

GLIDER · SECURITY REVIEW

Glider

Protocol Review.

A security review of Protocol Review.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Glider	Protocol Review	Roberto Reigada, Felipe Gomez, Nicholas Chew, Owen Thurm	May 29 – September 18, 2026	September 18, 2026

§ 00

Contents.

140 pages: overview, confidence ranking, scope, 95 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. 07
	Main Review · 61 findings	P. 11
	Remediation Review 1 · 22 findings	P. 98
	Remediation Review 2 · 12 findings	P. 123
§ 05	Disclaimer	P. 139
§ 06	About Guardian	P. 140

§ 01

Overview.

Guardian reviewed Protocol Review across 3 review rounds. The final report includes 1 Critical, 10 High, 28 Medium, 38 Low, 18 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/glider-fi/glider-contracts-audit
CLIENT	METHODOLOGY
Glider	Smart contract · Manual · Main Review · Remediation Review 1 · Remediation Review 2
PROJECT	MAIN REVIEW COMMIT
Protocol Review	aa620a8e9024
PREPARED BY	REMEDIATION REVIEW 1 COMMIT
Roberto Reigada, Felipe Gomez, Nicholas Chew, Owen Thurm	1ccfa80b6587
AUDIT WINDOW	REMEDIATION REVIEW 2 COMMIT
May 29 – September 18, 2026	7b4bfd594728
FINAL REPORT	
September 18, 2026	

1.1 Confidence Ranking

2 / 5

Low Confidence

Guardian assigns a Confidence Ranking of 2 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	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED	FIX REVIEW
● Critical	1	0	0	0	1	0
● High	10	2	2	2	4	0
● Medium	28	6	4	4	12	2
● Low	38	13	4	2	17	2
● Informational	18	4	2	1	3	8
All severities	95	25	12	9	37	12

§ 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	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 ASSIGNED	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 Protocol Review in <https://github.com/glider-fi/glider-contracts-audit>. The scope includes 1,387 source lines across 6 files.

ALSO IN SCOPE

External Scope [OWB Staking]: onchain-base-type4/contracts/Staking.sol

External Scope [OWB Staking]: onchain-base-type4/contracts/staking/StakingCore.sol

External Scope [OWB Staking]: onchain-base-type4/contracts/staking/StakingMethods.sol

External Scope [Kernel]: GliderScan/lib/kernel/src/Kernel.sol

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.

FILE	NSLOC	TOTAL
src/async-nav-vault/contracts/AsyncNavVault.sol	15	27
src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol	681	1,071
src/glider-kernel-validator/lib/GliderRootValidatorDigests.sol	86	167
src/glider-kernel-validator/lib/MultiOwnerStructs.sol	119	229
src/glider-kernel-validator/lib/StrictWebAuthn.sol	118	204
src/owb-staking-wrapper/contracts/OWBStakingWrapper.sol	368	623

§ 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					
C-01	YS vault depositors permanently stuck	DoS	CRITICAL	RESOLVED	p. 11
H-01	Refresh bypasses NAV freshness gate	MEV	HIGH	PENDING	p. 13
H-02	JIT claims skim returned yield	MEV	HIGH	PARTIALLY RESOLVED	p. 15
H-03	Epoch gate ignores NAV report freshness	Frontrunning	HIGH	PENDING	p. 17
H-04	Permissionless reward claims strand rewards	Frontrunning	HIGH	RESOLVED	p. 18
H-05	Matured claims capture loss recovery	Gaming	HIGH	RESOLVED	p. 20
H-06	Owner storage layout blocks all user operations	DoS	HIGH	RESOLVED	p. 22
M-01	Signed NAV snapshot has no recency bound	Signatures	MEDIUM	PENDING	p. 24
M-02	Loss freezes deposits, exits stay open	Logical Error	MEDIUM	RESOLVED	p. 25
M-03	NAV disclosure cancellation option	MEV	MEDIUM	PENDING	p. 26
M-04	Delayed deposit claims capture NAV gains	Unexpected Behavior	MEDIUM	PENDING	p. 28
M-05	Root reactivation ignores replacement owners	Unexpected Behavior	MEDIUM	ACKNOWLEDGED	p. 30
M-06	Target-epoch views double count claims	Unexpected Behavior	MEDIUM	PENDING	p. 32
M-07	Aave reward variant incompatible with Earn Vault	Compatibility	MEDIUM	RESOLVED	p. 33
M-08	Multichain validation uses banned opcode	Compatibility	MEDIUM	RESOLVED	p. 35
M-09	ERC1271 owners violate validation rules	DoS	MEDIUM	ACKNOWLEDGED	p. 38
M-10	Zero-supply strands unharvestable yield	Unexpected Behavior	MEDIUM	RESOLVED	p. 39
M-11	Fluid extra rewards unclaimable	Rewards	MEDIUM	RESOLVED	p. 40
M-12	Cross-chain ERC-1271 owner takeover	Access Control	MEDIUM	PENDING	p. 41
M-13	WebAuthn crossOrigin not validated	Signatures	MEDIUM	RESOLVED	p. 42
M-14	P-256 verifier choice is not signed	Unexpected Behavior	MEDIUM	RESOLVED	p. 43
M-15	NAV requests gameable pre-report	Gaming	MEDIUM	PENDING	p. 44

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
M-16	Activation-loss tolerance DoSes deposits	DoS	MEDIUM	RESOLVED	p. 46
L-01	Unstake yield orphaned on full exit	Rewards	LOW	RESOLVED	p. 48
L-02	Withdraw rounding leaks value to harvest	Rounding	LOW	RESOLVED	p. 49
L-03	Reward harvest assumes same-chain claim	Trust Assumptions	LOW	ACKNOWLEDGED	p. 50
L-04	NAV price drift overpays redeems	Trust Assumptions	LOW	PENDING	p. 51
L-05	Dust edits keep deposits reserved	DoS	LOW	PENDING	p. 52
L-06	Disabled manager strands yield, rewards	Validation	LOW	RESOLVED	p. 53
L-07	Root execution can erase validator owners	Unexpected Behavior	LOW	RESOLVED	p. 55
L-08	Withdrawals burn excess managed value	Unexpected Behavior	LOW	RESOLVED	p. 57
L-09	Deposit events attribute operator claims to the operator	Events	LOW	PENDING	p. 59
L-10	ERC165 misses the required ERC7575 vault id	Best Practices	LOW	PENDING	p. 60
L-11	ERC7575 share side is not advertised	Best Practices	LOW	RESOLVED	p. 61
L-12	Zero-amount exits can harvest rewards	Unexpected Behavior	LOW	RESOLVED	p. 62
L-13	AsyncNav share side lacks ERC7575 id	Best Practices	LOW	PENDING	p. 64
L-14	Share allowance can retarget redeems	Unexpected Behavior	LOW	PENDING	p. 65
L-15	Same-epoch redemption views overcommit liquidity	Unexpected Behavior	LOW	PENDING	p. 67
L-16	Sole address owner can become unusable	Unexpected Behavior	LOW	PENDING	p. 68
L-17	Asset deployment event omits manager	Events	LOW	PENDING	p. 69
L-18	Passkey removal also removes raw P-256 owners	Unexpected Behavior	LOW	RESOLVED	p. 71
L-19	Decrease events omit retargeted epoch	Events	LOW	PENDING	p. 72
L-20	Unbounded admin delays can block admin handoff	DoS	LOW	PENDING	p. 74
L-21	Chain-bound leaf version lacks SDK helper	Configuration	LOW	RESOLVED	p. 75
L-22	Public EIP-712 shows wrapper domain	Unexpected Behavior	LOW	PENDING	p. 77
L-23	Emptied NAV vault state not reset	Logical Error	LOW	PENDING	p. 78
L-24	Owner-mutation guard lacks origin bind	Access Control	LOW	ACKNOWLEDGED	p. 79
L-25	Partial claims below max values can revert	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 80

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L-26	Temporary deposits reset pooled staking age	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 81
L-27	No device-bound passkey policy	Validation	LOW	RESOLVED	p. 82
L-28	P-256 public keys not validated on curve	Validation	LOW	RESOLVED	p. 84
L-29	Direct authorizations cannot be revoked	Signatures	LOW	RESOLVED	p. 86
L-30	Advertised ERC7540 delegation is rejected	Compatibility	LOW	RESOLVED	p. 87
I-01	Unharvested yield wiped by vault loss	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 88
I-02	Redeem liquidity is not reserved	Trust Assumptions	INFORMATIONAL	PENDING	p. 89
I-03	Default admin transfers have no effective delay	Configuration	INFORMATIONAL	PENDING	p. 91
I-04	Address-owner ECDSA accepts high-s	Best Practices	INFORMATIONAL	RESOLVED	p. 93
I-05	Dust redeem burns shares for zero assets	Unexpected Behavior	INFORMATIONAL	RESOLVED	p. 94
I-06	ZeroDev library findings report	Warning	INFORMATIONAL	PENDING	p. 95
I-07	Dead error Nav0racleInvalidConfig	Superfluous Code	INFORMATIONAL	PENDING	p. 96
I-08	Zero activation-loss bricks deposits	Configuration	INFORMATIONAL	RESOLVED	p. 97

REMEDIATION REVIEW 1

H-01	Close redirects manager return	Gaming	HIGH	RESOLVED	p. 98
H-02	One-atom donation lets a depositor seize almost all post-loss recovery	Informational	HIGH	ACKNOWLEDGED	p. 99
H-03	Idle-only full-exit guard lets a dust successor capture deployed pre-settlement surplus	Informational	HIGH	ACKNOWLEDGED	p. 100
H-04	Epoch-close front-run vetoes depleted reset and captures delayed legacy proceeds	Informational	HIGH	PARTIALLY RESOLVED	p. 101
M-01	One wei self-transfer kills orphan capture	Logical Error	MEDIUM	RESOLVED	p. 103
M-02	Revoked auth re-arms at historical generation	Signatures	MEDIUM	ACKNOWLEDGED	p. 104
M-03	A live one-share dust request can indefinitely force whole epochs into recovery	Informational	MEDIUM	PARTIALLY RESOLVED	p. 105
M-04	NAV payouts allow receiver == vault	Validation	MEDIUM	RESOLVED	p. 106
M-05	Lazy redemption settlement grants a one-sided option on post-target strategy returns	Informational	MEDIUM	PARTIALLY RESOLVED	p. 107
M-06	Remaining share atoms let a later cohort steal prior residuals	Informational	MEDIUM	ACKNOWLEDGED	p. 108
M-07	Externally backed pending claims mask full-exit residuals from the new-cohort safeguard	Informational	MEDIUM	PARTIALLY RESOLVED	p. 109
M-08	Shortage-side full-exit gate lets stale redeemers seize an entire successor deposit	Informational	MEDIUM	RESOLVED	p. 110

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
M-09	Ceil-valid floor-priced reports let a full-supply holder indefinitely veto deposits	Informational	MEDIUM	RESOLVED	p. 111
M-10	Bundled deposits replace impaired VaultV2 shares at a transient stale NAV	Informational	MEDIUM	PARTIALLY RESOLVED	p. 112
L-01	Aave exit residual can drift by one atom	DoS	LOW	PARTIALLY RESOLVED	p. 113
L-02	Authoritative liquidity blocks exit	DoS	LOW	RESOLVED	p. 115
L-03	Donation blocks same epoch deposits	DoS	LOW	RESOLVED	p. 117
L-04	Dust redeem blocks funded cohort	DoS	LOW	RESOLVED	p. 118
L-05	Raw-zero orphan classification confiscates yield from accepted but unminted deposit equity	Informational	LOW	RESOLVED	p. 119
L-06	Exact withdrawal can burn the full supply while paying only a partial target	Informational	LOW	PARTIALLY RESOLVED	p. 120
I-01	Operator-executed async deposit claims are logged as controller calls	Informational	INFORMATIONAL	ACKNOWLEDGED	p. 121
I-02	Ceil rounding makes advertised partial withdrawal amounts unexecutable	Informational	INFORMATIONAL	PARTIALLY RESOLVED	p. 122

REMEDIATION REVIEW 2

M-01	Close prices Morpho pin without live NAV	Logical Error	MEDIUM	REPORTED	p. 123
M-02	Pending redeem shares earn yield on idle reserve	Logical Error	MEDIUM	REPORTED	p. 125
L-01	Zero-tolerance live check skips compounding	Rounding	LOW	REPORTED	p. 127
L-02	Pending Morpho burns do not stop intake	Validation	LOW	REPORTED	p. 129
I-01	Redeem claim invalidates prepared report	DoS	INFORMATIONAL	REPORTED	p. 130
I-02	Misleading escrow claim in reserve check	Informational	INFORMATIONAL	REPORTED	p. 131
I-03	Unused imports and errors	Superfluous Code	INFORMATIONAL	REPORTED	p. 132
I-04	minShares check can never fail	Superfluous Code	INFORMATIONAL	REPORTED	p. 133
I-05	Unreachable clamp in _maxWithdraw	Superfluous Code	INFORMATIONAL	REPORTED	p. 134
I-06	claimWithdraw pays assets for 0 shares	Events	INFORMATIONAL	REPORTED	p. 135
I-07	Return retry can land on a new cohort	Trust Assumptions	INFORMATIONAL	REPORTED	p. 136
I-08	Safe owners cannot reuse replayable signatures	Trust Assumptions	INFORMATIONAL	REPORTED	p. 137

CRITICAL C-01

RESOLVED

YS vault depositors permanently stuck

CATEGORY	DoS
LOCATION	AsyncYieldSplitVault.sol:1017-1018
REVIEW	Main Review

DESCRIPTION

`AsyncYieldSplitVault` is an ERC-4626 wrapper: depositors enter via an async request, activate into an immutable `yieldVault`, then exit through `withdraw` / `redeem`. Each exit is capped by `_activeLiquidManagedAssets() = idle + yieldVault.maxWithdraw(this)`, and the OZ base requires `assets <= maxWithdraw` / `shares <= maxRedeem`.

Morpho Vault V2 is a team-confirmed yield vault (i.e. Steakhouse Prime USDC on Base). Every Vault V2 max function returns zero by design, while `deposit` / `withdraw` work normally. Wrapper deposits still succeed (they never read `maxWithdraw`).

Concretely, post-activation:

- activation into the V2 vault succeeds; Glider holds Morpho shares (`previewRedeem` $\setminus > 0$), 0 idle USDC.
- `MorphoV2.maxWithdraw(glider) = maxRedeem(glider) = 0`.
- so Glider `maxWithdraw(alice) = maxRedeem(alice) = 0`.
- `alice.withdraw(...)` / `redeem(...)` revert.
- manager `harvestYield(...)` reverts too (`maxHarvest() = 0`).

The freeze is total and permanent: harvest is gated by the same figure, no function withdraws from the V2 outside the reverting path, the wrapper is immutable, and `recover` only handles zero-output deposit requests. Every depositor's funds are frozen while fully backed.

RECOMMENDATION

C-01 YS vault depositors permanently stuck

CONTINUED

Do not size synchronous exit liquidity from `yieldVault.maxWithdraw()`. ERC-4626 does not guarantee it reflects real withdrawable liquidity, and Morpho Vault V2 returns zero by design while `withdraw`/`redeem` still work. Derive exit capacity from what the wrapper can actually realize:

`yieldVault.convertToAssets(yieldVault.balanceOf(address(this)))` (or `previewRedeem` of held shares) plus idle. Let `_pullAssets` attempt the real `yieldVault.withdraw`/`redeem` and surface a revert only if it genuinely fails.

Also validate `yieldVault` compatibility at deployment by asserting `maxWithdraw` tracks the held position, and add a permissioned path to move `yieldVault` principal into idle so positions stay exitable when the underlying's `maxWithdraw` is unreliable. Vault V2's always-zero max functions must be a supported case, since it is a confirmed target.

PROOF OF CONCEPT <https://gist.github.com/wafflemakr/c4b50370628a6801ffeaec91f688f7d2>

— RESOLUTION

Status: Resolved.

HIGH **H-01**

PENDING

Refresh bypasses NAV freshness gate

CATEGORY MEV
 LOCATION AsyncNavVaultRequests.sol
 REVIEW Main Review

DESCRIPTION

A request made in epoch N must settle at epoch $N + 1$'s price (`_requireTargetEpoch`). The design intends this next price to be unknown when the user commits, so nobody can pick a price they already like. To bind a request to epoch N you set `requestEpoch = N`. Create and increase do this but also call `_requireOracleFresh()`, which makes binding to N impossible once $N + 1$ is about to post: epoch N stops being fresh at `T_N + epochDuration` (`_isFreshTimestamp`), the exact moment $N + 1$ becomes postable (`_recordAttestation`). So by the time the $N + 1$ price exists, no fresh request can target it. The bug: `refreshDepositRequest`, `refreshRedeemRequest`, and the partial retarget in `_decreaseDepositRequest` / `_decreaseRedeemRequest` also set `requestEpoch = currentEpoch()` but skip `_requireOracleFresh()`. An attacker who parks one stale-but-refreshable request can retarget it to N after the $N + 1$ price is known, then settle at that known price.

Timing: this must happen before $N + 1$ is recorded. While `currentEpoch()` is still N , refresh sets `requestEpoch = N` (target $N + 1$); after $N + 1$ is recorded refresh can only reach $N + 2$. In signed-relay mode `pushSignedSnapshot` is callable by anyone with the reporter signature, so the attacker submits the record and all three steps in one transaction.

Example (redeem, `epochDuration = 1h`):

1. At a fresh epoch the attacker calls `requestRedeem(100 shares)` and never claims; the request goes stale but stays refreshable, 100 shares escrowed.
2. Latest epoch is 8 , price 1.00 . NAV spikes and the reporter signs epoch 9 at price 1.50 ; the attacker sees the relayed payload.
3. In one transaction: `refreshRedeemRequest` (binds `requestEpoch = 8`, target 9), `pushSignedSnapshot(9, 1.50, ...)`, `claimRedeem(100)`.
4. The 100 shares settle at 1.50 and pay 150 USDC, a price the attacker already knew.

