-09	Old Implementations Remain Active	Logical Error	Medium	Resolved
M-10	Risk > 100% Freezes Senior APR Math	DoS	Medium	Resolved
M-11	APR Updates Rewrite The Previous Accrual Window	Logical Error	Medium	Resolved
M-12	Negative APRs Break Accounting Updates	Validation	Medium	Resolved
M-13	reduceReserve Uses Stale Accounting	Logical Error	Medium	Resolved
L-01	Incorrect Validation In setProvider Function	Validation	Low	Resolved
L-02	Feed Updates Skip Accounting Refresh	Configuration	Low	Resolved
L-03	Max Limits Ignore Min Shares Guard	Compatibility	Low	Resolved
L-04	Missing Event In Critical Function	Best Practices	Low	Resolved
L-05	ERC20Cooldown Accepts Any Token	Validation	Low	Resolved
I-01	STR APR Can Be Stale	Warning	Info	Resolved
I-02	Compound VS Linear Comment Mismatch	Warning	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-03	Unused Code	Best Practices	Info	Resolved
I-04	Typos	Informational	Info	Resolved
I-05	SRT Yield Is Not Guaranteed	Warning	Info	Acknowledged

C-01 | Reserve Withdrawal Unit Mismatch

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

Description PoC

StrataCDO.reduceReserve debits accounting by the requested base-asset amount, then asks the strategy to send that many tokens to the treasury. In the USDe path, sUSDeStrategy.reduceReserve forwards the same value to unstakeCooldown.transfer as if it were sUSDe shares:

```
// StrataCDO.sol
function reduceReserve (address token, uint256 tokenAmount) external onlyRole(RESERVE_MANAGER_ROLE) {
if (treasury = address(0)) {
revert ZeroAddress();
}
// Reverts if the token is not supported
uint256 baseAssets = strategy.convertToAssets(token, tokenAmount, Math.Rounding.Floor);
// Reverts if not enough reserve
accounting.reduceReserve(baseAssets);
// Transfers tokens out instantly if possible, or through the cooldown process
strategy.reduceReserve(token, tokenAmount, treasury);
emit ReserveReduced(token, tokenAmount);
}
// sUSDeStrategy.sol
function reduceReserve (address token, uint256 tokenAmount, address receiver) external onlyCDO {
if (token = address(sUSDe)) {
erc20Cooldown.transfer(sUSDe, receiver, tokenAmount, 0);
return;
}
if (token = address(USDe)) {
unstakeCooldown.transfer(sUSDe, receiver, tokenAmount);
return;
}
revert UnsupportedToken(token);
}
```

When the share price exceeds 1 (normal after yield accrues), the helper only needs previewWithdraw(baseAssets) shares, but the strategy is forced to deliver the full base-asset amount as shares. In the reproduced run, withdrawing 1 USDe from the reserve led the strategy to transfer 1 full share even though the correct amount was ~0.53 shares at the prevailing exchange rate. After cooldown the treasury would receive ~1.889 USDe while accounting deducted only 1 USDe. This totally breaks the internal accounting.

Recommendation

Before calling unstakeCooldown.transfer for token = USDe, compute the share amount with uint256 shares = sUSDe.previewWithdraw(tokenAmount); (or convertToShares) and supply shares instead of tokenAmount, keeping the accounting units consistent.

Resolution

Strata Team: Resolved.

H-01 | Withdraw Griefing DoS

Category	Severity	Location	Status
DoS	High	UnstakeCooldown.sol: 46-105	Resolved

Description PoC

• Anyone is able to use the `transfer` function of the `ERC20Cooldown` or `UnstakeCooldown` contract to create new proxies for an arbitrary recipient and lock up tokens

• If this arbitrary recipient later on calls `finalize` the system will loop through all of these proxies.

This enables a permanent DoS griefing attack:

• Bob performs a normal withdraw flow in the system to redeem $100k

• Eve creates a lot of proxies with Bob as recipient locking up 1 wei each

• After a week bob wants to claim his $100k and calls `finalize` but the transaction runs out of gas as the loop is too big

Recommendation

Add access control to these functions.

Resolution

Strata Team: Resolved.

H-02 | Withdrawals Flip Token And Base Assets

Category	Severity	Location	Status
Logical Error	High	Tranche.sol: 205	Resolved

Description

The call `cdo.withdraw(address(this), token, baseAssets, tokenAssets, receiver);` passes `baseAssets` and `tokenAssets` in the wrong order to `StrataCDO.withdraw` which expect the following parameter order:

`function withdraw(address tranche, address token, uint256 tokenAmount, uint256 baseAssets, address receiver)`

Consequently, the strategy interprets USDe-denominated value as the specified token amounts, which will ultimately lead to users receiving less funds than they are owed within the withdrawal flow.