Impact

The attacker fires only when the known price favors them and skips otherwise, turning settlement at an unknown future NAV into a risk-free pick. The gain comes from remaining shareholders.

H-01 Refresh bypasses NAV freshness gateCONTINUED

— RECOMMENDATION

Do not let a retarget pin a `requestEpoch` whose successor can be posted in the same block. Options:

- Require `_requireOracleFresh()` inside `refresh*` and the `_decrease*` retarget, mirroring the create and increase paths. Because this narrows the recovery use case, pair it with a freshness-independent cancel or withdraw-to-self path so escrow is never trapped.
- Or require settlement to observe that the target report was posted at least one block after the request's last `requestTimestamp`.
- Or only allow a retarget to move `requestEpoch` forward when the latest report is fresh, and otherwise restrict the request to cancel or withdrawal of escrow.

— RESOLUTION

Status: Pending.

HIGH **H-02**

PARTIALLY RESOLVED

JIT claims skim returned yield

CATEGORY	MEV
LOCATION	AsyncYieldSplitVault.sol
REVIEW	Main Review

— DESCRIPTION

`returnYieldAssets` and `returnYieldVaultShares` add returned yield to `accountedShareholderAssets`, splitting it across the current share supply.

The two-epoch deposit delay does not protect that split. A deposit request becomes claimable when `currentEpoch >= requestEpoch + 2`, but it never expires. The controller can leave capital in a matured request, cancel it, or claim it later.

An attacker can pre-position a matured request, wait for a visible `returnYield*` transaction, claim immediately before it, then redeem immediately after it. The request already aged in advance, and redemption has no holding period.

Example:

- Existing holders have 1,000 shares backed by 1,000 assets.
- The attacker has a matured 1,000 asset request.
- A 100 asset return is pending.
- The attacker claims 1,000 shares before the return.
- The return is split across 2,000 shares.
- The attacker redeems for 1,050 assets, so existing holders receive only 50 of the 100 asset return.

Impact

Pre-positioned claims can repeatedly skim returned yield from existing shareholders. The attacker must lock refundable capital while parked, but has no price exposure until claiming.

— RECOMMENDATION

Define the yield recipient set before the return is observable. For example:

- Accrue returned yield through a per-share accumulator that excludes shares minted after the return cutoff.
- Snapshot eligible shares before accepting the returned yield.
- Vest returned yield over time, or require a holding period before newly claimed shares can redeem against returned yield.

H-02 JIT claims skim returned yield

CONTINUED

— **RESOLUTION**

Status: Partially resolved.

HIGH **H-03**

PENDING

Epoch gate ignores NAV report freshness

CATEGORY Frontrunning
LOCATION AsyncNavVaultSettlement.sol:171-190
REVIEW Main Review

DESCRIPTION

Every request action gates on the epoch counter, never on report freshness. `_requireTargetEpoch` allows settlement while `currentEpoch()` equals `requestEpoch + 1`. `_requireEditablePendingEpoch` allows a cancel or decrease before that. Neither checks the target attestation timestamp.

The next report becomes postable exactly when the target goes stale, since both spans equal `epochDuration`. Until it posts, `currentEpoch()` still holds the old target and both helpers pass. A controller reads the next price from the signed attestation or a permissionless `pushSignedSnapshot`. On the claim side a controller settles at the old price. Bob requests to redeem 100 shares in epoch 3, targeting epoch 4. Epoch 4 posts at 1.00, claimable, but he waits. Once epoch 5 is postable he sees it pending at 0.50, then claims at 1.00 before it posts. He takes 100 assets and dodges the loss; deposits are symmetric.

On the edit side the window keeps `cancelDepositRequest` and `cancelRedeemRequest` open. A controller cancels unfavorable requests after learning the next NAV and keeps favorable ones. Either door is a free option at the previous price; passive shareholders absorb the shifted NAV.

RECOMMENDATION

Gate both helpers on report freshness, not only the epoch counter. The vault already exposes `_isFreshTimestamp`; reuse it. In `_requireTargetEpoch`, reject settlement once the target attestation is stale, so a claim is valid only while `block.timestamp < targetTimestamp + epochDuration`. Apply the same cutoff in `_requireEditablePendingEpoch`, so cancel and decrease close at that boundary. That boundary is exactly when the next report can post, which removes the disclosure window. After the cutoff, require settlement, a refresh, or a priced recovery path.

RESOLUTION

Status: Pending.

HIGH **H-04**

RESOLVED

Permissionless reward claims strand rewards

CATEGORY	Frontrunning
LOCATION	ExternalRewardsAsyncYieldSplitVault.sol, MorphoRewardsAsyncYieldSplitVault.sol
REVIEW	Main Review

— DESCRIPTION

The reward variants forward external incentives by measuring the vault's reward token balance immediately before and after the vault's own protocol claim, and sending only that delta to `harvestRecipient`:

```

`plain text
uint256 balanceAfter = rewardToken.balanceOf(address(this));
rewards = balanceAfter > balanceBefore ? balanceAfter - balanceBefore : 0;
...
if (rewards == 0) revert ExternalRewardAmountZero(address(rewardToken));
rewardToken.safeTransfer(harvestRecipient, rewards);
`

```

Morpho's URD claim is permissionless (note: Merkl is not) and can be called by anyone on behalf of any account / user, sending the cumulative rewards directly to the vault, outside `harvestMorphoUrdReward`. After that:

1. The reward tokens sit in the vault, but are excluded from all vault accounting and from `harvestYield` (which only moves asset surplus).
2. The manager's `harvestMorphoUrdReward` now claims a delta of zero and reverts with `ExternalRewardAmountZero`. `harvestRecipient` never executes.`
3. The vault exposes no token rescue or sweep. The directly-claimed tokens are therefore permanently locked.

The attack is repeatable: a griever frontruns every cumulative increase by claiming directly at the distributor, so the manager's harvest never succeeds and the entire reward stream is locked over the life of the campaign.

— RECOMMENDATION

Decouple forwarding from the just-claimed delta so that already-arrived rewards are still recoverable:

- Forward the vault's full reward-token balance to `harvestRecipient` instead of only the claim delta. This is safe because reward tokens are excluded from accounting, but it must explicitly reject `rewardToken == asset()` and `rewardToken == address(yieldVault)` so it

H-04 Permissionless reward claims strand rewards

CONTINUED

can never drain shareholder backing.

- Or add a `harvestManager` / `harvestRecipient` gated rescue for tokens that are neither `asset()` nor the `yieldVault` share token, so externally-claimed rewards can always be swept to the recipient.

— RESOLUTION

Status: Resolved.

HIGH **H-05**

RESOLVED

Matured claims capture loss recovery

CATEGORY	Gaming
LOCATION	AsyncYieldSplitVault.sol
REVIEW	Main Review

— DESCRIPTION

Eligible deposit requests remain idle, refundable, and excluded from share price until the controller claims them. The claim path does not check collateralization, so a controller can pre-position a matured request, wait until the underlying `yieldVault` has impaired active shareholders, and then claim at the depressed `totalAssets() / totalSupply()` price.

The request's capital was not active when the loss occurred. It also remained cancellable through `cancelDepositRequest`, so the controller had no downside exposure while waiting. Once claimed, the new shares participate in any later organic/external recovery of the underlying vault up to `accountedShareholderAssets`, reducing the recovery that should have gone to the shareholders who actually suffered the loss.

Example from the PoC:

- Alice has 1,000 shares backed by 1,000 assets.
- Bob parks a 500 asset request until it is claimable.
- The underlying vault loses 500 assets, so Alice's shares are worth 500.
- Bob claims after the loss and receives 1,000 shares, worth 500 at the impaired price.
- The underlying vault later recovers the 500 assets (Protocol external donation)
- Without Bob's claim, Alice would recover to 1,000. With Bob's claim, Alice and Bob each hold 1,000 of the 2,000 shares, so Bob can redeem 750 assets after depositing only 500.

Impact

A pre-positioned request can siphon a large share of principal recovery from existing shareholders after an underlying-vault impairment. The attacker only needs to lock refundable idle capital in advance and can avoid claiming unless a profitable recovery window appears.

— RECOMMENDATION

Reject claims when `isCollateralized = false`. Alternatively, consider tracking a loss epoch or recovery accumulator so newly claimed shares enter at the impaired value but do not receive pre-claim recovery.

H-05 Matured claims capture loss recovery

CONTINUED

— **RESOLUTION**

Status: Resolved.

HIGH **H-06**

RESOLVED

Owner storage layout blocks all user operations

CATEGORY	DoS
LOCATION	MultiOwnerDeterministicValidator.sol
REVIEW	Main Review

DESCRIPTION

`validateUserOp` runs inside the ERC-4337 validation phase, so every storage slot it reads must satisfy the ERC-7562 storage rules that bundlers enforce during mempool simulation. The relevant rule is:

>
 \[STO-021\] Access to associated storage of the account in an external (non-entity) contract is allowed if the account already exists.

with the spec definition:

>
 A storage slot is associated with address `A` if the slot value is `A`, or the slot was calculated as `keccak(A || x) + n`, where `x` is a `bytes32` and `n` is in `0..128`.

The owner state is keyed with the account as the outer mapping key and the owner as the inner key:

```

`plain text
mapping(address => OwnerStorage) internal ownerStorage;
struct OwnerStorage {
    uint8 ownerCount;
    mapping(address => AddressOwnerConfig) addressOwners;
    mapping(bytes32 => P256OwnerConfig) p256Owners;
    mapping(bytes32 => bool) p256OwnerExists;
    ...
}
`

```

`ownerStorage[account].ownerCount` resolves to `keccak(account || base)`, which is account-associated, so the `_isInitialized` read is fine. But every signature check then reads an owner-keyed slot:

```

`plain text
// address owner
MultiOwnerStructs.AddressOwnerConfig memory config =

```

H-06 Owner storage layout blocks all user operations

CONTINUED

```
ownerStorage[account].addressOwners[owner];
// raw P-256 / WebAuthn
if (!accountStorage.p256OwnerExists[payload.ownerId]) { ... }
MultiOwnerStructs.P256OwnerConfig memory config = accountStorage.p256Owners[payload.ownerId];
``
```

`addressOwners[owner]` resolves to `keccak(owner || (keccak(account || base) + 1))`, which is of the form `keccak(owner || x)`, so it is associated with `owner`, not the account. The P-256 and WebAuthn reads are keyed by a `bytes32 ownerId`, so the slot `keccak(ownerId || x)` is associated with no address at all. None of these are account-associated, so they fall outside ST0-021. Because every successful verification must read one of these owner-keyed slots, a canonical-mempool bundler that traces validation rejects every user operation this validator would approve. This affects all schemes (address, raw P-256, WebAuthn) and both direct and multichain modes.

Kernel's own multi-key signer avoids this by placing the account as the innermost key so the final slot stays account-associated: `mapping(bytes32 id => mapping(address wallet => address)) signer`, where `signer[id][wallet]` resolves to `keccak(wallet || x)`. The Glider validator inverts the nesting, which moves the final slot off the account.

RECOMMENDATION

Reorder the owner storage so that every slot read during validation is account-associated, by making the account the innermost mapping key. For example use `mapping(address owner => mapping(address account => AddressOwnerConfig))` and `mapping(bytes32 ownerId => mapping(address account => P256OwnerConfig))`, so each validation read resolves to `keccak(account || x)`.

RESOLUTION

Status: Resolved.

MEDIUM **M-01**

PENDING

Signed NAV snapshot has no recency bound

CATEGORY Signatures
LOCATION AsyncNavVaultCore.sol:350-356
REVIEW Main Review

— DESCRIPTION

`attestationDigest` binds only `(epoch, priceWad, totalNav, chainid, address)` — no deadline or observation timestamp. Anyone holding a reporter-signed attestation (a relayer or MEV actor) can submit it via `pushSignedSnapshot` for the next sequential epoch at an arbitrary later time. The contract records it with `timestamp = block.timestamp` at `AsyncNavVaultCore.sol:430-434`, treating a price signed long ago as the fresh latest NAV.

That stale price then gates new deposit/redeem requests and prices settlement for its epoch. The team documents same-block reporting as the operational assumption, so this is a trust/operational gap rather than an open exploit, but the signature itself carries no freshness binding.

— RECOMMENDATION

Include an observation timestamp or signature deadline in the attestation digest and reject attestations older than a bounded window in `pushSignedSnapshot`.

— RESOLUTION

Status: Pending.

MEDIUM **M-02**

RESOLVED

Loss freezes deposits, exits stay open

CATEGORY Logical Error
LOCATION AsyncYieldSplitVault.sol:456-470
REVIEW Main Review

DESCRIPTION

When the `yieldVault` loses value so that `_activeManagedAssets()` falls below `accountedShareholderAssets`, `isCollateralized()` returns false. The vault then drops into a frozen, redeem-only mode. `requestDeposit`, `closeEpoch`, and every harvest function revert in this state (see `AsyncYieldSplitVault.sol:459`). So `currentEpoch` cannot advance, and a not-yet-eligible deposit never reaches its `requestEpoch+2` threshold. Meanwhile `withdraw`, `redeem`, `cancelDepositRequest`, and the claim of an already-eligible request all stay open. The vault has no internal path back to health. Redemptions burn `accounted` and `managed` in the same proportion, so they preserve the shortfall ratio. `returnYieldAssets` raises `accounted` by what it adds, so it does not close the gap either. Collateralization is restored only externally: the `yieldVault` price recovers, someone donates raw assets, or every shareholder exits and the vault resets empty. If the `yieldVault` is permanently impaired, the freeze is permanent and the vault sits in indefinite runoff. There is also an exit race. Because exits stay open while entry is frozen, a deepening loss lets an early `redeem` lock in the shallower haircut while a slower shareholder absorbs the rest. Pending depositors never lose principal: `cancelDepositRequest` is ungated and refunds the full request.

RECOMMENDATION

When the vault is undercollateralized, record a loss epoch and let `deposit` and `mint` claims mint at the impaired NAV while barring newly-minted shares from sharing any recovery that accrued before they were claimed. This stops a pre-positioned claimer from siphoning recovery owed to the holders who absorbed the loss, and keeps deposits and exits live throughout the shortfall.

RESOLUTION

Status: Resolved.

MEDIUM **M-03**

PENDING

NAV disclosure cancellation option

CATEGORY MEV

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultRequests.sol:133-171`

REVIEW Main Review

DESCRIPTION

`AsyncNavVaultRequests.cancelDepositRequest` and `cancelRedeemRequest` remain available until the target epoch is posted on-chain. A controller who sees the reporter's signed N+1 attestation before it is recorded can cancel while `_requireEditablePendingEpoch` still treats the request as editable at `AsyncNavVaultRequests.sol:133`.

This gives queued users a free option. They can cancel unfavorable executions after learning the next NAV, but keep requests that settle at a favorable price. Other shareholders absorb the adverse selection.

RECOMMENDATION

Close cancellation and decrease requests at a deterministic cutoff before the target epoch's execution NAV can be disclosed. After that cutoff, require settlement, refresh, or a priced recovery path.

RESOLUTION

Status: Pending.

Guardian: Main review M-03 and M-15 were closed by making deposit cancel expiry only, so a request could not be pulled once it was bound to the next NAV. The v2 request rewrite removed that gate.

`cancelDepositRequest` now refunds in full until `pushSnapshot` accepts the batch:

```
``plain text
if (request.pendingAssets == 0 || request.cancelled ||
depositBatches[request.requestEpoch].accepted) {
revert AsyncNavVaultRequestNotPending(requestId);
}
`[
```

A controller who can see a public mempool report, or infer the next price from public holdings, can still drop an unfavorable request and keep a favorable one. Remaining shareholders absorb that selection. `SECURITY.md` already states that cancel and increase before a

M-03 NAV disclosure cancellation option

CONTINUED

visible report can exploit impending price, and that private submission is an operational control rather than a contract guarantee. This is that documented residual, not a new class of bug. Do not count it separately from M-03. Late favorable intake (M-15) was already left open under the prior F* close.

****Recommendation****

Keep private report submission as the residual control already documented in `SECURITY.md`. Restore an onchain cancel cutoff only if that operational control is not actually used.

MEDIUM **M-04**

PENDING

Delayed deposit claims capture NAV gains

CATEGORY Unexpected Behavior

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol`

REVIEW Main Review

DESCRIPTION

`_settleDepositRequest` fixes a deposit's claimable share amount at the target epoch price, but it does not mint those shares or include them in `totalSupply`. It only stores `claimableAssets` and `claimableShares` in request accounting. Later, `claimDeposit` and `claimMint` mint those old fixed shares when the controller finally claims.

This creates a mismatch between the time the share amount is fixed and the time the shares join the active supply. The docs tell the reporter to exclude queued deposit capital that has not minted shares from NAV. Therefore, while a settled deposit remains unclaimed, later NAV reports can assign gains or losses only to the already minted share supply. The delayed depositor still keeps the right to mint the old fixed share amount afterward.

For example, the active vault has 100 shares backed by 100 USDC. Bob's 100 USDC deposit settles at 1.0 PPS, so the vault records 100 claimable shares for Bob, but those shares are not minted. Bob waits. The active strategy then gains 100 USDC, so the next report prices the 100 existing shares at 2.0 PPS. Bob now claims and receives 100 shares for his old 100 USDC claim. After Bob's deposit assets join active accounting, the vault has 300 USDC backing 200 shares, so the fair PPS is 1.5. Bob's delayed claim is worth 150 USDC even though his queued capital did not participate in the gain while it was excluded from supply. The extra 50 USDC comes from dilution of the active shareholders who were exposed during the gain.

It can move meaningful value when a large settled deposit is left unclaimed across a NAV increase, but the old instant-exit realization route no longer exists in the latest code. The depositor must already have a settled claim and must wait through favorable NAV movement before claiming.

RECOMMENDATION

Use one consistent accounting point for settled deposit shares.

The safest fix is to mint settled deposit shares into vault escrow during `_settleDepositRequest`, include those shares in `totalSupply` immediately, and make later claims transfer the escrowed shares to the receiver instead of minting new supply. The NAV reporter should then include the settled deposit capital from the same point onward.