Recommendation

Replace the existing call with `cdo.withdraw(address(this), token, tokenAssets, baseAssets, receiver);` instead.

Resolution

Strata Team: Resolved.

H-03 | Juniors Do Not Profit On Loss

Category	Severity	Location	Status
Logical Error	High	Accounting.sol: 256	Resolved

Description

Within function `calculateNAVSplit`, if the SRT reports a loss (case `if (srtGainTarget < 0)`), the loss should be transferred to the Juniors as profit according to the comment.

However, instead of adding the loss to `jrtNavT1` which represents the updated Junior NAV, the loss is added to the old Junior NAV: `jrtNavT0 + srtLoss;`.

Consequently, the loss by the Seniors is not deducted from the Senior NAV and it is also not added as profit to the Junior NAV, violating the spec for each Tranche.

Recommendation

Use `srtNavT1 - srtLoss;` and `jrtNavT1 + srtLoss;` instead.

Resolution

Strata Team: Resolved.

H-04 | JRT totalSupply Will Grow Constantly

Category	Severity	Location	Status
DoS	High	Accounting.sol	Resolved

Description PoC

The senior-first reallocation policy, as applied in Accounting.calculateNAVSplit, can drive junior NAV (JRT) down to a hard floor while leaving JRT totalSupply unchanged whenever the senior target jumps because a feed update or normal parameter changes. The code path is:

AprPairFeed.updateRoundData → Accounting.onAprChanged → updateAprs/updateIndexes → Tranche.deposit → cdo.updateAccounting → Accounting.updateAccounting/calculateNAVSplit.

Inside calculateNAVSplit, the senior gain target derived from the index jump is pulled from the junior side first and clamped only by a 1-token floor:

```
uint256 srtGainTargetAbs = Math.min(
uint256(srtGainTarget),
Math.saturatingSub(jrtNavT1, 1e18) // hard floor
);
jrtNavT1 = jrtNavT1 - srtGainTargetAbs; // JRT can collapse to 1e18
srtNavT1 = srtNavT0 + srtGainTargetAbs;
```

After this, JRT.totalAssets ≈ 1e18 (the floor) while JRT.totalSupply is unchanged. The price per JRT share becomes smaller. On the very next deposit, OpenZeppelin's ERC-4626 share conversion:

```
shares = floor(assets * (totalSupply + 1) / (totalAssets + 1));
```

multiplies assets by a very high ratio (S{+}1)/(A{+}1), minting a very high amount of shares. Repeated deposits at tiny price cause runaway growth of totalSupply. For sufficiently large deposits and, sufficiently enough APR increases, the product assets * (S+1) exceeds 256 bits and Math.mulDiv reverts, DoS'ing deposits. Even before hard reverts, the system inflates supply massively.

Importantly, this is reachable under normal operations, not just extreme spikes. For example, mild APR oscillations (e.g., 1.0% ↔ 1.5%) can ratchet junior down over time: each uptick to 1.5% transfers incremental value from JRT to SRT (bounded only by the 1-token floor), while the downtick back to 1.0% does not claw anything back from SRT.

With continued deposits during these cycles, JRT NAV trends toward its floor, price per share shrinks and eventually deposits at ordinary sizes mint huge share amounts and can overflow during conversion. Thus, even "small" changes accumulated over days/weeks can lead to a runaway-supply / deposit-overflow state. The main impact of this issue, JRT price collapse leads to unbounded share minting and, at realistic sizes, ERC-4626 deposit conversion overflow/reverts, threatening protocol liveness.

Recommendation

Consider mitigating the junior-runaway condition by enforcing hard and soft Junior/Senior TVL ratio checks within the Accounting contract. On the other hand, consider adding an automatic junior-price guard in the StrataCDO contract that re-computes the JRT share price after every flow and pauses deposits into both tranches whenever it slips below a configurable jrtShortfallPausePrice, preventing the low-price/high-supply regime that previously led to overflows

Resolution

Strata Team: Resolved.

H-05 | MEV APR Front-Run Via onAprChanged

Category	Severity	Location	Status
MEV	High	Accounting.sol	Resolved

Description PoC

When the APR feed is updated, keepers call Accounting.onAprChanged(), which immediately fetches the new pair (aprTarget, aprBase) and re-computes the senior APR (aprSrt) and rolls the accrual index/clock based on the current tranche balances stored in jrtNav/srtNav.