M-04 Delayed deposit claims capture NAV gains

CONTINUED

If the intended design is that deposit capital only joins active supply at claim time, do not preserve a stale target-epoch share amount indefinitely. Reprice long-lived claims at claim time, require same-epoch claiming, or add an expiry/refund rule for settled but unclaimed deposits.

— RESOLUTION

Status: Pending.

MEDIUM **M-05**

ACKNOWLEDGED

Root reactivation ignores replacement owners

CATEGORY Unexpected Behavior

LOCATION `src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol`

REVIEW Main Review

DESCRIPTION

`MultiOwnerDeterministicValidator.onInstall` initializes the owner set only when the account has no owners stored. `onUninstall` is the only lifecycle function that clears those owners. This does not match Kernel root-validator reactivation. `Kernel.changeRootValidator` sets the new root validator, but it calls `_installValidation` only when the target validation id has no stored validation config. If an account previously used `MultiOwnerDeterministicValidator`, then switched to another root validator without uninstalling it, the multi-owner validation id remains installed. Switching back to the same validator instance later does not call `onInstall`, so any replacement owner data passed to `changeRootValidator` is ignored.

Consequently, an old owner set can become active again when the account reactivates the same validator. If the account switched away from this validator because an owner was compromised or deprecated, that stale owner regains root validation authority when the validator is selected again. The intended replacement owner is not added, so the account can also become unusable for the operator who expected the new owner set to apply.

RECOMMENDATION

Choose one reactivation rule and enforce it consistently: either switching away must uninstall and clear this validator's owner storage or switching back must explicitly replace the stored owners.

If root switches away from this validator are meant to deactivate it, the account integration should uninstall the old validation id and trigger `onUninstall` before switching away. If reactivation is meant to support replacement owners, the account should call `replaceAllOwners` through `kernel.execute` while the validator is still initialized, or the validator/integration should expose a lifecycle-gated reinstall operation that replaces owners when the account intentionally reactivates the validator.

Do not rely on `validatorData` in `changeRootValidator` to update owners for a validation id that Kernel already considers installed.

M-05 Root reactivation ignores replacement owners

CONTINUED

— **RESOLUTION**

Status: Acknowledged.

MEDIUM **M-06**

PENDING

Target-epoch views double count claims

CATEGORY	Unexpected Behavior
LOCATION	AsyncNavVaultSettlement.sol
REVIEW	Main Review

— DESCRIPTION

At the target epoch, the status views can report the same request as both pending and claimable.

`pendingDepositRequest()` returns `request.pendingAssets` directly while `claimableDepositRequest()` can also return `_pendingDepositClaimable(request)` from that same pending request. The redemption views mirror this behavior: `pendingRedeemRequest()` returns `request.pendingShares`, while `claimableRedeemRequest()` can return `_pendingRedeemClaimable(request)` before settlement has moved storage out of pending state.

ERC-7540 separates pending and claimable request balances. Pending views must not include amounts that are already claimable and claimable views must not include amounts that remain pending. This implementation intentionally exposes lazy target-epoch claimability, but the pending views do not become mutually exclusive with those lazy claimable views.

Consequently, ERC-7540 adapters, routers, accounting systems and frontends can double count a target-epoch request. A deposit request can appear as both pending assets and claimable assets. A liquid redemption request can appear as both pending shares and claimable shares. This is an integration and accounting correctness issue rather than a direct vault asset drain.

— RECOMMENDATION

Make the status views mutually exclusive at the target epoch. If a pending deposit request is lazily claimable, `pendingDepositRequest()` should return zero for that request and `claimableDepositRequest()` should return the claimable assets. If a pending redemption request is lazily claimable and liquid, `pendingRedeemRequest()` should return zero and `claimableRedeemRequest()` should return the claimable shares.

Alternatively, require an explicit settlement step that moves request storage from pending to claimable before any claimable status view reports a nonzero value.

— RESOLUTION

Status: Pending.

MEDIUM **M-07**

RESOLVED

Aave reward variant incompatible with Earn Vault

CATEGORY	Compatibility
LOCATION	AaveRewardsAsyncYieldSplitVault.sol
REVIEW	Main Review

— DESCRIPTION

`AaveRewardsAsyncYieldSplitVault.harvestAaveReward` claims only by calling the Aave `RewardsController.claimRewardsToSelf`. That controller accrues liquidity mining incentives per `user` based on the `user`'s scaled balance of an incentivized asset (an `aToken` or debt token), and `claimRewardsToSelf` pays out only what `msg.sender` accrued.

The team confirmed the intended `yieldVault` is the Aave Earn Vault (`ATokenVault`), an ERC-4626 wrapper that itself supplies to Aave and holds the `aTokens`. The async vault holds only Earn Vault shares, which are not an Aave incentivized asset and carry no scaled balance in the `RewardsController`. So when `harvestAaveReward` calls `claimRewardsToSelf` with `msg.sender = address(this)`, the controller sees a zero scaled balance for the async vault, accrues and transfers 0, the forwarded balance delta is 0, and the call reverts with `ExternalRewardAmountZero` (or the `minRewards` guard). The path can never succeed for this deployment.

The incentives the controller would pay accrue to the Earn Vault (the actual `aToken` holder), and the Earn Vault only lets its `owner` claim them via `claimRewards(address to) onlyOwner`, with no per-depositor distribution. The async vault is therefore never entitled to those tokens through any on-chain path.

Example:

- The async vault holds Earn Vault shares as its `yieldVault` position; the Earn Vault holds `aUSDC`.
- An Aave campaign accrues 100 reward tokens to the Earn Vault (the `aUSDC` holder).
- The manager calls `harvestAaveReward([aUSDC], rewardToken, type(uint256).max, 1)`.
- The controller sees the async vault holds no `aUSDC` scaled balance, transfers 0, and `harvestAaveReward` reverts.

— RECOMMENDATION

For the Earn Vault deployment, do not use `AaveRewardsAsyncYieldSplitVault`; deploy the base vault (or a non-Aave variant), since supply interest is already captured through the yield-split path. If the Earn Vault's liquidity-mining incentives must reach shareholders,

M-07 Aave reward variant incompatible with Earn Vault

CONTINUED

they can only be obtained out-of-band by the Earn Vault `owner` via `claimRewards(to) onlyOwner` and then routed back through `returnYieldAssets` / `returnYieldVaultShares`; the async vault cannot self-claim them.

If a future deployment instead uses raw aTokens or a holder-distributing wrapper such as `StaticATokenLM`, add a claim path that calls that wrapper's holder-level reward functions as `address(this)` rather than the Aave `RewardsController`, and document which `yieldVault` types each reward variant supports.

— RESOLUTION

Status: Resolved.

MEDIUM **M-08**

RESOLVED

Multichain validation uses banned opcode

CATEGORY Compatibility
 LOCATION MultiOwnerDeterministicValidator.sol
 REVIEW Main Review

DESCRIPTION

Multichain authorizations carry a signed `expiry`. The validator enforces it inside `_validateMultichainAuthorization` by reading `block.timestamp` and reverting once the deadline has passed:

```
``solidity
if (authorization.expiry < block.timestamp) {
  revert ExpiredAuthorization();
}
`
```

However, before a user operation enters the mempool, a bundler simulates the execution to enforce the ERC-7562 rules. Rule OP-011 bans opcode `TIMESTAMP`: a value that can change between simulation and inclusion would let an operation pass simulation and then revert on-chain, which is a DoS vector against the bundler. The eth-infinitism reference bundler enforces this [rule](/3688bda5828c8010822ef27715f9988e?pvs=25).

As a result, multichain user operations are incompatible with standard ERC-7562-compliant public bundlers, because validation reads a banned opcode and is rejected at simulation. ERC-4337 already provides the correct mechanism for time-bounding an operation. Validation should return `validAfter` and `validUntil` in the packed validation data. The EntryPoint then enforces that window at inclusion. However, currently, on success `validateUserOp` returns only `SIG_VALIDATION_SUCCESS_UINT (0)`. It never packs the signed `expiry` into the `validUntil` field of the returned data.

Illustration:

```
``plain text
CURRENT (reads block.timestamp in validation)

Bundler EntryPoint.simulateValidation Validator
| simulate + trace ||
|----->| validateUserOp ----->|
| | | _validateMultichainAuthorization
| | | reads TIMESTAMP
```

M-08 Multichain validation uses banned opcode

CONTINUED

```
(0x42)
| trace contains TIMESTAMP | |
|<-----| |
| OP-011 violation |
| REJECT before mempool | (never reaches on-chain handleOps)
X
```

CORRECT (return validUntil, let EntryPoint enforce)

```
Bundler EntryPoint Validator
| simulate + trace | |
|----->| validateUserOp ----->|
| | | verify proof + owner sig
| | validationData(validUntil=expiry, validAfter=0)
| no banned opcode |<-----|
|<-----|
| ACCEPT into mempool |
| |
| handleOps (on-chain) |
|----->| block.timestamp > validUntil ?
| | yes -> FailedOp "AA22 expired or not due"
| | no -> execute
...

```

RECOMMENDATION

Stop reading `block.timestamp` in the user operation path and delegate the time bound to the EntryPoint's built-in check. For multichain signatures, verify the proof and owner signature, then return packed validation data with `validUntil` set to the signed `expiry` and `validAfter` set to `0`.

RESOLUTION

Status: Resolved.

Guardian: `_verifySignatureEnvelope` takes a `checkExpiry` flag. `validateUserOp` passes `false`, and the comment at `MultiOwnerDeterministicValidator.sol:1405-1406` states the opcode never executes in the public `validateUserOp` trace. That holds for the direct entry point only.

Kernel's `VALIDATION_MODE_ENABLE` is the one way to install a validator, hook or selector through a UserOp. Its `_checkEnableSig` calls `isValidSignatureWithSender` from inside

M-08 Multichain validation uses banned opcode

CONTINUED

`Kernel.validateUserOp`. That entry point routes through the overload hard-coding `checkExpiry = true` at `MultiOwnerDeterministicValidator.sol:1131`, reaching `authorization.expiry < block.timestamp` at `:1408`.

The comparison sits only in `_validateMultichainAuthorization`. The trigger is an enable-mode `UserOp` whose `enableSig` is a chain-bound or replayable multichain envelope. Direct-mode envelopes never reach it.

Holding one `UserOp`'s bytes fixed, `Kernel.validateUserOp` installs the enabled validator below the signed expiry and reverts `EnableNotApproved()` above it. Only block time differs between the attempts, so validation depends on `TIMESTAMP`.

`ERC7562_RELEASE_GATE.md:57` certifies the opposite, requiring tests to show public `validateUserOp` does not execute `TIMESTAMP` for multichain expiry. Its target-chain trace list carries no enable-mode case, so the gate cannot observe this path.

Conformant bundlers reject such a `UserOp` and can reputation-throttle the sender, making multichain authorization unusable for the enable flow. Owners keep a workaround: sign the enable with a direct-mode envelope.

****Recommendation****

Keep `checkExpiry` enabled on the ERC-1271 surface. Change the expiry comparison to `if (checkExpiry && authorization.expiry < type(uint48).max && authorization.expiry < block.timestamp)`. Short-circuit evaluation means a perpetual authorization never reads `TIMESTAMP`, so enable-mode validation stops depending on block time, while every finite expiry stays enforced. Document that enable-mode envelopes must be perpetual and are revoked via `authorizationGeneration`.

MEDIUM **M-09**

ACKNOWLEDGED

ERC1271 owners violate validation rules

CATEGORY	DoS
LOCATION	MultiOwnerDeterministicValidator.sol
REVIEW	Main Review

— DESCRIPTION

`validateUserOp` validates address-owner signatures through `_verifyAddressOwner`. When the owner is configured for ERC-1271, the validator falls through to `_verifyAddressOwner1271`, which performs an external `staticcall` to the owner contract:

```
``plain text
```

```
owner.staticcall(abi.encodeWithSelector(IERC1271.isValidSignature.selector, hashToVerify, ownerSig));
```

The validator explicitly supports Safe-style ERC-1271 owners. However, Safe ERC-1271 validation reads Safe storage during `checkSignatures`, including `threshold` and `owners[currentOwner]`. That storage belongs to the Safe owner contract, not to the Kernel account whose user operation is being validated. Under ERC-7562 canonical mempool [rules](https://eco.com/support/en/articles/11564592-what-is-erc-7562-aa-validation-rules), an unstaked account validation frame may ONLY access the account's own storage and account-associated external storage. As a result, a Glider user operation that relies on a Safe or similar storage reading ERC-1271 owner will be rejected by standard public ERC-4337 bundlers during validation.

— RECOMMENDATION

Do not rely on arbitrary ERC-1271 owner calls inside the public `validateUserOp` path. Either keep the user operation signer state in Glider or other account associated storage, use a staked validation entity or explicitly documented alt mempool, or limit Safe/ERC-1271 support to `isValidSignature` message checks where ERC-7562 validation rules do not apply.

If Safe owners remain supported for user operations, document the required staking or private bundler assumptions and reject unsupported configurations during installation.

— RESOLUTION

Status: Acknowledged.

MEDIUM **M-10**

RESOLVED

Zero-supply strands unharvestable yield

CATEGORY Unexpected Behavior
LOCATION AsyncYieldSplitVault.sol:709-727
REVIEW Main Review

DESCRIPTION

The vault accrues yield as unallocated surplus: `_shareholderAssets()` caps shareholder value at `accountedShareholderAssets`, which only rises when the manager calls `returnYieldAssets`. Holders cannot self-credit it, as the yield-return functions are manager-only. If every shareholder exits while surplus is unreturned, `totalSupply()` hits zero with real yield still held as yield-vault shares. Then all four value-moving paths revert `HarvestRequiresSupply` (`AsyncYieldSplitVault.sol:712`), so the yield can neither be harvested nor returned.

Recovering it needs a full async deposit cycle. A new request, two `closeEpoch` calls (a request is claimable only at `requestEpoch + 2`), then a claim restores supply. Each epoch lasts `epochDuration`, commonly one day or more, so the yield stays locked for two full epochs (about two days at a one-day epoch).

RECOMMENDATION

Allow surplus to be moved when `totalSupply()` is zero, instead of reverting `HarvestRequiresSupply` on every value path. For example, let `harvestAllYield` and `harvestYield` run at zero supply so the harvest recipient can sweep orphaned surplus immediately. Keep the existing guard on the shareholder-return functions.

Alternatively, add a permissioned recovery function that the harvest recipient can call to pull stranded yield-vault shares when no shares are outstanding. At minimum, document this zero-supply behaviour in the residual-risks section. Operators should then return or harvest pending yield before a full exit, and avoid abandoning a vault that still holds surplus.

RESOLUTION

Status: Resolved.

MEDIUM **M-11**

RESOLVED

Fluid extra rewards unclaimable

CATEGORY	Rewards
LOCATION	ExternalRewardsAsyncYieldSplitVault.sol:15-132
REVIEW	Main Review

DESCRIPTION

`ExternalRewardsAsyncYieldSplitVault` claims external incentives exclusively through a single hardcoded Merkl distributor interface, `claim(address[],address[],uint256[],bytes32[][])` (lines 21-26), invoked by `harvestMerklReward` (lines 107-132). There is no generic call executor and no arbitrary-token sweep path. On a Fluid-based deployment, Fluid distributes its optional extra incentives through its own native Merkle distributor, whose claim signature differs from Merkl's. So `harvestMerklReward` cannot call it and no fallback exists to retrieve those rewards.

The consequence is scoped to Fluid's optional extra Merkle incentives. Fluid's native lending yield is already reflected in the underlying ERC4626 share price and `totalAssets`, so vault principal and base yield are unaffected.

Fluid is a confirmed deployment target, with named USDC and ETH vaults on Base and Mainnet. When a Fluid campaign is active, the rewards it allocates to the vault belong to depositors. They can never be claimed or swept, so that value is permanently stranded.

RECOMMENDATION

Add a `FluidRewardsAsyncYieldSplitVault` variant before deploying against any Fluid 4626 vault. It should hardcode Fluid's distributor ABI: `claim(recipient, cumulativeAmount, positionType, positionId, cycle, proof, metadata)`. This follows the same per-protocol pattern as the Aave and Morpho variants and respects the documented choice to avoid a generic call executor. Until such a variant exists, do not rely on these vaults capturing Fluid's extra incentives.

RESOLUTION

Status: Resolved.

MEDIUM **M-12**

PENDING

Cross-chain ERC-1271 owner takeover

CATEGORY	Access Control
LOCATION	MultiOwnerDeterministicValidator.sol:1034-1082
REVIEW	Main Review

— DESCRIPTION

An address owner can be configured (via `setAddressOwnerFlags`) with `allow1271Multichain`, letting a contract owner approve replayable multichain authorizations through ERC-1271. The replayable digest built in `hashReplayableMultichainAuthorization` omits `block.chainid`, so one signature is valid verbatim on every supported chain.

The 1271 owner is keyed only by its bare address in `_verifyAddressOwner1271`, with no codehash or chain binding. The same address can host different code on another chain. An attacker who controls that address on a second chain can deploy an ERC-1271 contract that returns the magic value for an arbitrary replayable authorization, such as one calling `replaceAllOwners`. That signature replays back to the victim chain and is accepted, granting full account takeover. The `batchNonce` field is never consumed on-chain, so it adds no replay protection.

— RECOMMENDATION

Bind ERC-1271 multichain owners to per-chain code identity, not a bare address. Either disallow the replayable (chain-agnostic) 1271 path entirely, or store and check the owner contract's codehash so divergent code on another chain cannot satisfy the owner. Also include `block.chainid` (or the namespace) inside the replayable authorization digest, and consume `batchNonce` in a per-account mapping so each multichain envelope is single-use.

— RESOLUTION

Status: Pending.

MEDIUM **M-13**

RESOLVED

WebAuthn crossOrigin not validated

CATEGORY	Signatures
LOCATION	StrictWebAuthn.sol:42-117
REVIEW	Main Review

DESCRIPTION

`verifySignature` authenticates a P-256 WebAuthn assertion by confirming the `clientDataJSON` contains the expected challenge and that the `origin` value matches the configured `originHash`. The origin check is done by `_extractOriginHash` (`StrictWebAuthn.sol:90`), which only reads the quoted `origin` string and ignores the `crossOrigin` and `topOrigin` fields that WebAuthn includes in `clientDataJSON`. As a result a `clientDataJSON` carrying the allowed `origin` together with `"crossOrigin":true` (an assertion produced inside an attacker-controlled iframe whose top-level frame is the attacker's site) passes validation and is treated as first-party authentication. If the relying-party origin is framable, an attacker can embed it, trick the user into completing a passkey ceremony, and forge a signature the validator accepts as legitimate.

RECOMMENDATION

Parse and enforce the `crossOrigin` field: reject any `clientDataJSON` where `crossOrigin` is true (unless cross-origin use is an explicit, separately-configured policy), and when present validate `topOrigin` against an allowlist. Add a configurable flag on the WebAuthn policy so accounts that do not intend cross-origin assertions reject them by default.

RESOLUTION

Status: Resolved.

MEDIUM **M-14**

RESOLVED

P-256 verifier choice is not signed

CATEGORY	Unexpected Behavior
LOCATION	src/glider-kernel-validator/lib/GliderRootValidatorDigests.sol
REVIEW	Main Review

— DESCRIPTION

The validator lets each P-256 signature payload choose which verifier will check the signature. Raw P-256 and WebAuthn payloads both include `usePrecompiled`; when it is true, `StrictWebAuthn` calls the RIP-7212 precompile at `0x100` and when it is false, it calls the Daimo verifier.

That selector is not part of the signed authorization. `hashDirectAuthorization`, `hashMultichainAuthorization` and `hashReplayableMultichainAuthorization` bind the validator address, account, owner id, operation or Merkle root, replay domain, batch nonce and expiry. They do not bind the verifier target. The WebAuthn challenge is derived from the same digest, so it also does not commit to the selected verifier.

As a result, a relay or caller can take a valid P-256 authorization and change only `usePrecompiled`. The owner still signed the same digest and the same WebAuthn challenge, but the contract now routes the signature to a different verifier than the signer or SDK may have intended.

— RECOMMENDATION

Do not let unsigned calldata choose the P-256 verifier. Bind the verifier target or verifier policy into the signed P-256 authorization or store the allowed verifier target per account or owner and ignore the payload selector during validation.

If both verifier targets must remain supported, the SDK should display and sign the selected target explicitly. Keep deployment-time verifier codehash checks, but also add runtime regression tests proving that a signature cannot be redirected from one verifier target to another without changing the signed digest.

— RESOLUTION

Status: Resolved.

MEDIUM **M-15**

PENDING

NAV requests gameable pre-report

CATEGORY Gaming
LOCATION AsyncNavVaultRequests.sol:20-146
REVIEW Main Review

DESCRIPTION

A request created in epoch `N` is forward-priced. It settles at the next posted NAV, `priceOf(N+1)`, not at a price visible when committing. Forward pricing only protects shareholders if that next NAV is unknowable while the request can still be withdrawn. It is not. The asset manager holds the portfolio on-chain, and the planned holdings are Ondo Global Markets tokens with public market prices. Any user can mark the visible positions to market and predict the direction of report `N+1` before it lands.

Knowing the next NAV before it posts turns a pending request into a one-sided bet. `requestDeposit` and `requestRedeem` are the ERC-7540 intake points, and a pending request can be withdrawn for free before epoch `N+1` posts (`AsyncNavVaultRequests.sol:433`). The user keeps the request when the inferred report is favorable and backs out at no cost when it is not.

Only the favorable side of each move ever settles. Gamers capture the upside of report `N+1` and shed the downside onto shareholders who hold across it, diluting honest holders over time. The vault never asserts the next NAV is unknowable, and the design docs do not acknowledge this gap.

RECOMMENDATION

Make withdrawal of a pending request cost-bearing or time-boxed, so a committed request stops being a free option on the next NAV. One option is a small anti-dilution fee on cancel and decrease. Another is to freeze edits once intake for epoch `N+1` closes, i.e. a request cutoff before that report posts. A third is swing pricing or a dilution levy at settlement, so completed requests bear the cost they impose on honest shareholders.

Because the holdings are public on-chain, the next NAV cannot be made fully unknowable. Document that predictable portfolios such as Ondo GM weaken the async pricing model. Have the reporter post each NAV promptly after a material rebalance, so the inference window stays short.

M-15 NAV requests gameable pre-report

CONTINUED

— RESOLUTION

Status: Pending.

MEDIUM **M-16**

RESOLVED

Activation-loss tolerance DoSes deposits

CATEGORY	DoS
LOCATION	AsyncYieldSplitVault.sol:899-908
REVIEW	Main Review

DESCRIPTION

AsyncYieldSplitVault is an ERC-7540 async-deposit wrapper over an ERC-4626 yield vault. A depositor `requestDeposit`s, then after a two-epoch wait claims with `deposit / mint`. The claim is rejected unless the deposit-then-redeem rounding loss is within the immutable `maxDepositActivationLossAssets` tolerance, in which case `deposit / mint` revert.

That loss is deterministic, $\text{loss} = \text{ceil}((\text{assets} * S \bmod A) / S)$ with $S = \text{totalSupply}$ and $A = \text{totalAssets}$, bounded by `ceil(sharePrice)`. When the tolerance is below that bound, a deterministic fraction $1 - 1 / \text{sharePrice}$ of deposit amounts is rejected. The documented Base tolerance of 1 is below the bound of 2 whenever the share price exceeds 1.

The bound is `ceil(totalAssets / (totalSupply + 10decimalsOffset))`, so it follows the decimals offset. Only Morpho USDC is safe (12-decimal offset, bound 1); Morpho WETH, Fluid USDC, and Fluid WETH each have bound 2 and reject roughly 0.85%, 10.5%, and 2.57% of deposits at tolerance 1.

Funds are recoverable via `recoverZeroOutputDepositRequest`, but the tolerance is immutable, so a too-low value needs a redeploy. It is not attacker-amplifiable: the underlyings ignore raw donations.

RECOMMENDATION

Set `maxDepositActivationLossAssets` per underlying to at least its loss bound `ceil(totalAssets / (totalSupply + 10decimalsOffset))` -- 2 for the WETH vaults and Fluid USDC today, 1 for Morpho USDC.

That bound rises by one each time the share price crosses an integer over the vault's life. Prefer a relative bound (basis points of the deposit) that tracks the share price rather than a fixed absolute unit.

Validate the chosen value against the selected underlying over a realistic two-epoch maturity window, and do not reuse the Morpho-USDC value of 1 for any other underlying.

PROOF OF CONCEPT <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/guardian/src/async-yield-split-vault/test/guardian/ActivationLossToleranceDoS.t.sol>

M-16 Activation-loss tolerance DoSes deposits

CONTINUED

— RESOLUTION

Status: Resolved.

LOW **L-01**

RESOLVED

Unstake yield orphaned on full exit

CATEGORY Rewards

LOCATION `src/owb-staking-wrapper/contracts/OWBStakingWrapper.sol:408`

REVIEW Main Review

— DESCRIPTION

`withdraw` and `redeem` claim pending rewards via `_claimPendingYield`, snapshot `totalAssets()`, fix the receiver's `assets`, and burn the owner's shares before `_pullAssets` calls `staking.unstake`. The surplus branch at `OWBStakingWrapper.sol:426` treats any `received - unstakeAmount` from `unstake` as yield, fees it, and keeps the net in idle — never adding it to the exiting owner's already-fixed `assets`.

With the current OWB staking implementation this branch is dead code. `unstake` does auto-realize pending reward, but the wrapper claims first, so pending is already zero by then and `unstake` returns exactly the principal — the surplus is always zero.

It fires only under a staking implementation whose unstake-time release is not already drained by the prior `claimRewards` — behavior the wrapper deliberately supports and the upgradeable upstream proxy could be changed into. Then an exiting owner is underpaid by the surplus. When the last owner exits the whole supply, the surplus is left in idle with `totalSupply()` at zero and no shares to claim it, orphaned until a future depositor absorbs it. The fee is still charged on yield the owner never receives.

— RECOMMENDATION

Pay the exiting owner their share of any yield realized by their own `unstake`. In `_pullAssets`, measure the unstake `received` before the exit price is finalized. Fold the surplus into the `managedAssets` snapshot used to compute `assets`. The redeemer is then paid pro-rata, and the fee applies only to the portion that stays with remaining shareholders.

If instead the wrapper is only meant to run against a staking impl that surfaces rewards through `pendingReward`, enforce that assumption. Revert when `received != unstakeAmount` in `_pullAssets`, turning the silent socialization into an explicit failure. Either way, do not charge a yield fee on a full exit when the net surplus cannot be paid to anyone.

— RESOLUTION

Status: Resolved.

LOW **L-02**

RESOLVED

Withdraw rounding leaks value to harvest

CATEGORY Rounding
LOCATION AsyncYieldSplitVault.sol:802-823
REVIEW Main Review

DESCRIPTION

`withdraw/assets` computes `shares = previewWithdraw/assets` rounded up, then `_withdraw` reduces `accountedShareholderAssets` by `shareholderAssetsToBurn = shares.mulDiv(shareholderAssetsBefore, totalSupply(), Floor)` (AsyncYieldSplitVault.sol:811) while paying out only `assets` (line 818). Since the share count was rounded up, `shareholderAssetsToBurn >= assets`, so accounting drops by more than the assets paid.

That excess raises `availableYieldBalance()` (lines 282-293) and is harvestable: `harvestAllYield` sends it to `harvestRecipient` instead of leaving it to lift remaining holders' share price (the vanilla ERC-4626 outcome). `redeem(shares)` is unaffected — its `previewRedeem` uses the same Floor conversion as the accounting reduction, so the residual is zero. The leak is specific to the exact-asset Ceil path.

Impact is bounded: the Ceil overshoot is under one share-wei, so the residual per withdraw is at most `floor(accountedShareholderAssets / totalSupply)` asset-wei, independent of withdrawal size. A PoC confirms a 2-wei and a 33-ether withdraw each leak exactly 1 wei at a 1.5 share price. The loss falls on the withdrawer, accrues to the trusted `harvestRecipient`, and has no external-attacker profit path.

RECOMMENDATION

In `_withdraw`, reduce `accountedShareholderAssets` by the assets actually paid out (or by `min(shareholderAssetsToBurn, assets)`) so the rounding residual remains in shareholder backing instead of becoming harvestable surplus. Alternatively, document explicitly that exact-asset withdraw rounding accrues to harvestable yield.

RESOLUTION

Status: Resolved.

LOW **L-03**

ACKNOWLEDGED

Reward harvest assumes same-chain claim

CATEGORY	Trust Assumptions
LOCATION	ExternalRewardsAsyncYieldSplitVault.sol:107-158
REVIEW	Main Review

— DESCRIPTION

`harvestMerkleReward` claims by calling `merkleDistributor.claim` with `users[0] = address(this)` and then forwards the locally measured `rewardToken.balanceOf(address(this))` delta to `harvestRecipient` via `_forwardExternalReward` (`ExternalRewardsAsyncYieldSplitVault.sol:107-158`); the Morpho variant's `harvestMorphoUrdReward` follows the same local-claim-and-forward pattern. Both assume the configured distributor lives on the vault's own chain and credits the vault's own address.

Merkle campaigns, however, can reward a position held on one chain (e.g. a Base supply) with tokens that are only claimable on a different chain (e.g. OP on Optimism). In that setup the Base-deployed vault cannot call the Optimism distributor. It also cannot move tokens that land at its address there, so the incentives accrue to the vault address on the reward chain and become operationally unclaimable.

No deposited principal is at risk and external rewards are never counted in `totalAssets()` before realization, so share pricing is unaffected — the harm is limited to stranded incentive rewards for participants. The variant hardcodes a single same-chain claim surface and exposes no general call executor, so recovery depends on configuring the reward chain *before* accrual.

— RECOMMENDATION

Before enabling a cross-chain Merkle or Morpho campaign, configure Merkle reward forwarding and address remapping, or deploy a companion receiver on the reward chain. This routes incentives to a claimable destination. Document that these reward variants claim only same-chain rewards owned by the vault address, and treat cross-chain campaigns as an unsupported configuration absent such forwarding.

— RESOLUTION

Status: Acknowledged.

LOW **L-04**

PENDING

NAV price drift overpays redeems

CATEGORY Trust Assumptions

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:87-95,311-336`

REVIEW Main Review

DESCRIPTION

`AsyncNavVaultSettlement._settleRedeemRequest` prices redeem claims with `AsyncNavVaultCore.priceOf`, while `totalNav` is stored as a separate reported value. The settlement path never checks that `priceWad` agrees with `totalNav` before paying a claim at `AsyncNavVaultSettlement.sol:95`.

An inconsistent report can state `totalNav` of 100 USDC while setting a share price that pays a 100-share redeemer 200 USDC. The reporter trust model lowers practical severity, but a bad signed report still lets the claim path consume more liquidity than the reported vault NAV supports.

RECOMMENDATION

Validate `priceWad` against `totalNav` and the relevant share base before storing an attestation. If `priceWad` is the only canonical settlement value, remove or clearly downgrade `totalNav` from the trusted accounting surface.

RESOLUTION

Status: Pending.

LOW **L-05**

PENDING

Dust edits keep deposits reserved

CATEGORY DoS

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultRequests.sol:288-298`

REVIEW Main Review

— DESCRIPTION

`AsyncNavVaultRequests.decreaseDepositRequest` lets a controller or operator reduce a pending deposit. When the request is only partially reduced, `_decreaseDepositRequest` refreshes `requestEpoch` to the current epoch at `src/async-nav-vault/contracts/modules/AsyncNavVaultRequests.sol:296`. A user can repeatedly remove dust while leaving nearly all assets pending. The request never ages out, but `pendingDepositAssets` stays reserved and reduces `_unreservedIdleAssets`, limiting manager deployment capacity.

— RECOMMENDATION

Do not refresh `requestEpoch` for dust-sized partial decreases, or require a meaningful minimum reduction. Preserve the original economic deadline unless the user performs an explicit full refresh.

— RESOLUTION

Status: Pending.

LOW **L-06**

RESOLVED

Disabled manager strands yield, rewards

CATEGORY	Validation
LOCATION	AsyncYieldSplitVault.sol:739-746
REVIEW	Main Review

— DESCRIPTION

`harvestManager` is gated far more broadly than its name suggests. Besides `harvestYield` and `harvestAllYield`, it controls `closeEpoch` — the epoch advance that activates pending deposits — plus the Merkl, Morpho, and Aave reward claims and its own rotation (`AsyncYieldSplitVault.sol:739`). Despite the “harvest” name it is effectively the vault’s epoch-lifecycle key, so an operator may under-protect it. Rotation is single-step and unguarded. `setHarvestManager` assigns whatever address is passed, with no zero-check and no accept step, so a mis-typed address takes effect instantly. Because the setter is itself manager-gated and the vault has no admin role, a wrong transfer is irreversible — only a redeploy recovers. The worst case is `setHarvestManager(address(0))`, which disables every manager-gated function forever. No depositor principal is lost: `redeem` and `withdraw` are synchronous and pending deposits are cancellable for a refund. But the vault is permanently frozen for growth — `closeEpoch` can never advance, accrued yield is stranded above `_shareholderAssets` with no path to `harvestRecipient`, and external rewards may be forfeited at their upstream distributors.

— RECOMMENDATION

Make `setHarvestManager` two-step — propose, then accept from the new address — so this delicate role cannot be handed off, or zeroed, by a single mis-typed transaction; reject `address(0)` unless disabling is genuinely intended. Since the role also gates `closeEpoch` and deposit activation rather than just harvesting, consider renaming it (or its docs) to match its real authority. Alternatively, at minimum extend the `setHarvestManager` natspec to document that passing `address(0)` is irreversible and permanently strands accrued yield and external rewards. That way operators see the full repercussions, not just that it “disables manager-only actions.”

L-06 Disabled manager strands yield, rewardsCONTINUED

— RESOLUTION

Status: Resolved.

LOW **L-07**

RESOLVED

Root execution can erase validator owners

CATEGORY Unexpected Behavior

LOCATION `src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol`

REVIEW Main Review

DESCRIPTION

`onUninstall()` trusts any call from an initialized account. It only checks `_isInitialized(msg.sender)` and then calls `_clearAllOwners(msg.sender)`. It does not verify that Kernel is actually uninstalling the validator.

This matters because Kernel account execution makes the account itself call the target. A root-authorized operation can therefore execute `validator.onUninstall("")` directly. Inside the validator, `msg.sender` is the Kernel account, so the uninstall check passes and `_clearAllOwners` removes every owner. The call did not go through `uninstallModule()` or `_uninstallValidation()`, so Kernel does not apply its `RootValidatorCannotBeRemoved` protection and still keeps this validator selected as the root validator. After the owner set is cleared, `_isInitialized(account)` returns false. `validateUserOp` then fails before checking any signature, including a signature from the owner that was valid before the direct `onUninstall` call. The normal owner-removal functions protect the last owner with `LastOwnerRemoval`, but this lifecycle function bypasses that invariant by setting `ownerCount` to zero.

RECOMMENDATION

Do not let ordinary account execution clear validator state through lifecycle selectors. The safest fix is to make `onUninstall()` reject while this validator is still the caller's active root validator. This preserves Kernel's `RootValidatorCannotBeRemoved` rule even when the selector is called directly.

At the integration layer, also block account execution to module lifecycle selectors such as

`onInstall(bytes)` and `onUninstall(bytes)`. Lifecycle calls should only happen through Kernel's module-management functions.

If the validator does not need to delete owner storage during module uninstall, make `onUninstall()` a no-op. Owner changes should go through explicit owner-management functions that preserve the final-owner invariant.

L-07 Root execution can erase validator ownersCONTINUED

— RESOLUTION

Status: Resolved.