An attacker can front-run this keeper transaction with a large, temporary ERC-4626 deposit into the Tranche that favors their objective (typically Senior), skewing the ratio srtNav / (srtNav + jrtNav) at the instant the keeper's call lands, and then back-run a withdrawal to fully exit.

The manipulated aprSrt and newly reset indexTimestamp persist for the next accrual segment, even though the attacker did not keep capital in the system.

Because onAprChanged() is a public mempool transaction (role-gated, but visible), the attacker can reliably insert a deposit before it and a withdraw after it to obtain a long-lived APR haircut favorable to their position, while only tying up capital for the block (or for the cooldown window, if configured).

This can be abused so the senior APR and accrual clock can be repeatedly biased at each feed update, shifting future period gains between tranches without the manipulator retaining the skewing capital within the protocol.

Recommendation

Do not recompute aprSrt or roll the index in onAprChanged(). Treat the feed update as a pure data refresh and move APR/clock updates to a checkpoint that operates on durable post-flow balances:

• Modify onAprChanged()/updateAprs() to only cache aprTarget/aprBase:

```
function onAprChanged() external onlyRole(UPDATER_FEED_ROLE) {
IAprPairFeed.TRound memory r = aprPairFeed.latestRoundData();
aprTarget = normalizeAprFromFeed(r.aprTarget);
aprBase   = normalizeAprFromFeed(r.aprBase);
// do NOT call updateIndexes here
}
```

• At accounting checkpoints, first settle the elapsed period by rolling the index with the previous aprSrt, then apply any balance flows, then compute the new aprSrt from the updated jrtNav/srtNav without rolling the index again (the new aprSrt applies going forward).

Resolution

Strata Team: Resolved.

M-01 | setMinimumJrtSrtRatio Mutates

Category	Severity	Location	Status
Logical Error	Medium	Accounting.sol	Resolved

Description

In the Accounting contract, the configuration routine intended to update the junior/senior TVL safeguard writes the incoming value to `reserveBps` and emits a `ReservePercentageChanged(reserveBps)` event.

The actual guardrail variable, `minimumJrtSrtRatio` is left untouched. Downstream logic that enforces the floor, for example `uint256 minJrt = srtNav * minimumJrtSrtRatio / 1e18;` and `uint256 maxSrt = jrtNav * 1e18 / minimumJrtSrtRatio;`, now continues to use the stale default.

Meanwhile, the reserve split is silently overwritten with the requested ratio, because `setReserveBps` and `setMinimumJrtSrtRatio` both mutate `reserveBps` and emit the same `ReservePercentageChanged` event.

Because of this, risk managers can not raise or lower the tranche. Tvl floor and reserve accounting is mutated unexpectedly.

Recommendation

Assign the provided value to `minimumJrtSrtRatio`, emit a dedicated event (for example `MinimumJrtSrtRatioChanged(bps)`) and leave `reserveBps` unchanged so reserve configuration remains isolated from the ratio guardrail.

Resolution

Strata Team: Resolved.

M-02 | grantCall Gives DEFAULT_ADMIN Role

Category	Severity	Location	Status
Validation	Medium	AccessControlManager.sol	Resolved

Description PoC

AccessControlManager.grantCall takes a contract address and selector, derives a role with roleFor and forwards the request to OpenZeppelin's grantRole. The helper packs the address in the high 20 bytes and the selector in the low 4 bytes:

```
role = (bytes32(uint256(uint160(contractAddress))) << 96) | bytes32(uint256(uint32(sel)));
```

When an administrator tries to grant a global permission by supplying contractAddress = address(0) and sel = bytes4(0), the computed role collapses to bytes32(0), which OpenZeppelin defines as DEFAULT_ADMIN_ROLE.

The NatSpec explicitly advertises this pattern: “if contractAddress is zero address, the account can access the specified function on any contract managed by this ACL”, and isAllowedToCall looks up roleFor(address(0), sel) as the fallback namespace.

Consequently, the intuitive call path grantCall(address(0), bytes4(0), user) silently elevates user to default admin, giving them total control over every role and contract managed by the ACL. Any mistaken global fallback grant therefore becomes a full protocol takeover vector.

Recommendation

Reject or special case the all-zero pair before delegating to grantRole. For example, revert when contractAddress = address(0) and sel = bytes4(0) or require admins to call grantRole(DEFAULT_ADMIN_ROLE, …) explicitly.

Resolution

Strata Team: Resolved.

M-03 | JRT maxWithdraw Redemption DoS

Category	Severity	Location	Status
DoS	Medium	Accounting.sol: 124-131	Resolved

Description

The `maxWithdraw` function can be abused to DoS junior tranche redemptions as a malicious actor could deposit the maximum possible into the senior tranche every time someone else withdraws from it or enters the junior tranche.

This could of course also happen accidentally in case the senior tranche is favored over the junior tranche by users.

Recommendation

Consider to either acknowledge this behavior or the introduce a deposit and withdraw queue to protect junior tranche depositors from this DoS vector.

Resolution

Strata Team: Resolved.

M-04 | Missing updateAccounting Call

Category	Severity	Location	Status
Rewards	Medium	Global	Resolved

Description PoC

The system does not call `updateAccounting` before updating parameters which will influence its outcome.

For example, the `setReserveBps` function will update the `reserveBps` parameter which influences the gain of the tranches.

As `updateAccounting` is not called in the beginning of the function this will not only influence future gains but also past ones and therefore suddenly influence yield some users may have already calculated with.

Recommendation

Always call `updateAccounting` before updating any parameter that will influence its outcome.

Resolution

Strata Team: Resolved.

M-05 | Redeem Mistreats Shares As Assets

Category	Severity	Location	Status
Logical Error	Medium	Tranche.sol: 150	Resolved

Description

Function `redeem` routes the redemption of shares through the Tranche's `withdraw` function: `uint256 assets = super.withdraw(shares, receiver, owner);`

However, function `withdraw` in `ERC-4626` expects an asset amount rather than the currently supplied share amount.

Consequently, users will receive less assets than they are owed for the shares they are redeeming and will have to complete multiple redemptions to redeem their shares.

Recommendation

Use `super.redeem(shares, receiver, owner)` within function `redeem`.

Resolution

Strata Team: Resolved.

M-06 | Old Data Can Overwrite Newer Data

Category	Severity	Location	Status
Validation	Medium	AprPairFeed.sol: 91-95	Resolved

Description

The `updateRoundData` pushes APR values by providing `aprTarget` and `aprBase` values. It saves them together with the current `block.timestamp`.

However, the values must not be from this `block.timestamp` they could also be laying around in the mempool for a while as not enough gas was provided.

If the needed amount of gas is reduced later on, the transaction might go through and overwrite newer data with stale one.

Recommendation

Consider supplying the `block.timestamp` as param and make sure that a older transaction does not overwrite a newer one.

Resolution

Strata Team: Resolved.

M-07 | JRT maxMint Reverts When Share Price < 1

Category	Severity	Location	Status
Logical Error	Medium	Tranche.sol	Resolved

Description PoC

`StrataCDO.maxDeposit(address(this))` returns `type(uint256).max` for the junior tranche. `Tranche.maxMint` passes that value to `_convertToShares`, which multiplies it by `totalSupply() + 1` and then divides by `totalAssets() + 1`.

Whenever the junior share price dips below 1 (i.e. `totalAssets < totalSupply` after juniors absorb a loss), the 512-bit multiplication produces a high word greater than the `denominator`.

`OpenZeppelin`'s `Math.mulDiv` detects this overflow and reverts with a `panic: arithmeticError` instead of returning a finite cap.

Because `ERC4626.mint` always calls `maxMint(receiver)` prior to minting, this revert bricks every mint call even if the user requests a tiny number of shares.

Recommendation

Consider returning `type(uint256).max` directly from `maxMint` instead, when “no deposit limit” is required.

Resolution

Strata Team: Resolved.

M-08 | Cooldown Removal Ignores Pending Requests

Category	Severity	Location	Status
Validation	Medium	UnstakeCooldown.sol: 85	Resolved

Description

The `transfer` function in the `UnstakeCooldown` contract manages unstaking requests with cooldown periods by transferring tokens to a proxy contract, initiating unstaking, and either immediately processing withdrawals (if no cooldown is required) or storing requests for later withdrawal.

Users then call `finalize` to complete unstaking once the cooldown expires. The issue arises when `cooldownDuration` is later set to zero. In this case, new unstaking requests can be withdrawn immediately, but existing pending requests remain locked.

This is because `finalize` validates requests using the originally stored `unlockAt` timestamp, preventing withdrawals until the original cooldown elapses, even though the cooldown is no longer active.

As a result, users who initiated unstaking requests before the cooldown was removed are unfairly stuck waiting. This could also cause problems if Ethena implements emergency measures that allow immediate withdrawals, since affected users would still be unable to access their funds.

Recommendation

Modify the cooldown validation in the finalize function to allow withdrawals if either the original unlock time has passed or the cooldown has since been disabled.

Resolution

Strata Team: Resolved.