LOW **L-08**

RESOLVED

Withdrawals burn excess managed value

CATEGORY	Unexpected Behavior
LOCATION	AsyncYieldSplitVault.sol:802-823
REVIEW	Main Review

— DESCRIPTION

`AsyncYieldSplitVault._withdraw()` burns the user's wrapper shares, calls `_pullAssets(assets)`, transfers exactly `assets` to the receiver and emits the withdrawal event. `_pullAssets()` satisfies any idle shortfall with `yieldVault.withdraw(assets - _activeIdleAssets(), address(this), address(this))`. The wrapper never checks how much NAV was consumed from its underlying vault-share position. This matters because an ERC-4626 exact-asset `withdraw()` can burn underlying shares worth more than the assets returned to the wrapper. That can happen when the underlying vault has coarse shares relative to its price per share, applies withdrawal fees, or realizes loss during withdrawal. Deposits defend against the matching issue by checking that managed assets increased by at least the shares minted. Harvests also recheck collateralization after pulling assets. User withdrawals do not enforce a postcondition such as `managedAssetsBefore - managedAssetsAfter <= assets`. Consequently, a small withdrawal can destroy or transfer more managed value than the withdrawing user receives.

— RECOMMENDATION

Measure managed assets before and after a withdrawal and revert if the successful withdrawal consumed more managed value than the assets paid to the receiver:

```
```solidity
uint256 managedBefore = _totalManagedAssets();

_burn(owner, shares);
_pullAssets(assets);
IERC20(asset()).safeTransfer(receiver, assets);

uint256 managedAfter = _totalManagedAssets();
if (managedBefore > managedAfter + assets) {
 revert WithdrawalConsumedExtraManagedAssets(managedBefore, managedAfter, assets);
}
```
```

L-08 Withdrawals burn excess managed value

CONTINUED

Do not replace this with a plain `isCollateralized()` check. Undercollateralized wrappers still need to support pro-rata withdrawals. The guard should reject only withdrawals that make remaining holders worse by consuming extra NAV beyond the amount paid out.

Also consider reducing `maxWithdraw()` and `maxRedeem()` when the underlying vault's current share granularity or withdrawal fee would cause a requested exact-asset withdrawal to overburn value.

— RESOLUTION

Status: Resolved.

LOW **L-09**

PENDING

Deposit events attribute operator claims to the operator

CATEGORY Events

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol`

REVIEW Main Review

DESCRIPTION

`claimDeposit()` and `claimMint()` emit the ERC-4626 `Deposit` event with `msg.sender` as the first indexed parameter. Under the async deposit interface, an approved operator can claim for a controller. In that case `msg.sender` is the operator, not the controller that owns the request. ERC-7540 requires the first `Deposit` event parameter to be the controller and the second parameter to be the receiver when async deposit claims are executed. The request-specific `DepositRequestClaimed` event is correct, but standard indexers and adapters that read the ERC-4626 `Deposit` event will attribute the claim to the operator.

RECOMMENDATION

Emit the ERC-4626 `Deposit` event with `controller` as the first parameter and `receiver` as the second parameter for async deposit claims. Keep the request-specific event unchanged.

RESOLUTION

Status: Pending.

LOW **L-10**

PENDING

ERC165 misses the required ERC7575 vault id

CATEGORY Best Practices
LOCATION AsyncNavVaultCore.sol
REVIEW Main Review

— DESCRIPTION

`supportsInterface()` advertises `type(IERC7575).interfaceId`. The local `IERC7575` interface inherits ERC-4626 and ERC-165 in addition to declaring `share()`, so its Solidity-computed interface id is `0xa8d5fd65`. ERC-7575 requires vaults to return true for the constant vault interface id `0x2f0a18c5`. As a result, the vault returns false for the ERC-7575 id that standards-based integrations are expected to query, while returning true for a local non-standard id. The same function also returns true for `0x00000000` through an aggregate local interface id, which can further confuse interface discovery.

— RECOMMENDATION

Return true for the explicit ERC-7575 vault id `0x2f0a18c5` rather than relying on `type(IERC7575).interfaceId`. Keep the ERC-7540 operator, deposit and redeem ids explicit as well so local interface inheritance cannot change the advertised standards surface.

— RESOLUTION

Status: Pending.

LOW **L-11**

RESOLVED

ERC7575 share side is not advertised

CATEGORY Best Practices
LOCATION AsyncYieldSplitVault.sol
REVIEW Main Review

— DESCRIPTION

`share()` returns `address(this)`, so the vault contract is also the ERC-20 share token. The contract advertises the ERC-7575 vault interface id `0x2f0a18c5`, but `supportsInterface()` does not return true for the ERC-7575 share-token interface id `0xf815c03d`.

ERC-7575 requires vault contracts to advertise `0x2f0a18c5`. It also requires the share token to return true for `0xf815c03d` when the share token supports ERC-165. Since this vault uses itself as the share token and already implements `supportsInterface()`, ERC-7575 share-side discovery fails.

The contract also does not implement the optional `vault(address asset)` share-to-vault lookup. That method is a SHOULD-level part of ERC-7575.

— RECOMMENDATION

Return true for the ERC-7575 share-token interface id when `share() == address(this)`. If the project wants full share-to-vault lookup support, also implement `vault(address asset)` and return `address(this)` for the configured asset.

— RESOLUTION

Status: Resolved.

LOW **L-12**

RESOLVED

Zero-amount exits can harvest rewards

CATEGORY Unexpected Behavior
 LOCATION OWBStakingWrapper.sol
 REVIEW Main Review

DESCRIPTION

`withdraw()` calls `_claimPendingYield()` before it checks the requested asset amount against the owner's capacity. `redeem()` similarly calls `_claimPendingYield()` before it converts the requested share amount into assets. Neither function rejects a zero amount before claiming rewards.

Consequently, any address can call `withdraw(0, receiver, owner)` with any nonzero `owner` address and force the wrapper to claim pending OWB staking rewards. The later share allowance spend and burn use `0` shares, so the caller does not need to own wrapper shares or have allowance from the owner. The same issue exists for `redeem(0, receiver, owner)`.

This bypasses the intended direct-harvest access control. `harvestYield()` is restricted to `harvestRecipient`, but zero-amount exits let an arbitrary caller trigger the same reward claim and fee payment. The caller cannot steal the rewards: the configured fee still goes to `harvestRecipient` and the net reward remains idle in the wrapper for shareholders. The concrete impact is unauthorized harvest timing. For example, a third party can force fees to the current `harvestRecipient` immediately before a planned recipient rotation.

RECOMMENDATION

Make zero-amount exits no-ops before reward claiming or reject them explicitly before calling `_claimPendingYield()`.

```
```plain text
```

```
function withdraw(uint256 assets, address receiver, address owner_) public override nonReentrant returns (uint256) {
 if (assets == 0) return 0;
 _claimPendingYield();
 ...
}
```

```
function redeem(uint256 shares, address receiver, address owner_) public override nonReentrant returns (uint256) {
 if (shares == 0) return 0;
 ...
}
```

---

**L-12** Zero-amount exits can harvest rewardsCONTINUED

---

```
}
...
```

---

**— RESOLUTION**

Status: Resolved.

LOW **L-13**

PENDING

# AsyncNav share side lacks ERC7575 id

CATEGORY Best Practices

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol`

REVIEW Main Review

## DESCRIPTION

`share()` returns `address(this)`, so the vault contract is also the ERC-20 share token. ERC-7575 share-side discovery expects a share token that supports ERC-165 to advertise the share-token interface id `0xf815c03d`. The current `supportsInterface()` implementation only checks the local vault interfaces and inherited interfaces, so `supportsInterface(0xf815c03d)` returns false on the returned share token. Integrations that discover a vault from `share()` and then inspect the returned share token can therefore reject this vault or disable ERC-7575 handling for it. The contract also does not implement the ERC-7575 `vault(address)` share-to-vault lookup, so share-side discovery has no fallback.

## RECOMMENDATION

Advertise the ERC-7575 share-token id from `supportsInterface()` when `share() == address(this)`. Implement the ERC-7575 share-side lookup for the configured asset.

```
```plain text
```

```
function vault(address asset_) external view returns (address) {  
    return asset_ == asset() ? address(this) : address(0);  
}
```

```
function supportsInterface(bytes4 interfacedId) public view override returns (bool) {  
    return interfacedId == 0xf815c03d  
    || ...;  
}
```

RESOLUTION

Status: Pending.

LOW **L-14**

PENDING

Share allowance can retarget redeems

CATEGORY Unexpected Behavior

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultRequests.sol`

REVIEW Main Review

DESCRIPTION

`requestRedeem` lets a caller that is neither the owner nor an approved operator use ERC-20 share allowance as authorization. That is acceptable when the caller is only starting a new redemption for the shares it is allowed to move. The existing pending-request case gives that caller more power. After spending allowance only for the newly supplied `shares`, the function adds those shares to the existing request and rewrites `existingRequest.requestEpoch` for the whole pending balance.

Consequently, a small share allowance can change the settlement timing of a much larger redemption whose shares are already escrowed in the vault. The same caller would be rejected by

`refreshRedeemRequest`, `decreaseRedeemRequest`, `cancelRedeemRequest` and the claim and recovery functions unless it was the controller or an approved operator. `requestRedeem` therefore turns share-spending permission into an unintended request-management permission.

This does not let the allowance spender redirect assets or claim the redemption. It can still grief the controller's timing in a realistic stale-request scenario. For example, Alice has a 100-share redemption that has become stale, so she expects to cancel it or wait for a favorable NAV. Bob is not Alice's operator, but Alice previously gave him a one-share allowance for an unrelated interaction. Bob can call

`requestRedeem(1 share, Alice, Alice)` before the next signed NAV is posted. The vault spends only Bob's one-share allowance, but it retargets Alice's whole 101-share pending redemption to the current epoch. Once the signed NAV for that epoch is posted, Alice's cancellation reverts because the request is now claimable. She must either settle under Bob's chosen timing or wait for the request to become stale again.

RECOMMENDATION

Do not let share allowance alone edit an existing pending redemption request. In the

`existingRequest.pendingShares != 0` case, require the caller to be the `owner` or an approved ERC-7540 operator before changing `pendingShares`, `requestEpoch` or `requestTimestamp`. If non-operator allowance spending must remain supported, restrict it to creating a new owner-controlled redemption when there is no active pending request. A stronger design is to

L-14 Share allowance can retarget redeems

CONTINUED

split allowance-created redemptions into independent requests so spending a small allowance cannot retarget shares that were already escrowed.

— RESOLUTION

Status: Pending.

LOW **L-15**

PENDING

Same-epoch redemption views overcommit liquidity

CATEGORY Unexpected Behavior

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol`

REVIEW Main Review

DESCRIPTION

`_pendingRedeemClaimable` checks one pending redemption request against the full `_unreservedIdleAssets()` balance. It does not account for other pending redemptions that target the same epoch and can consume that liquidity first. Consequently, `claimableRedeemRequest`, `maxRedeem` and `maxWithdraw` can report that multiple same-epoch redemptions are fully claimable even when the vault only has enough idle assets for one of them. The first controller to settle consumes the shared liquidity. Later controllers then revert with `AsyncNavVaultInsufficientClaimLiquidity`. If another NAV is posted before more liquidity returns, the later requests become stale and cannot settle at the original target price. This does not overpay the first redeemer. The problem is that target-epoch claimability is advertised per request without an aggregate reservation, FIFO allocation or pro-rata allocation. Integrators and users can therefore see redemptions as claimable even though the full same-epoch set cannot settle from the current idle balance.

RECOMMENDATION

Make target-epoch redemption claimability aggregate-aware. A conservative fix is to return zero from lazy redemption claimability when same-epoch pending redemption liabilities exceed currently unreserved idle liquidity, unless a deterministic allocation has already been recorded. For stronger behavior, record explicit same-epoch liquidity allocations before user claims can consume the shared idle balance. The allocation can be FIFO or pro-rata, but once it is recorded, `claimableRedeemRequest`, `maxRedeem` and `maxWithdraw` should report only the amount allocated to that request.

RESOLUTION

Status: Pending.

LOW **L-16**

PENDING

Sole address owner can become unusable

CATEGORY Unexpected Behavior

LOCATION `src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol`

REVIEW Main Review

— DESCRIPTION

`setAddressOwnerFlags` only checks that at least one of `allowEoa` or `allow1271` remains true. It does not check whether the selected mode can currently validate signatures for the owner address, nor does it require another usable owner before changing the last owner's flags.

Consequently, a one-owner account can change an EOA address owner from EOA validation to ERC-1271-only validation while that address has no code. The owner remains enabled and `ownerCount` stays nonzero, so the validator still reports the account initialized. However, `_verifyAddressOwner` no longer tries ECDSA recovery and `_verifyAddressOwner1271` fails closed against the no-code address because the `staticcall` returns empty data. Normal Kernel owner mutation then cannot repair the flags because account execution must first be authorized by this validator.

— RECOMMENDATION

Reject flag updates that would leave the last owner without a currently usable validation method. The minimal guard is to prevent disabling `allowEoa` for a sole no-code address owner unless another active owner remains or to require a successful ERC-1271 probe before permitting an address owner to become ERC-1271-only.

— RESOLUTION

Status: Pending.

LOW **L-17**

PENDING

Asset deployment event omits manager

CATEGORY Events

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultAssetManagement.sol`

REVIEW Main Review

DESCRIPTION

`deployToAssetManager` transfers assets to the current `assetManager`, but `AssetsDeployedToAssetManager` records only the asset amount:

```
``plain text
IERC20(asset()).safeTransfer(assetManager, assets);
emit AssetsDeployedToAssetManager(assets);
`
```

The vault allows the asset manager to rotate through `setAssetManager` and the docs tell operators to monitor manager balances and `AssetsDeployedToAssetManager`. A log-only monitor therefore needs to replay constructor logs and every `AssetManagerSet`` event perfectly to infer which manager received each deployment. The deployment event itself does not identify the destination, so missed historical logs, late-starting indexers, reorg repair or rotation-era reconciliation can attribute deployed liquidity to the wrong manager. That makes it harder to distinguish funds still held by an old manager from funds deployed to the current manager.

Consequently, monitors can misreconstruct where vault liquidity was sent and may miss that a deployment starved redemption liquidity or that assets remain with an old manager after rotation.

RECOMMENDATION

Include the destination manager in the deployment event and emit the value used for the transfer:

```
``plain text
event AssetsDeployedToAssetManager(address indexed manager, uint256 assets);

address manager = assetManager;
IERC20(asset()).safeTransfer(manager, assets);
emit AssetsDeployedToAssetManager(manager, assets);
``
```

L-17 Asset deployment event omits managerCONTINUED

— RESOLUTION

Status: Pending.

LOW **L-18**

RESOLVED

Passkey removal also removes raw P-256 owners

CATEGORY Unexpected Behavior

LOCATION `src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol`

REVIEW Main Review

DESCRIPTION

`removePasskeyOwners` does not check that each supplied owner id is marked in `passkeyOwnerExists`. It forwards every id to `_removeP256Owner`, which removes any active P-256 owner and then emits `PasskeyOwnerRemoved` only when the removed id was actually marked as a passkey. Consequently, an account-authorized call to the passkey-specific removal function can remove an unmarked raw P-256 owner. This contradicts the exposed owner split: `isP256Owner(account, id)` can be true while `isPasskeyOwner(account, id)` is false, yet `removePasskeyOwners([id])` still deletes the raw owner. The same authorization boundary protects all owner mutations, so this is not an external takeover. Consequently, tooling or account modules that call the passkey remover because they intend to affect only passkey credentials can accidentally remove raw P-256 recovery keys.

RECOMMENDATION

Make `removePasskeyOwners` enforce the passkey marker before deleting owner state. For each supplied id, require `ownerStorage[msg.sender].passkeyOwnerExists[id]` to be true, then call `_removeP256Owner`.

RESOLUTION

Status: Resolved.

LOW **L-19**

PENDING

Decrease events omit retargeted epoch

CATEGORY	Events
LOCATION	src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol
REVIEW	Main Review

— DESCRIPTION

`DepositRequestDecreased` and `RedeemRequestDecreased` report the request id, controller, receiver, decreased amount and remaining balance. They do not report the request epoch that applies to the remaining balance.

This matters because a partial decrease retargets the surviving request. In `_decreaseDepositRequest`, when `request.pendingAssets` remains nonzero, the helper sets `request.requestEpoch = currentEpoch()` and updates `request.requestTimestamp`. `_decreaseRedeemRequest` does the same for remaining redemption shares. The refresh events include the new `requestEpoch`, but the decrease events do not.

Consequently, a log-only indexer that follows the deployment docs and monitors request edit events can keep the old target epoch for a partially decreased request. That can make the indexer mark the request claimable at the wrong NAV, miss the new target epoch or report stale pending-request alerts incorrectly. On-chain settlement still uses storage, so this is an off-chain accounting and monitoring risk rather than a direct fund-loss bug.

— RECOMMENDATION

Include the retargeted request epoch in both partial decrease events and emit the updated value after the helper mutates storage.

```
``plain text
event DepositRequestDecreased(
    uint256 indexed requestId,
    address indexed controller,
    address indexed receiver,
    uint256 assets,
    uint256 remainingAssets,
    uint256 requestEpoch
);
`
Apply the same change to RedeemRequestDecreased`.
```

L-19 Decrease events omit retargeted epochCONTINUED

— RESOLUTION

Status: Pending.

LOW **L-20**

PENDING

Unbounded admin delays can block admin handoff

CATEGORY	DoS
LOCATION	src/async-nav-vault/contracts/access/TransferableAccessControl.sol
REVIEW	Main Review

— DESCRIPTION

`TransferableAccessControl` inherits `OpenZeppelin AccessControlDefaultAdminRules` without adding any protocol-level upper bound to the inherited `changeDefaultAdminDelay(uint48)` setter. The current default admin can therefore schedule `type(uint48).max` as the next default-admin delay. `OpenZeppelin` caps the wait before an increased delay takes effect to `defaultAdminDelayIncreaseWait()`, which is five days by default. After that schedule passes, `defaultAdminDelay()` returns the huge value. A normal admin handoff then reaches `_beginDefaultAdminTransfer`, which computes `SafeCast.toUint48(block.timestamp) + defaultAdminDelay()`. With `defaultAdminDelay() == type(uint48).max`, any nonzero realistic timestamp makes the checked `uint48` addition revert. The same overflow blocks a direct reduction to a usable delay. Calling `changeDefaultAdminDelay(0)` after the huge delay has materialized computes a wait of `currentDelay - newDelay`, then adds that wait to the current timestamp. That addition also reverts. The admin can replace or roll back the pending huge delay before it takes effect, so this is not an unprivileged takeover. It is a configuration trap: if the five-day rollback window is missed, future default-admin recovery or migration can become practically unavailable while the current admin remains the only default admin.

— RECOMMENDATION

Override `changeDefaultAdminDelay` or wrap the inherited admin-delay controls so production deployments reject delays above a documented maximum that matches the governance recovery SLA.

— RESOLUTION

Status: Pending.

LOW **L-21**

RESOLVED

Chain-bound leaf version lacks SDK helper

CATEGORY	Configuration
LOCATION	src/glider-kernel-validator/contracts/interfaces/IMultiOwnerDeterministicValidator.sol, src/glider-kernel-validator/lib/GliderRootValidatorDigests.sol
REVIEW	Main Review

DESCRIPTION

The validator exposes public helpers for SDK-critical multichain constants, but only the replayable leaf version is available through the deployed contract ABI. `IMultiOwnerDeterministicValidator` exposes `getReplayableChainNamespace()`, `getNonReplayableChainNamespace()` and `getReplayableLeafVersion()`, while `GliderRootValidatorDigests.defaultLeafVersion()` remains internal.

Chain-bound validation still requires `authorization.leafVersion == GliderRootValidatorDigests.defaultLeafVersion()`. Today that value is `1` and repository tests and docs hardcode `1` when building chain-bound leaves. This contradicts the SDK boundary documented in `src/glider-kernel-validator/docs/ACTORS.md`, where the `Off-chain signer/SDK` row says SDKs must use the contract helper getters as the source of truth for digests, namespaces, account locators and leaf versions.

The result is an avoidable ABI compatibility hazard: SDKs can query the replayable leaf version but must hardcode the chain-bound version. If the default leaf version changes in a later deployment or SDK code accidentally treats replayable and chain-bound versions symmetrically, owners can sign chain-bound Merkle roots whose leaves cannot be verified onchain.

RECOMMENDATION

Expose the chain-bound/default leaf version in the public interface, for example:

```
``plain text
function getNonReplayableLeafVersion() external pure returns (uint32 version);
`
or:
`plain text
function getDefaultLeafVersion() external pure returns (uint32 version);
`
```

Update docs and SDK conformance tests so chain-bound builders query this helper instead of hardcoding `1`.

L-21 Chain-bound leaf version lacks SDK helper

CONTINUED

— RESOLUTION

Status: Resolved.

LOW L-22

PENDING

Public EIP-712 shows wrapper domain

CATEGORY Unexpected Behavior

LOCATION src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol

REVIEW Main Review

— DESCRIPTION

`MultiOwnerDeterministicValidator` inherits `Solady EIP712` only to wrap direct and chain-bound ERC-1271 owner checks. Therefore the public `eip712Domain()` inherited from `Solady` advertises `name = "ERC1271Validator"` and `version = "0.0.1"`.

Address EOA owners, raw P-256 owners and WebAuthn owners do not sign that domain. Their direct and chain-bound multichain digests are built by `GliderRootValidatorDigests` with `name = "GliderRootValidator"` and `version = "1"`. Consequently, an SDK or wallet flow that follows the validator's public EIP-5267 domain metadata can produce a well-formed EIP-712 signature that the validator rejects.

— RECOMMENDATION

Expose the owner-facing Glider domain explicitly and document that `eip712Domain()` is only the ERC-1271 wrapper domain. A safer option is to avoid exposing misleading EIP-5267 metadata from the validator contract by moving the ERC-1271 wrapper hash into a small helper or by overriding `eip712Domain()` with a clear revert or an explicit wrapper-only signal.

— RESOLUTION

Status: Pending.

LOW **L-23**

PENDING

Emptied NAV vault state not reset

CATEGORY	Logical Error
LOCATION	AsyncNavVaultSettlement.sol:111-139
REVIEW	Main Review

— DESCRIPTION

`AsyncNavVault` is fully reporter-priced. `pushSnapshot(epoch, priceWad, totalNav)` takes the price and total NAV as independent fields, with no on-chain check against `totalSupply()` or the real balance. NAV accuracy is an off-chain Non-Goal.

When the last holder redeems, `totalSupply()` hits zero but the vault resets no state. The stale price persists, and `totalAssets()` (`totalSupply * priceWad`) reads zero, blind to the real balance.

`requestDeposit` has no zero-supply guard. Two exposures follow when the reported price drifts from real backing.

First, stranded idle. The reporter can mark below the value realized on unwind, via conservative marks on illiquid Ondo GM and Perp positions or drift. The surplus left after a full exit has no share backing and no recover path. A dust depositor becomes sole holder and redeems it once the reporter folds idle into the next mark.

Second, the re-bootstrap price. The contract never defines the price for an emptied vault, so the reporter must keep the stale price or reset to par. An attacker pre-positions a deposit around either choice. A reset-to-par over-mark lets an early holder over-mint and extract the gap from the next depositor.

— RECOMMENDATION

Define the zero-supply transition on-chain instead of leaving it to the reporter. While `totalSupply() == 0`, reject deposits when `idleAssets() > 0`, so no fresh depositor mints against orphaned value, and pin the re-bootstrap price, for example by forcing par and ignoring the stale `priceWad`.

Add an admin or reporter path to sweep idle assets that have no share backing to a protocol recipient. As defense in depth, reject any snapshot whose `priceWad` is inconsistent with `totalNav / totalSupply()` beyond a tolerance, so price and NAV cannot be posted decoupled from backing.

— RESOLUTION

Status: Pending.

LOW **L-24**

ACKNOWLEDGED

Owner-mutation guard lacks origin bind

CATEGORY	Access Control
LOCATION	MultiOwnerDeterministicValidator.sol:1184-1188
REVIEW	Main Review

— DESCRIPTION

Owner-mutating entry points, including `addAddressOwners`, `replaceAllOwners`, and the remove functions, gate access through `_enforceOwnerMutationAuth`. It only verifies that `msg.sender` is an initialized smart account with code (`MultiOwnerDeterministicValidator.sol:1185`).

It does not bind the call to which validator or module actually authorized it. A Kernel account can install restricted session-key or permission validators, expecting them to be unable to change root ownership. But if such a validator does not explicitly blocklist these selectors, it can route a call into the module and mutate the owner set. This leads to owner takeover from a supposedly limited credential. The exposure is bounded since an unrestricted executor already has full control.

— RECOMMENDATION

Bind owner mutation to the root authorization path. Require that owner-mutating selectors are reachable only through the account's primary validator, by checking the active validator context or maintaining an allowlist.

Document that integrators must blocklist these selectors in any restricted session or permission validator.

— RESOLUTION

Status: Acknowledged.

LOW L-25

ACKNOWLEDGED

Partial claims below max values can revert

CATEGORY	Unexpected Behavior
LOCATION	src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol
REVIEW	Main Review

— DESCRIPTION

`maxDeposit()` and `maxMint()` report the full remaining claimable request amount when `_previewClaimShares(request.assets)` is nonzero. The exact-share `mint()` function then prices any smaller claim with `_assetsForClaimShares(shares)`, which does not include the activation loss that `_activateDepositAssets()` may later observe from the underlying yield vault. Consequently, a controller can see a positive `maxMint()` but have a smaller exact-share claim revert with `EpochDepositSharesZero`. The exact-asset `deposit()` variant has the same edge at dust sizes: `maxDeposit()` can report 2 asset units while `deposit(1, receiver, controller)` reverts with `DepositValueBelowAssets` when the live yield vault round trip loses 1 unit. In the observed Base fork cases, this does not lock the request or strand user funds. The controller can still claim the full amount reported by `maxDeposit()` or `maxMint()`, and the request remains cancellable until a claim consumes it. The issue is narrower: routers and frontends may treat the advertised max as proof that any smaller partial claim will execute, but dust-sized `deposit()` or `mint()` calls can still revert.

— RECOMMENDATION

Make the max views conservative for every claim amount they advertise. One option is to cap `maxMint()` to the largest share amount whose computed asset input still mints at least that many shares after activation loss. Another option is to make partial exact-share claims consume enough assets to cover the tolerated activation loss, while never exceeding the request's remaining assets. For exact-asset claims, return zero from `maxDeposit()` when the smallest executable partial claim would be below the activation-loss floor, or document that only full request claims are supported for integrations.

— RESOLUTION

Status: Acknowledged.

LOW L-26

ACKNOWLEDGED

Temporary deposits reset pooled staking age

CATEGORY Unexpected Behavior

LOCATION `src/owb-staking-wrapper/contracts/OWBStakingWrapper.sol`

REVIEW Main Review

DESCRIPTION

`OWBStakingWrapper` stakes every depositor's OWB into one aggregated upstream position owned by `address(this)`. A new deposit therefore calls `staking.stake(assets)` from the wrapper itself, which lets OWB Staking recompute the wrapper position's weighted `startTime`.

The wrapper prices the temporary depositor's shares correctly and lets that depositor exit without taking incumbent principal. The problem is that the upstream staking age is not restored when the temporary depositor redeems. After a large short-lived deposit exits, the wrapper can be left with essentially the same managed assets for incumbent holders but a much newer upstream `startTime` and lower time-weighted reward weight.

This is distinct from the documented operational tradeoff that a real long-term top-up makes the pooled position younger. Here the depositor can enter with a large amount, immediately redeem, recover the temporary principal and leave incumbents with reduced future reward accrual.

RECOMMENDATION

Enforce the staking-age impact limit on-chain. Before accepting a deposit or mint, compute the projected upstream `startTime` after the new stake and revert if the shift is above a configured threshold.

If large public deposits must remain available, add a minimum holding period or exit fee for deposits that materially move the pooled `startTime`, so a depositor cannot immediately exit after reducing incumbents' future reward weight. If the intended design is to accept this risk, document the short-lived deposit-and-redeem case explicitly in user-facing materials, not only as an operational top-up consideration.

RESOLUTION

Status: Acknowledged.

LOW L-27

RESOLVED

No device-bound passkey policy

CATEGORY	Validation
LOCATION	StrictWebAuthn.sol
REVIEW	Main Review

DESCRIPTION

WebAuthn assertions carry an authenticator-data flags byte (`authenticatorData[32]`) whose bits include backup eligibility and backup state:

- `AUTH_DATA_FLAGS_UP` (`0x01`): user present.
- `AUTH_DATA_FLAGS_UV` (`0x04`): user verified.
- `AUTH_DATA_FLAGS_BE` (`0x08`): backup eligible. The credential may be synced/exported to a cloud account (a multi-device passkey).
- `AUTH_DATA_FLAGS_BS` (`0x10`): backup state. The credential is currently backed up/synced.

`BE` distinguishes two security models. A device-bound passkey (`BE=0`) is hardware-bound: the private key never leaves the secure element. A synced passkey (`BE=1`) is eligible to be backed up to a cloud account and restored onto other devices, so its security also depends on that cloud account.

`_checkAuthFlags` enforces user presence, optional user verification, and the backup *consistency* invariant, but offers no policy over backup *eligibility*:

```
``plain text
if (flags & AUTH_DATA_FLAGS_BE != AUTH_DATA_FLAGS_BE) {
if (flags & AUTH_DATA_FLAGS_BS == AUTH_DATA_FLAGS_BS) {
return false;
}
}
`
```

The only `BE` logic rejects the impossible combination `BE=0 + BS=1`. It does not let an account reject `BE=1`, so synced passkeys (`BE=1, BS=0` and `BE=1, BS=1`) are accepted unconditionally. The per-owner policy struct also has no field to express a preference:

```
``plain text
struct WebAuthnPolicy {
bytes32 rpIdHash;
bytes32 originHash;
bool requireUserVerification;
bool enabled;
}
```

L-27 No device-bound passkey policy

CONTINUED

}

``

There is no `requireDeviceBound` flag, so an account that wants hardware-bound-only credentials cannot enforce it.

Impact

For a `BE=1` synced passkey, the key's security no longer rests solely on the user's device; it also rests on the cloud account hosting the sync (its password, 2FA, and account-recovery flow) and on anyone who can restore that backup onto a new device. If that cloud account is compromised, the passkey can be restored on an attacker device and used to sign, fully passing this validator, because nothing rejects `BE=1`. For a high-value or institutional Glider account that intends a key to live on specific hardware and nowhere else, there is no way to encode that intent on-chain. Because the flags are checked in the on-chain validation path, the SDK cannot compensate; a relayer or signer can simply present a `BE=1` assertion and it is accepted.

— RECOMMENDATION

Add an optional per-owner flag to `WebAuthnPolicy` (for example `bool requireDeviceBound`) and enforce it in `_checkAuthFlags`: when set, reject any assertion with `BE=1` (`flags & AUTH_DATA_FLAGS_BE != 0`). Leave it `false` by default so existing synced-passkey UX is unaffected.

— RESOLUTION

Status: Resolved.

LOW L-28

RESOLVED

P-256 public keys not validated on curve

CATEGORY Validation
 LOCATION StrictWebAuthn.sol
 REVIEW Main Review

— DESCRIPTION

When a P-256 owner (raw or passkey) is registered, the validator stores the caller-supplied public-key coordinates after a single non-zero check. It never verifies that `(pubKeyX, pubKeyY)` is a valid secp256r1 point or that each coordinate is a valid field element.

The only coordinate validation in `_storeP256Owner` is:

```
``plain text
if (ownerInit.ownerId == bytes32(0) || ownerInit.pubKeyX == 0 || ownerInit.pubKeyY == 0) {
  revert InvalidOwnerAddress();
}
```

There is no check that:

- each coordinate is less than the secp256r1 field prime `p`, or
- the pair satisfies the curve equation $y^2 == x^3 + a \cdot x + b \pmod{p}$.

Any non-zero pair (a truncated coordinate, swapped X/Y, wrong encoding, coordinate $\geq p$, or an off-curve point) therefore passes. The owner is then incremented into the active set and stored as enabled:

```
`plain text
accountStorage.p256OwnerExists[ownerInit.ownerId] = true;`
```

The invalid key is now an enabled owner, counts toward `ownerCount / MAX_OWNERS`, and (because `ownerCount > 0`) makes `_isInitialized` return true, so installation appears to succeed. The error only surfaces later, silently, at verification time. `_verifyP256RawOwner` / `_verifyP256WebAuthnOwner` reach `StrictWebAuthn._verifyP256`, which staticcalls the selected verifier with the stored coordinates.

A correct P-256 verifier (RIP-7212 or the Daimo verifier) rejects an off-curve or out-of-range key, so `_verifyP256` returns `false` for every signature against that owner.

— RECOMMENDATION

L-28 P-256 public keys not validated on curve

CONTINUED

Validate the public key in `_storeP256Owner` before storing it: require `0 < pubKeyX < p` and `0 < pubKeyY < p`, and require `pubKeyY^2 == pubKeyX^3 + a*pubKeyX + b (mod p)` for the secp256r1 parameters, reverting otherwise.

— RESOLUTION

Status: Resolved.

LOW L-29

RESOLVED

Direct authorizations cannot be revoked

CATEGORY	Signatures
LOCATION	MultiOwnerDeterministicValidator.sol:805-816
REVIEW	Main Review

— DESCRIPTION

The validator verifies ERC-1271 authorizations in two modes. Multichain authorizations carry a mandatory `expiry` and a `batchNonce`: `_validateMultichainAuthorization` reverts once the expiry passes (`MultiOwnerDeterministicValidator.sol:961`). Direct authorizations carry neither.

A direct authorization is accepted whenever the owner is currently enabled and an ECDSA signature recovers over `hashDirectAuthorization(validator, account, ownerId, operationHash)`. That digest binds no expiry and no install generation. `onUninstall` clears the owner set, so the signature stops validating and looks revoked. But reinstalling the same owner key re-enables it, and the identical earlier signature validates again.

Uninstall is therefore not a durable revocation for an outstanding direct authorization. An account that uninstalls to kill a pending ERC-1271 grant (a session key, permit, or off-chain agreement keyed by `operationHash`) silently revives it on the next reinstall of that key. The userOp lane is unaffected, since the EntryPoint nonce is folded into its operation hash. The exposure is the ERC-1271 lane, where `operationHash` is supplied by the integrating dapp.

— RECOMMENDATION

Mirror the protection the validator already applies to multichain authorizations. Two options. Fold a per-account install generation into the direct digest and bump it on every `onInstall` and `onUninstall`. A signature from a prior installation then never validates after a reinstall. Alternatively, require an `expiry` on direct authorizations, as multichain mode already does.

Document that uninstalling the validator permanently invalidates outstanding direct authorizations, so operators do not treat it as a temporary kill switch.

— RESOLUTION

Status: Resolved.

LOW **L-30**

RESOLVED

Advertised ERC7540 delegation is rejected

CATEGORY	Compatibility
LOCATION	src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol
REVIEW	Main Review

DESCRIPTION

`AsyncYieldSplitVault` advertises ERC7540 deposit and operator support from `supportsInterface()`, but `requestDeposit()` rejects every request where `controller != owner` before checking whether the caller is an approved operator for the owner.

ERC7540's async deposit surface separates the account that supplies assets (`owner`) from the account that controls the request (`controller`). The function signature and operator model allow integrations to submit a request where an approved operator spends from `owner` while assigning the resulting request to a separate `controller`. This vault instead hard-reverts that role separation with `DepositRequestControllerOwnerMismatch`.