M-09 | Old Implementations Remain Active

Category	Severity	Location	Status
Logical Error	Medium	UnstakeCooldown.sol: 146	Resolved

Description

The `setImplementations` function in the `UnstakeCooldown` contract allows the owner to update the implementation for a given token. This implementation is used as the proxy implementation contract that holds and processes user tokens during the cooldown period.

However, because the transfer and finalize functions are designed to reuse existing proxies, an outdated implementation remains valid even after a new implementation has been set.

This creates a potential issue if the old implementation contained a bug, became incompatible, or required deprecation. In such cases, funds could remain stuck or even be at risk if users continue interacting with the outdated proxy.

Recommendation

Add a version check mechanism to the proxy reuse logic in the transfer function, ensuring that only proxies created with the current implementation are reused from the pool, while outdated proxies are discarded and new ones are created instead.

Resolution

Strata Team: Resolved.

M-10 | Risk > 100% Freezes Senior APR Math

Category	Severity	Location	Status
DoS	Medium	Accounting.sol	Resolved

Description PoC

`Accounting.calculateRiskPremium()` derives a "risk" factor as `riskX + riskY * pow(tvlRatio, riskK)`, where `tvlRatio = srtNav / (srtNav + jrtNav)` and the three coefficients are admin-configurable. When `updateIndexes` runs, it applies that factor in `aprSrt1 = mul(aprBase_, UD60x18.wrap(1e18) - risk)`.

The math uses the unsigned `UD60x18` type, so 1e18 represents 100%. If runtime risk ever reaches or exceeds 1e18, the subtraction underflows and the transaction reverts.

The existing guard in `setRiskParameters` only checks the current risk given the current tranche split; it does not bound future values as the TVL mix evolves.

Consequently, choosing parameters where `riskX + riskY > 100%` looks safe while the senior pool is small, but the next deposit wave that drives `tvlRatio` close to 1 can push risk above 100%.

From that point on, every call to `updateIndexes` and, therefore every deposit, withdrawal, or explicit accounting update, reverts, freezing the protocol until an admin dials the parameters back down.

Recommendation

Consider adding two layers of protection. First, enforce a configuration invariant that keeps the theoretical maximum risk below 100%, e.g. `require riskX.unwrap() + riskY.unwrap() < 0.95e18` (with optional headroom) when setting parameters and bound `riskK` to a very sensible range.

Second, inside `updateIndexes`, check `if (risk.unwrap() > 1e18) revert RiskTooHigh(risk.unwrap());` or clamp the factor before subtracting; this turns the generic underflow into a clear failure mode and prevents the unsigned arithmetic panic from bricking every call.

Resolution

Strata Team: Resolved.

M-11 | APR Updates Rewrite The Previous Accrual Window

Category	Severity	Location	Status
Logical Error	Medium	Accounting.sol	Resolved

Description PoC

`getSrtTargetIndexT1()` expands the index over the elapsed interval `dt = block.timestamp - indexTimestamp` using the newly stored `aprSrt`.

Because the old APR was overwritten first, that dt window is retroactively capitalized at the new rate, not the one that was actually in force during the elapsed time.

In other words, the moment an updater supplies APR “B”, the entire period since the last accounting update is recomputed as if “B” had been active all along.

By choosing whether to report a higher or lower replacement APR, whoever controls the feed (or any account with `UPDATER_FEED_ROLE`) can push value between the Senior and Junior tranches for the just-finished interval.

That retroactive rewrite affects the next `calculateNAVSplit` output, so deposits, withdrawals, and accounting updates that follow inherit the new distribution.

Recommendation

Apply the old APR to the completed interval before committing the new one. For example:

1. Advance `srtTargetIndex` using the current `aprSrt` (as of the previous update).

2. Only after the index and `indexTimestamp` are updated, assign the new APR parameters that should take effect going forward.

Resolution

Strata Team: Resolved.

M-12 | Negative APRs Break Accounting Updates

Category	Severity	Location	Status
Validation	Medium	Accounting.sol, AprPairFeed.sol	Resolved

Description

The APR feed explicitly checks that pushed values stay between -50% and +200%:

```
require(APR_BOUNDARY_MIN < answer answer < APR_BOUNDARY_MAX, "INVALID_APR");
```

with APR_BOUNDARY_MIN = -0.5e12. Accounting, however, normalizes the same feed output with a tighter guard:

```
require(APR_BOUNDARY_MIN < apr apr < APR_BOUNDARY_MAX, "invalid apr");
```

where APR_BOUNDARY_MIN = 0.