Consequently, routers, allocators, custody accounts and account-abstraction integrations that rely on ERC7540 role separation can fail after discovering the vault as ERC7540-compatible. The affected flow is async deposit intake: assets are not lost, but integrations can route deposits into reverting transactions or need vault-specific special cases.

RECOMMENDATION

Either implement true controller/owner separation for async deposit requests or expose and document a project-specific capability signal that tells adapters `controller` must equal `owner`. If the vault keeps the restricted model, integration documentation and router adapters should explicitly pass the owner as the controller and should not treat the advertised ERC7540 deposit/operator ids as support for separated request controllers.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-01

ACKNOWLEDGED

Unharvested yield wiped by vault loss

CATEGORY Informational
LOCATION AsyncYieldSplitVault.sol:282-293
REVIEW Main Review

DESCRIPTION

The yield available to the recipient is `availableYieldBalance() = activeManaged - shareholderAssets`, floored at zero at `AsyncYieldSplitVault.sol:282-293`. The `accountedShareholderAssets` is a high-water mark that a `yieldVault` loss never writes down. So the recipient's accrued-but-unharvested surplus is the first loss-absorbing layer: a loss reduces `_activeManagedAssets` and erases that surplus to zero before shareholder principal is touched.

A recipient who calls `harvestYield` only weekly can lose a whole week of accrued yield to one loss event. A recipient who harvests daily realizes most of it first and exposes only the last day's accrual. This is the intended waterfall (the yield buffer protects principal), but the recipient's exposure scales directly with how stale their last harvest is.

RECOMMENDATION

Document that unharvested yield is at risk from `yieldVault` losses and advise the harvest recipient to call `harvestYield` / `harvestAllYield` frequently so accrued yield is realized before a loss can erase it. Consider an operational schedule (e.g. per-epoch harvest) rather than ad-hoc harvesting.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-02

PENDING

Redeem liquidity is not reserved

CATEGORY Trust Assumptions

LOCATION `src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:87-115`

REVIEW Main Review

DESCRIPTION

NAV redeem requests escrow shares, but `_unreservedIdleAssets()` (`AsyncNavVaultCore.sol:483-489`) reserves only pending/claimable deposits and `pendingClaims` — not `pendingRedeemShares`. So `deployToAssetManager` can move idle assets out, and a redeem reaching its single settlement epoch (`_requireTargetEpoch` permits exactly `requestEpoch+1`) can revert `AsyncNavVaultInsufficientClaimLiquidity` (`AsyncNavVaultSettlement.sol:121`). Once the next NAV posts, the request becomes `AsyncNavVaultRequestStale` and must be refreshed/cancelled and re-priced at the later NAV.

Filed as Informational because this is an explicitly documented, accepted product tradeoff — not accidental behavior. `ASSUMPTIONS_AND_TRADEOFFS.md` assumes "the asset manager returns liquidity before redemption target epochs when redemptions should settle," and its "User-Borne Missed Target" tradeoff states that if redemption liquidity is unavailable at the target and the next NAV is posted before settlement, the request becomes stale (the vault deliberately does not settle at a substitute epoch). The deposit-vs-redeem reservation asymmetry is itself called out there as "a liquidity-management risk, not an insolvency bug," and "stale-price rescue settlement" is listed as an explicit Non-Goal.

Residual note for the client: redemption settlement liveness depends on the trusted asset manager's off-chain liquidity coordination plus reporter cadence. No funds are lost and the redeemer keeps refresh/cancel recourse, but a redeemer who lands in a low-liquidity epoch is forced to re-queue and accept a later (possibly lower) NAV.

RECOMMENDATION

No code change is required if the documented assumption holds; this is surfaced so the team consciously re-confirms intent. To reduce reliance on off-chain coordination, optionally: (1) reserve an estimated liability for `pendingRedeemShares` inside `_unreservedIdleAssets()` so queued redemptions cannot be starved by `deployToAssetManager`; or (2) let a settled-but-unfunded request remain claimable at its original epoch price once liquidity returns, instead of forcing a re-price via refresh. At minimum, monitor idle balance vs pending-redeem liability and confirm the asset-manager liquidity SLA ahead of each redemption target epoch.

I-02 Redeem liquidity is not reserved

CONTINUED

— **RESOLUTION**

Status: Pending.

INFORMATIONAL I-03

PENDING

Default admin transfers have no effective delay

CATEGORY Configuration

LOCATION src/async-nav-vault/contracts/access/TransferableAccessControl.sol

REVIEW Main Review

DESCRIPTION

`TransferableAccessControl` opts into OpenZeppelin `AccessControlDefaultAdminRules`, but it initializes the rule set with a zero-second delay:

```
``plain text
constructor(address initialDefaultAdmin) AccessControlDefaultAdminRules(0, initialDefaultAdmin)
{}
```

OpenZeppelin schedules a default-admin transfer at `block.timestamp + defaultAdminDelay()`. With this constructor, `beginDefaultAdminTransfer` schedules the handoff at the current timestamp. `acceptDefaultAdminTransfer` can then complete as soon as a later block has a greater timestamp. The same timing applies to default-admin renunciation. A current admin can call `beginDefaultAdminTransfer(address(0))`, wait one later timestamp and then call `renounceRole(DEFAULT_ADMIN_ROLE, admin)`. That leaves the vault with no default admin and disables all `onlyRole(DEFAULT_ADMIN_ROLE)` functions. This is an intended OpenZeppelin feature, but the zero delay removes the review period that would normally protect against accidental or compromised-key renunciation.

Post-deployment hardening also has a gap. Calling `changeDefaultAdminDelay(newDelay)` after deployment schedules the new delay for later. Until that schedule passes, `defaultAdminDelay()` still returns the current delay, which is zero. Therefore a deployment runbook cannot immediately recover the intended delay unless the constructor sets it from the start. The repository docs tell operators to monitor pending default-admin transfers. With the deployed default, that monitoring has almost no reaction window. If the current default admin key is compromised, the attacker can schedule a new attacker-controlled admin and accept the transfer almost immediately, leaving little practical opportunity for the legitimate admin to call `cancelDefaultAdminTransfer`.

The default admin is already a trusted role and can rotate the singleton reporter and asset manager.

I-03 Default admin transfers have no effective delay

CONTINUED

— RECOMMENDATION

Give `TransferableAccessControl` a nonzero initial admin-transfer delay, either as an explicit constructor parameter or as a protocol constant chosen from the production governance response SLA. Add a deployment check that rejects `defaultAdminDelay() == 0` for production deployments. If immediate default-admin transfers are intentional, document that operational choice clearly and do not rely on monitoring pending default-admin transfers as a meaningful recovery control.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-04

RESOLVED

Address-owner ECDSA accepts high-s

CATEGORY Best Practices

LOCATION MultiOwnerDeterministicValidator.sol:1034-1057

REVIEW Main Review

DESCRIPTION

The EOA address-owner path verifies signatures with Solady `ECDSA.tryRecover` (`MultiOwnerDeterministicValidator.sol:1047`), which does not enforce a canonical low-s value. The P-256 path rejects high-s at `StrictWebAuthn.sol:177`. So for any valid EOA owner signature, the high-s twin recovers the same owner and also validates, as confirmed by the proof of concept.

The high-s twin does not create a second account-level execution, because nonce and `userOpHash` replay protection is digest-based, not signature-byte-based. But the validator exposes two distinct valid signature byte strings for the same owner authorization.

This can bypass byte-level signature deduplication in bundlers, relayers, paymasters, analytics, or ERC-1271 integrators that incorrectly track consumed signatures by raw signature hash. Solady documents this behavior and tells callers to rely on a nonce rather than signature uniqueness, which the validator does on-chain. The exposure is an off-chain integration hazard, not an on-chain defect.

RECOMMENDATION

Enforce low-s on the EOA owner path. Use a recovery routine that rejects high-s values, those above half the group order, and a malformed `v`, such as OpenZeppelin `ECDSA.recover`, matching the P-256 path. This yields one canonical signature per authorization and protects off-chain consumers that key on signature bytes.

On-chain replay is already prevented by the EntryPoint nonce, so this is a hardening and consistency change, not a fix for an exploitable bug.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-05

RESOLVED

Dust redeem burns shares for zero assets

CATEGORY Unexpected Behavior

LOCATION src/owb-staking-wrapper/contracts/OWBStakingWrapper.sol

REVIEW Main Review

DESCRIPTION

`OWBStakingWrapper.redeem()` checks that the requested shares do not exceed `maxRedeem(owner_)`, converts those shares to assets with floor rounding, and then calls `_withdraw()`.

The current implementation handles the full-balance impairment case because `maxRedeem(owner_)` returns zero when the owner's full balance would redeem to zero assets. It still does not reject a smaller nonzero redeem amount that rounds down to zero assets.

Because wrapper shares use a decimal offset, dust-sized share amounts can be worth less than one wei of OWB. In that case `previewRedeem(dustShares) == 0`, but `dustShares <= maxRedeem(owner_)`, so `redeem(dustShares, receiver, owner)` burns the shares and transfers zero OWB.

RECOMMENDATION

Add the same zero-output protection used on the mint/deposit side to `redeem()`:

```
``plain text
if (shares != 0 && assets == 0) {
  revert RedeemWouldReturnZeroAssets(shares);
}
```

Place the guard after computing `assets` and before `_withdraw()`. Keep the existing `maxRedeem()` override, since it correctly blocks the larger full-balance impairment case.

RESOLUTION

Status: Resolved.

INFORMATIONAL I-06

PENDING

ZeroDev library findings report

CATEGORY Warning
LOCATION lib/kernel/*, lib/kernel-7579-plugins/*
REVIEW Main Review

DESCRIPTION

Guardian Audits reviewed the ZeroDev `Kernel` library and the ZeroDev `Kernel 7579 plugins` library in this repository.

The related findings have been consolidated into one report so the affected maintainers can review the full set together:

[<https://gist.github.com/GuardianAudits/ff9ea0bfbe8709fc5b2df2b516bdd734>]

(<https://gist.github.com/GuardianAudits/ff9ea0bfbe8709fc5b2df2b516bdd734>)

RECOMMENDATION

Use the linked report as the issue packet for the reviewed ZeroDev libraries. Triage each listed finding against the current ZeroDev `Kernel` and `Kernel 7579 plugins` code, then track fixes and acknowledgements against the individual issues in that report.

RESOLUTION

Status: Pending.

INFORMATIONAL **I-07**

PENDING

Dead error NavOracleInvalidConfig

CATEGORY Superfluous Code
LOCATION AsyncNavVaultErrors.sol:28
REVIEW Main Review

— DESCRIPTION

`AsyncNavVaultErrors.sol:28` declares `error NavOracleInvalidConfig()`. It is never reverted or referenced anywhere in scope. Only `AsyncNavVaultInvalidConfig` and `NavOracleInvalidAttestation` are used.

This is dead code that bloats the ABI. It may also indicate a validation path the author intended but did not wire up.

— RECOMMENDATION

Remove the unused error, or wire it into the oracle-config validation path it was presumably meant to guard.

— RESOLUTION

Status: Pending.

INFORMATIONAL I-08

RESOLVED

Zero activation-loss bricks deposits

CATEGORY	Configuration
LOCATION	AsyncYieldSplitVault.sol:908-952
REVIEW	Main Review

DESCRIPTION

`_activateDepositAssets` is fail-closed. When a deposit into the yield vault shows any loss exceeding `maxDepositActivationLossAssets`, it reverts at line 952.

ERC-4626 deposits routinely lose one atomic unit to rounding. So configuring

`maxDepositActivationLossAssets` to 0 makes essentially every activation revert, bricking all deposit claims for that vault.

The parameter is immutable and the constructor performs no guard rejecting a zero value. A misconfigured deployment is therefore unrecoverable without redeploying.

This is an informational deployment-hardening note rather than a code flaw in normal operation.

`DEPLOYMENT.md` already recommends setting the tolerance to at least 1 and gating it with a fork test, so correct deployment avoids the issue entirely.

RECOMMENDATION

Reject a zero `maxDepositActivationLossAssets` in the constructor by requiring it to be at least 1. This makes the misconfiguration impossible at deploy time, while keeping the `DEPLOYMENT.md` fork-test gate as a secondary check.

Document the minimum recommended tolerance relative to the underlying asset's decimals.

RESOLUTION

Status: Resolved.

HIGH **H-01**

RESOLVED

Close redirects manager return

CATEGORY	Gaming
LOCATION	src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:returnGrossYieldAssets
REVIEW	Remediation Review 1

DESCRIPTION

`returnGrossYieldAssets` and `returnGrossYieldVaultShares` classify a manager return as orphan backing only when `totalSupply()` is zero at execution time. They do not bind the receipt to an expected epoch, checkpoint, or beneficiary cohort. `closeEpoch()` is permissionless and can settle a pending deposit batch, mint escrow shares, and move supply from zero to positive before the return lands.

An attacker who prepositions a valid pending deposit can order an eligible `closeEpoch()` immediately before an honest manager return. The close bootstraps supply for the attacker's cohort. The return then skips `_isolateOrphanBacking`, remains active surplus, and is recognized for that new cohort on a later close. Value that docs treat as manager recoverable orphan backing is instead captured by the bootstrap depositor (net of any protocol yield fee).

Example: supply is zero, attacker deposits 10 assets pending, manager is about to return 4 assets of residual. Close first mints escrow supply; the return no longer orphans; after the next close and claim, the attacker redeems the returned value instead of the manager recovering it.

```
``plain text
// returnGrossYieldAssets / returnGrossYieldVaultShares
if (totalSupply() == 0) _isolateOrphanBacking(currentEpoch, manager);
``
```

RECOMMENDATION

Attribute manager returns to an explicit cohort or checkpoint chosen at authorization time, not to execution time `totalSupply()`. Isolation versus active surplus must be invariant under reordering with permissionless `closeEpoch()`.

RESOLUTION

Status: Resolved.

HIGH **H-02**

ACKNOWLEDGED

One-atom donation lets a depositor seize almost all post-loss recovery

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:activeRawAssets():L285-L287 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:totalAssets():L300-L303 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_requestDeposit():L641-L692 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_closePendingBatch():L948-L990
REVIEW	Remediation Review 1

DESCRIPTION

After a live total impairment leaves legacy supply outstanding but before the loss is checkpointed, the intake circuit breaker depends on `totalAssets() == 0`. A public one-atom raw donation changes live backing to one, reopens intake, and lets a small request settle at the dust exchange rate. The newly minted dominant share position then participates in later recovery intended for the impaired cohort.

Impact: An unprivileged depositor can capture virtually all subsequent remediation or recovered backing while retaining its deposit principal, diluting legacy holders to a negligible fraction. The passing PoC uses the public request/close/claim lifecycle and production manager return/close flow and demonstrates over 99.999% capture of a recovery vastly larger than attacker capital.

RECOMMENDATION

Do not allow uncheckpointed donated backing to reopen intake for nonzero legacy supply after a live depletion. Require an explicit recapitalization/checkpoint transition before accepting a new cohort, and enforce a meaningful minimum viable rate or cohort-aware recovery allocation rather than an exact-zero predicate.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/f29b7243951b8aaaa6675b35493151f6>

RESOLUTION

Status: Acknowledged.

HIGH **H-03**

ACKNOWLEDGED

Idle-only full-exit guard lets a dust successor capture deployed pre-settlement surplus

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleDepositRequest():L182-L211 package s/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleRedeemRequest():L219-L252 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_fullExitResidualCaptureCheck():L409-L428 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_unreservedIdleAssets():L495-L508
REVIEW	Remediation Review 1

DESCRIPTION

For a same-target redemption of the complete incumbent supply, the deposit-settlement guard measures residual value only through unreserved idle assets.

Deployed backing and already-accrued surplus in asset-manager custody are invisible, so a single dust successor request can settle even while same-target redemption shares equal totalSupply.

The new mint prevents the incumbent burn from reaching zero; after the manager unwinds, the fixed redemption is paid but the surplus is never classified as orphaned and the next report attributes it to the dust successor.

The passing PoC holds the same 1,300 USDC in manager custody before both branches: moving it idle rejects the 1-USDC settlement due to the 100-USDC residual, while leaving it deployed permits the attacker ultimately to redeem about 101 USDC.

Impact: An unprivileged depositor can capture essentially all deployed positive realization surplus that the vault's idle-path policy would isolate as prior-cohort orphan value. The loss scales with the strategy surplus rather than attacker principal and requires no false report or malicious trusted-role action.

RECOMMENDATION

Treat a same-target redemption of the complete incumbent supply as a cohort boundary regardless of backing denomination. Prevent successor settlement until the incumbent burn reaches zero and prior-position returns are classified, or add explicit cohort/deployed-position accounting that attributes later returns correctly.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/28dbb97d5ba71ad62fb451624c397eaf>

RESOLUTION

Status: Acknowledged.

HIGH **H-04**

PARTIALLY RESOLVED

Epoch-close front-run vetoes depleted reset and captures delayed legacy proceeds

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_closePendingBatch():L948-L992 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:abandonDepletedShares():L1083-L1104 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_claimByAssets():L695-L725 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_isolateOrphanBacking():L1266-L1273
REVIEW	Remediation Review 1

— DESCRIPTION

A dust depositor can prepare a request and, once the epoch is closable, front-run the final depleted holder's abandonment with closeEpoch. Because the nonzero one-atom checkpoint passes the pending-batch depletion test, close mints a large accepted wrapper-share escrow to the vault. The holder's unchanged abandonment then burns only its own balance and does not reach zero supply or isolate orphan backing. Only the request controller can release the accepted escrow, which does not expire, so the attacker controls whether the vault can ever return to zero supply. Manager-returned delayed proceeds are consequently classified as active backing rather than zero-supply orphan backing and can be claimed through the attacker's escrow shares.

Impact: The revised passing PoC uses the same 5e11-atom manager-funded raw-asset return in both orderings. In the victim-first control, the manager recovers the complete return and the attacker receives no more than its 100-atom pending lot. In the close-first ordering, the attacker receives approximately 490.196 billion atoms above that control while the manager suffers the corresponding shortfall; alternatively, the attacker can leave the escrow unclaimed and indefinitely prevent orphan recovery. The captured value is unbounded relative to the dust request.