When an operator publishes a negative APR (still within the feed's advertised bounds), normalizeAprFromFeed reverts and onAprChanged fails, so the new value is never applied and the system keeps using stale APRs.

This operational mismatch makes the live APR updates fragile and can leave the protocol out of sync with real market conditions when negative rates occur.

Recommendation

Align the allowed ranges across the two components. Either clamp negative APRs to zero (or otherwise sanitize them) inside Accounting before normalization, or narrow AprPairFeed's bounds to match Accounting so that impossible inputs are rejected at the feed boundary.

Resolution

Strata Team: Resolved.

M-13 | reduceReserve Uses Stale Accounting

Category	Severity	Location	Status
Logical Error	Medium	StrataCDO.sol: 193	Resolved

Description

The `reduceReserve` function can be called by an address with the `RESERVE_MANAGER_ROLE` to reduce the reserve and transfer tokens to the treasury. However, the function does not call `updateAccounting` beforehand, which means `reserveNav` may not reflect the latest state.

If `reserveNav` is understated, `reduceReserve` won't be able to withdraw up to the actual available amount. More critically, if `reserveNav` is overstated, losses have not yet been allocated (JRT → Reserve → SRT).

In this case, `reduceReserve` may withdraw more than it should. Later, when `updateAccounting` runs, the remaining reserve is smaller, and additional losses are pushed to the SRT tranche or can’t be fully covered.

Recommendation

Modify the `reduceReserve` function to call `updateAccounting` before reducing reserves to ensure `reserveNav` reflects the latest state.

Resolution

Strata Team: Resolved.

L-01 | Incorrect Validation In setProvider Function

Category	Severity	Location	Status
Validation	Low	AprPairFeed.sol	Resolved

Description

AprPairFeed.setProvider attempts to verify the replacement APR provider before wiring it into the feed, but the check is pointed at the wrong address. The function still calls provider.getAprPair(). It queries whatever implementation was already set—before assigning provider = provider_:

```
function setProvider(IStrategyAprPairProvider provider_) external onlyOwner {
// compatibility check
(int64 aprTarget, int64 aprBase, ) = provider.getAprPair();  // ← old provider
ensureValid(aprTarget);
ensureValid(aprBase);
provider = provider_;
emit ProviderSet(address(provider_));
}
```

Because the compatibility probe never hits provider_, any address passes the guard: the zero address, an EOA, or a misconfigured contract that returns invalid data.

Recommendation

Validate the new provider before assignment: ensure that provider_.getAprPair() succeeds with APRs inside [APR_BOUNDARY_MIN, APR_BOUNDARY_MAX]. Only after the checks pass should the code write provider = provider_.

If getAprPair() throws or returns out-of-range values, revert the transaction to keep the existing, working oracle.

Resolution

Strata Team: Resolved.

L-02 | Feed Updates Skip Accounting Refresh

Category	Severity	Location	Status
Configuration	Low	AprPairFeed.sol	Resolved

Description

`AprPairFeed.updateRoundData` only writes `latestRound`/`latestRoundId` and emits an `AnswerUpdated` event, but never informs consumers that fresh APRs arrived.

The accounting module depends on an explicit `Accounting.onAprChanged()` call to pull the new feed data and recompute `aprTarget`, `aprBase` and the derived indexes.

If the operator who updates the feed forgets this step, the feed and accounting drift: fresh APRs sit in the oracle while tranche math stays on stale rates until the second transaction happens, delaying the intended risk adjustments and yield targets.

Recommendation

Add an observer mechanism or helper that bundles the feed write with the accounting refresh. For example, have the feed call registered listeners' `onAprChanged()` after a successful update, or expose an orchestrator function that performs both steps atomically so operators can't leave the system desynchronized.

Resolution

Strata Team: Resolved.

L-03 | Max Limits Ignore Min Shares Guard

Category	Severity	Location	Status
Compatibility	Low	Tranche.sol	Resolved

Description

`Tranche.maxWithdraw` and `Tranche.maxRedeem` return values that take tranche caps into account but do not consider `_onAfterWithdrawalChecks`, which is invoked at the end of every `withdraw`/`redeem` call and reverts with `MinSharesViolation` whenever the post-withdraw total supply would fall below `MIN_SHARES`.

Because the advertised maxima omit this floor, integrators can see a non-zero limit, attempt a withdraw/redeem within that bound and still hit a revert.

`ERC-4626` explicitly requires each max helper to "MUST NOT be higher than the actual maximum that would be accepted," so the current behavior violates the `ERC-4626` standard and breaks downstream slippage/limit checks.

Recommendation

Incorporate the minimum-supply guard into `maxWithdraw`, `maxRedeem` and any preview helpers that rely on those values. One approach is to compute the hypothetical post-withdraw supply and, if it would drop below `MIN_SHARES`, reduce the advertised limit accordingly.

Resolution

Strata Team: Resolved.

L-04 | Missing Event In Critical Function

Category	Severity	Location	Status
Best Practices	Low	Accounting.sol: 350-360	Resolved

Description

The `setRiskParameters` functions performs a critical state change but does not emit an event.

Recommendation

Consider emitting events in all functions which perform critical state changes to follow best practices.

Resolution

Strata Team: Resolved.

L-05 | ERC20Cooldown Accepts Any Token

Category	Severity	Location	Status
Validation	Low	ERC20Cooldown.sol: 28	Resolved

Description

The transfer function in the `ERC20Cooldown` contract allows users to create a cooldown request. Currently, there are no restrictions on which tokens can be used, allowing users to create requests for spam or potentially malicious tokens.

While this does not directly impact funds, it can pollute the system. Additionally, the transfer and finalize functions do not specify the token in their emitted events, which can lead to corrupted event logs.

Recommendation

Consider restricting it to approved tokens, similar to the `UnstakeCooldown` contract.

Resolution

Strata Team: Resolved.

I-01 | STR APR Can Be Stale

Category	Severity	Location	Status
Warning	Info	Global	Resolved

Description

Senior tranche APR changes must be applied manually with the `onAprChanged` function. If this function is called too late, the yield is distributed in a unfair manner.

Recommendation

Make sure to call `onAprChanged` and `updateAccounting` very regularly especially at the 8h markers when funding fees are settled for Ethena's open positions. Otherwise the senior tranche APR will be stale and yield distribution will be unfair.

Resolution

Strata Team: Resolved.

I-02 | Compound VS Linear Comment Mismatch

Category	Severity	Location	Status
Warning	Info	Accounting.sol	Resolved

Description

`calculateTargetIndex` is documented as “using compound interest formula,” yet the implementation multiplies the prior index by `1 + apr * dt / YEAR`, which is simple interest.

Readers expecting compounding logic will misinterpret how the target index grows, making it harder to reason about the accrual model.

Recommendation

Adjust the comment to describe simple interest accurately, or switch the implementation to a true compound interest calculation if that was the intended behavior.

Resolution

Strata Team: Resolved.

I-03 | Unused Code

Category	Severity	Location	Status
Best Practices	Info	Global	Resolved

Description

There is unused code in multiple parts of the system. The `MathExt` library is not used at all.

Unused Imports:

• `MathExt` in `contracts/tranches/Accounting.sol`

• `IERC4626` `contracts/tranches/StrataCDO.sol`

• `IERC20`, `IERC4626`, `SafeERC20`, `IErrors`, `AccessControlled` in `contracts/tranches/Strategy.sol`

Recommendation

Consider removing unused code.

Resolution

Strata Team: Resolved.

I-04 | Typos

Category	Severity	Location	Status
Informational	Info	Global	Resolved

Description

• `event NewAccessControlManager(address accessControllManager);` - replace `accessControllManager` with `accessControlManager`

Recommendation

Fix the typos.

Resolution

Strata Team: Resolved.

I-05 | SRT Yield Is Not Guaranteed

Category	Severity	Location	Status
Warning	Info	Global	Acknowledged

Description

The docs state out that the senior tranches yield is guaranteed to be at minimum the SSR. However, this requires enough funds in the junior tranche to take the yield from. In case the NAV of the junior tranche approaches zero the yield is no longer guaranteed.

As the SSR and the yield from `sUSDe` are not related it is possible that the SSR yield will become bigger than the `sUSDe` yield in the future. In this case the JRT will shrink up to the point of the SRT yield no longer being guaranteed.

This risk is especially given in a bear market as Ethena is a very bullish protocol. The yield of `sUSDe` is mostly given by funding fees from the hedging short positions.

In a bear market where more perp's liquidity is on the short side than the long side these hedging positions actually need to pay funding fees instead of gaining them. This can drastically reduce the yield of `sUSDe` or even set it to 0 (or in the worst-case lead to a collapse of Ethena).

In such a bearish scenario it is likely that the SSR will outperform the `sUSDe` yield as its yield source is less reliant on general crypto market conditions.

Recommendation

Be aware of this worst case scenario, think about steps to take in that case and continuously analyze market conditions to be able to react quickly.

Resolution

Strata Team: Acknowledged.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
M-01	Treasury Payouts Can Be DoS	DoS	Medium	Resolved
I-01	MIN_SHARES Blocks Final Tranche Withdrawals	Configuration	Info	Acknowledged
I-02	Global Selector Fallback In ACL Reverts	Compatibility	Info	Acknowledged
I-03	IStrategy.withdraw Parameter Typo	Best Practices	Info	Resolved

M-01 | Treasury Payouts Can Be DoS

Category	Severity	Location	Status
DoS	Medium	ERC20Cooldown.sol	Resolved

Description

`ERC20Cooldown.transfer` stores each pending cooldown under `activeRequests[address(token)][to]` and refuses new cross-user requests once the array length reaches `PUBLIC_REQUEST_SLOTS_CAP`.

Any shareholder can withdraw through the tranche to an arbitrary receiver, so a malicious actor can enqueue 40 dust-sized cooldowns for a shared address by calling:

```
if (initialFrom = to requestsCount > PUBLIC_REQUEST_SLOTS_CAP) {
revert ExternalReceiverRequestLimitRiched(token, initialFrom, to, amount);
}
```

On the other hand, `StrataCDO.reduceReserve` sends senior withdrawals through `sUSDeStrategy.reduceReserve`, which enqueues the payout in `UnstakeCooldown.transfer` under `activeRequests[address(token)][treasury]`.

The cooldown engine enforces `PUBLIC_REQUEST_SLOTS_CAP = 40` for cross-user requests. An attacker can repeatedly withdraw a dust amount to the treasury address, filling the 40-slot queue with pending entries that mature only after the seven-day cooldown.

Once saturated, every subsequent `reduceReserve` call reverts with `ExternalReceiverRequestLimitRiched`, preventing the protocol from delivering reserve funds—even though the attacker only sacrificed trivial amounts they later recover. In practice this allows a single account to freeze the treasury payout pipeline for the entire cooldown period.

Because finalize only pops fully matured entries, the victim must wait a full cooldown duration (7 days by default) before capacity frees up. Until then, all further cross-user withdrawals to that `receiver` revert. The attacker sacrifices only minimal capital that the target eventually receives, so the DoS is cheap and repeatable.

Recommendation

Consider exposing an owner/operator function that lets the receiver prune third-party requests without waiting out the entire cooldown.

Resolution

Strata Team: Resolved.

I-01 | MIN_SHARES Blocks Final Tranche Withdrawals

Category	Severity	Location	Status
Configuration	Info	Tranche.sol	Acknowledged

Description

Tranche._onAfterWithdrawalChecks() reverts whenever totalSupply() drops below MIN_SHARES:

```
function _onAfterWithdrawalChecks () internal view {
if (totalSupply() < MIN_SHARES) {
revert MinSharesViolation();
}
}
```

The contract never seed-mints those 0.1 ether shares during initialize, nor anywhere else, so the supply floor is enforced on ordinary user balances.

As liquidity is drained (or even on first deposits if supply stays under 0.1 ether), the next withdrawal burns shares, sees totalSupply() < MIN_SHARES and reverts, leaving the remaining assets permanently locked.

Recommendation

Consider performing an initial deposit of exactly MIN_SHARES and setting the receiver to an irrecoverable burn address (e.g., address(1)), so those shares remain outside user balances while satisfying the floor.

Resolution

Strata Team: Acknowledged.

I-02 | Global Selector Fallback In ACL Reverts

Category	Severity	Location	Status
Compatibility	Info	AccessControlManager.sol	Acknowledged

Description

`AccessControlManager.isAllowedToCall` first checks a role scoped to the calling contract and then attempts a global fallback by invoking `roleFor(address(0), sel)`.

The helper enforces `require(contractAddress = address(0) sel = bytes4(0), "StrictPermissionOnly")`, so the fallback always reverts, making `grantCall(address(0), sel, …)` unusable.

Any guard that relies on `_checkAccessAllowed` will revert for globally granted selectors, preventing operators from using the policy documented in `grantCall`.

Recommendation

Merely informative, as this restriction was added as a fix to the M-02: Global Fallback `grantCall` gives `DEFAULT_ADMIN` role issue

Resolution

Strata Team: Acknowledged.

I-03 | IStrategy.withdraw Parameter Typo

Category	Severity	Location	Status
Best Practices	Info	IStrategy.sol	Resolved

Description

The interface `IStrategy.withdraw` declares the fourth argument as `bseAssets`, omitting the “a”, while every implementation and call site expects `baseAssets`, for example `sUSDeStrategy.withdraw`.

Recommendation

Rename the interface argument to `baseAssets`.

Resolution

Strata Team: Resolved.

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