— RECOMMENDATION

Add final-supply slippage protection to abandonment, such as expectedTotalSupply and a requireFinalReset flag that reverts if the caller is no longer the complete supply. Align batch depletion classification with the zero-output abandonment criterion, and add expiry or forced resolution for accepted vault-held escrow so an inactive controller cannot indefinitely veto zero supply.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/de509b03e92885dac84fafc29a753726>

H-04 Epoch-close front-run vetoes depleted reset and captures delayed legacy proceeds **CONTINUED**

— **RESOLUTION**

Status: Partially resolved.

MEDIUM **M-01**

RESOLVED

One wei self-transfer kills orphan capture

CATEGORY Logical Error
LOCATION AsyncNavVaultCore.sol:479
REVIEW Remediation Review 1

DESCRIPTION

AsyncNavVault has no `_update` or transfer guard, so anyone can send shares to the vault itself. Those shares are unrecoverable: `sweepExternalRewardToken` rejects the share token (`AsyncNavVaultAssetManagement.sol:63`), and every payout path is bounded by a request's own recorded amounts. `totalSupply()` can therefore never return to zero again. `_captureOrphanedIdleAssetsIfZeroSupply` returns early unless `totalSupply()` is zero (`AsyncNavVaultCore.sol:479`), so one wei kills it permanently. `_fullExitResidualCaptureCheck` also takes its permissive `sameTargetRedeemShares < supply` branch (`AsyncNavVaultSettlement.sol:420`), and `_unreservedIdleAssets` (`AsyncNavVaultCore.sol:496`) then hands the prior cohort's residual to the next cohort as deployable liquidity. This permanently defeats a shipped remediation. `src/async-nav-vault/docs/SECURITY.md:114` states the residual "is atomically classified as `orphanedIdleAssets`". A later cohort cannot deploy or redeem that reserve." A passing proof of concept shows two otherwise identical vaults diverge by the whole residual after one 1-wei transfer.

RECOMMENDATION

Do not add a blanket `_update` revert on `to == address(this)`. `requestRedeem` escrows shares via `_transfer(owner, address(this), shares)` at `AsyncNavVaultRequests.sol:107` and `:125`, so that guard bricks every redemption request. Instead override the public `transfer` and `transferFrom` to revert when `to == address(this)`. Internal `_transfer` escrow and the settlement `_mint(address(this))` at `AsyncNavVaultSettlement.sol:200` stay untouched. Payout receivers are a separate vector, covered in a companion finding.

RESOLUTION

Status: Resolved.

MEDIUM **M-02**

ACKNOWLEDGED

Revoked auth re-arms at historical generation

CATEGORY Signatures
LOCATION MultiOwnerDeterministicValidator.sol:1418-1432
REVIEW Remediation Review 1

DESCRIPTION

Replayable multichain digests omit `chainId` (`REPLAYABLE_DOMAIN_TYPEHASH`, `GliderRootValidatorDigests.sol:9-10`) and the locator is address-only, so for a deterministic validator+account the ONLY chain separator is `authorizationGeneration` (`:1427`) — which is also the revocation mechanism. `_bumpAuthorizationGeneration:1048-1055` only ever adds one; no setter, no way to skip.

Not the documented fan-out gap (`DEPLOYMENT.md:48-55`): the PoC completes and confirms invalidation on every registered chain first. That check is point-in-time and cannot cover a chain onboarded later. Such a chain starts at 0, reaches 1 at `onInstall:177`, and must climb through every value used elsewhere — including revoked ones, since replayable use requires matching generations (`ASSUMPTIONS_AND_TRADEOFFS.md:18`).

While it rests on revoked N, a grant dead everywhere else validates and executes an arbitrary UserOp. PoC drains 10 ETH, and proves the digest is byte-identical on 8453 and 42161.

RECOMMENDATION

Let the counter skip: an owner-authorized entrypoint setting `authorizationGeneration` to any STRICTLY GREATER value, so a new chain jumps straight to the current global generation instead of climbing through revoked ones. Monotonicity is essential — a decreasing setter would re-arm any historical grant. Alternatively compare against an owner-signed revocation floor with `>=`, not `==`. Separately, consume `batchNonce`: unstored, so grants are unlimited-use per chain.

PROOF OF CONCEPT <https://gist.github.com/waffle-guardian/a6755fe75e50b955dcf33a032ea3d6ad>

RESOLUTION

Status: Acknowledged.

MEDIUM **M-03**

PARTIALLY RESOLVED

A live one-share dust request can indefinitely force whole epochs into recovery

CATEGORY Informational

LOCATION `packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol:_classifyPendingBatch():L294-L330 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_closePendingBatch():L948-L984 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_isAccepted():L1226-L1230`

REVIEW Remediation Review 1

— DESCRIPTION

A valid live dust request is accurately included in the pending aggregates and tracked as the smallest lot. If its pro-rata wrapper allocation floors to zero, the classifier records `REQUEST_ZERO` for the whole epoch, mints no escrow, and makes every unrelated request recoverable. The passing Base-fork PoC confirms the dust remains OPEN and active through close; it does not rely on cancellation or stale metadata.

Impact: With a microscopic high-rate active cohort and one recoverable one-share request per epoch, an untrusted holder can repeatedly prevent arbitrarily large honest deposits from onboarding. Depositors wait for close, receive no wrapper shares, must unwind their external-vault lots, incur extra gas and rounding exposure, and resubmit, enabling indefinite denial of the core async-deposit path.

— RECOMMENDATION

Isolate individual zero-allocation requests for recovery while accepting representable requests, or enforce a rate-aware minimum request size that remains valid through close. Cancellation-aware minimum tracking alone does not remediate this live-request path.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/50409c2bcf51932814b2d2110870adb0>

— RESOLUTION

Status: Partially resolved.

MEDIUM **M-04**

RESOLVED

NAV payouts allow receiver == vault

CATEGORY Validation
LOCATION AsyncNavVaultClaims.sol:239
REVIEW Remediation Review 1

DESCRIPTION

Every NAV path that pays out to a caller-supplied receiver rejects only `address(0)`. The asset transfers in `claimWithdraw` and `claimRedeem` are guarded at `AsyncNavVaultClaims.sol:239` and `:279`; `cancelDepositRequest` and `decreaseDepositRequest` share a single check at `AsyncNavVaultRequests.sol:319`, and `recoverZeroOutputDepositRequest` has its own at `:243`. A claim directed at the vault itself performs a self-transfer that satisfies the transfer while delivering nothing to the claimant.

The claimant is not necessarily the party choosing the receiver. `_requireControllerOperator` admits any approved ERC-7540 operator, and the operator supplies the `receiver` argument, so a third party acting for a controller can direct a payout into the vault.

The accounting still treats the claim as settled. The redemption is consumed and `pendingClaims` is decremented before the transfer, so `_unreservedIdleAssets` sees the same vault balance against a lower reservation and the assets become deployable. `deployToAssetManager` can then move them to the asset manager, converting a claimant's redemption into protocol-controlled funds. The sibling vault rejects `address(this)` on all eleven of its payout entrypoints at `AsyncYieldSplitVault.sol:1218`, so the guard exists elsewhere in the codebase.

RECOMMENDATION

Add a receiver helper that rejects `address(0)` and `address(this)`, mirroring `AsyncYieldSplitVault._requireReceiver` at `:1218`. Call it wherever a caller supplies a payout receiver: `claimWithdraw` and `claimRedeem` (`AsyncNavVaultClaims.sol:239, :279`), the shared `_decreaseDepositRequest` check (`AsyncNavVaultRequests.sol:319`) reached by `cancelDepositRequest` and `decreaseDepositRequest`, and `recoverZeroOutputDepositRequest` (`:243`). The share-mint paths at `:158` and `:199` need it too.

RESOLUTION

Status: Resolved.

MEDIUM **M-05**

PARTIALLY RESOLVED

Lazy redemption settlement grants a one-sided option on post-target strategy returns

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_unreservedIdleAssets():L495-L509 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleRedeemRequest():L219-L255 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultRequests.sol:_decreaseRedeemRequest():L352-L385
REVIEW	Remediation Review 1

— DESCRIPTION

A redemption is not irrevocably crystallized when its fixed N+1 price is reported. Its escrowed shares remain in totalSupply throughout N+1, and the fixed asset liability is created only if the controller initiates a claim. The controller can observe post-report performance and select the better denomination: after a loss it claims fixed N+1 assets; after a gain it waits for N+2 and recovers live appreciating shares through expiry cancellation. This remains reachable when the trusted manager protects the full fixed claim before N+1 and deploys only the residual cohort's backing.

Impact: An existing shareholder obtains downside-protected exit rights without surrendering upside, shifting the option cost to the residual shareholder cohort. In the compliant-manager PoC, a 30-USDC loss lets the requester take its protected 60 while the victim is left with 10. If the deployed lot instead earns 20 USDC, irrevocable exit would leave the residual victim with 60, but cancellation returns live shares worth 72 to the requester and reduces the victim to 48, capturing 12 of the post-fixing gain.

— RECOMMENDATION

Independently crystallize target-epoch redemption requests at the N+1 report transition: reserve their fixed asset liabilities and burn or economically segregate escrowed shares. Expiry recovery must return the already-fixed asset claim or an equivalent fixed-value entitlement, not live shares. Deposit-side remediation alone is insufficient because redemption uses separate request, reserve, burn, and cancellation accounting.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/045219b45fe639d219bc5017624d22ce>

— RESOLUTION

Status: Partially resolved.

MEDIUM **M-06**

ACKNOWLEDGED

Remaining share atoms let a later cohort steal prior residuals

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/lib/openzeppelin-contracts/contracts/token/ERC20/ERC20.sol:transfer():L99-L103 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_fullExitResidualCaptureCheck():L409-L428 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_captureOrphanedIdleAssetsIfZeroSupply():L478-L490
REVIEW	Remediation Review 1

— DESCRIPTION

Because both the same-target full-exit residual guard and zero-supply orphan classifier use raw totalSupply, one outstanding wei makes the redemption of all externally owned incumbent shares appear non-final. After an honest target snapshot, a strategy can realize excess value before the incumbent's fixed-price claim; the atom prevents the resulting residual from being classified.

In the case where all outstanding shares would have been redeemed and a residual would have been created and classified as an orphan, a single wei can be left behind, unclaimed, by a redeemer which would prevent the "orphan" case from being triggered.

A small later deposit can therefore enter in that same epoch at the residual-excluding price and capture nearly all of that residual when it is subsequently reported.

— RECOMMENDATION

Consider requiring that a redemption must not leave smaller than a certain amount of shares in the vault or in the specific account. A similar threshold would also have to be added for deposits.

Otherwise consider if it is acceptable that the residual amounts can be claimed by either new depositors in these edge scenarios or by those who remain in the vault after a large portion of depositors have left.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/5924e82fd8348318eebabfd7938a6d20>

— RESOLUTION

Status: Acknowledged.

MEDIUM **M-07**

PARTIALLY RESOLVED

Externally backed pending claims mask full-exit residuals from the new-cohort safeguard

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleDepositRequest():L182-L211 package s/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_fullExitResidualCaptureCheck():L409-L428 package s/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_unreservedIdleAssets():L495-L508 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_captureOrphanedIdleAssetsIfZeroSupply():L478-L490
REVIEW	Remediation Review 1

— DESCRIPTION

A crystallized redemption can reserve fungible idle while its economically corresponding assets remain deployed. The deposit full-exit guard nevertheless subtracts the entire pendingClaims balance from idle. An outstanding claim can therefore mask a post-report residual of up to that balance, allowing a target-epoch deposit to settle.

Its newly minted supply then prevents the incumbent's complete redemption from reaching zero supply and triggering orphan capture.

After the old claim is paid and its deployed backing returns, the masked residual becomes backing for the depositor.

This creates a scenario in which a deposit can claim the residual amount which otherwise should have been orphaned if the order of execution was different.

— RECOMMENDATION

Do not use aggregate pendingClaims as evidence that the claim's economic backing is already idle in the full-exit residual check.

Alternatively, some edge cases such as this may be accepted or require a larger refactor.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/b66ed2ce5429c08b0e5188a887cf0667>

— RESOLUTION

Status: Partially resolved.

MEDIUM **M-08**

RESOLVED

Shortage-side full-exit gate lets stale redeemers seize an entire successor deposit

CATEGORY	Informational
LOCATION	<code>packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_fullExitResidualCaptureCheck():L409-L428</code> <code>packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleDepositRequest():L182-L210</code> <code>package s/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:_consumeDepositClaim():L309-L330</code> <code>packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleRedeemRequest():L219-L252</code> <code>packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_unreservedIdleAssets():L495-L508</code>
REVIEW	Remediation Review 1

DESCRIPTION

When all incumbent shares target the same epoch redemption, the full-exit check rejects successor deposit settlement only for excess backing and allows every liquidity shortage.

A successor deposit is initially reserved, so the underfunded incumbent cannot redeem. Once the successor claims shares, however, claim consumption removes its assets from deposit reserves without proving the incumbent liability is funded from incumbent backing. The newly unreserved principal can immediately satisfy the stale incumbent redemption.

The production-contract PoC posts a 1,000-USDC NAV, models a post-report loss leaving 100 USDC, proves the old redemption is blocked, then shows a 900-USDC successor claim makes it executable and is wholly paid to the incumbent.

Impact: Up to 100% of an ordinary successor depositor's principal can be transferred to stale-price exiting incumbents after a post-snapshot loss. The successor receives nonzero shares with zero backing, and the nonzero-price reporting rule prevents a truthful zero-value next snapshot. The loss scales with the full redemption deficit and victim deposit rather than rounding dust.

RECOMMENDATION

Block successor deposit settlement or reservation release whenever same-target redemptions consume all incumbent supply and their complete liability is not already funded without successor request assets. Prefer settling the incumbent full exit through zero-supply classification before successor shares can be claimed, and enforce that releasing deposit reserves cannot increase executable redemption liquidity for the exiting cohort.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/1a121be4a9f6bbc529f740939ad84ffc>

RESOLUTION

§04 - REMEDIATION REVIEW 1 · M-08
Status: Resolved.

CONFIDENTIAL P. 110 / 140

MEDIUM **M-09**

RESOLVED

Ceil-valid floor-priced reports let a full-supply holder indefinitely veto deposits

CATEGORY Informational

LOCATION `packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_convertToAssetsAtPrice():L466-L473` `packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_requireConsistentReport():L628-L635` `packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleDepositRequest():L182-L202` `packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_fullExitResidualCaptureCheck():L409-L428`

REVIEW Remediation Review 1

— DESCRIPTION

The report consistency check accepts either the floor or ceil product of supply and WAD price. For a non-integral floor-derived price, an honest report can therefore attest N atoms of backing while full-supply redemption liability floors to $N-1$. A sole holder can escrow all shares for that target epoch and deliberately leave the redemption unsettled; the full-exit residual guard then treats the one-atom difference as capturable excess and rejects every unrelated deposit settlement. At expiry the holder cancels, recovers every share, and recreates the condition for the next epoch.

Impact: A cheap bootstrap holder controlling the complete supply can indefinitely prevent all later deposit cohorts from entering while preserving its full position. Each targeted cohort's entire pending principal remains immobilized until $N+2$ cancellation, and the attack can be repeated without incremental asset loss.

— RECOMMENDATION

Use a canonical rounding convention across report consistency, full-redemption valuation, and residual classification. For example, require `shareBackedNav` to equal the floor conversion used for redemption or tolerate only the report-proven ceil-minus-floor atom in the residual guard. Alternatively, make full-exit crystallization permissionless before allowing a pending redemption to block unrelated deposit settlement.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/d55851222a170788c3d94aa317c763be>

— RESOLUTION

Status: Resolved.

MEDIUM **M-10**

PARTIALLY RESOLVED

Bundled deposits replace impaired VaultV2 shares at a transient stale NAV

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultIntegration.sol:_morphoDepositCapacity():L98-L135 packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol:intake():L81-L127 packages/onchain-base/src/__onchain/vaultv2-beef0e0834849acc/src/vault-v2/src/VaultV2.sol:accrueInterestView():L652-L660
REVIEW	Remediation Review 1

— DESCRIPTION

VaultV2's first accrual pins `firstTotalAssets` for the transaction, so later accruals do not reread adapter assets. An attacker can pin pre-loss NAV, realize Morpho bad debt, and place a victim Kernel deposit later in the same EntryPoint bundle. Glider observes `expectedSupplyAssets < marketAllocation`, but treats the negative drift as zero rather than closing capacity; its before/after previews also use the pinned rate. A final attacker operation redeems impaired old VaultV2 shares using the victim's fresh liquidity. The updated PoC passes through canonical EntryPoint v0.7 with three signed operations and real Kernel accounts. Impact: An external VaultV2 holder with no Glider shares can substitute impaired shares for fresh Glider principal. In the PoC, a \$10m request retains about \$1.94m of real value while the attacker extracts \$10m. The loss can approach the full incoming deposit and can target multiple bundled requests.

— RECOMMENDATION

Return zero capacity or revert whenever expected market assets are below recorded allocation. Apply the negative-drift guard to MORPHO_V2 intake and close-time compounding/settlement, and do not rely on previews when `firstTotalAssets` may predate a realized adapter loss.

— RESOLUTION

Status: Partially resolved.

LOW **L-01**

PARTIALLY RESOLVED

Aave exit residual can drift by one atom

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/main/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol#L1157
REVIEW	Remediation Review 1

DESCRIPTION

The current release does not certify an Aave yield vault, and the production deployment script rejects targets outside the three qualified Base tuples. Aave is therefore not an affected production target. This behavior is relevant if an Aave backed ERC-4626 vault is considered for future qualification. For a partial exit with no idle assets, the planner can allocate the entire recognized budget to the underlying redemption:

```
``plain text
uint256 bestValue = Math.min(Math.min(highStandalone, highMarginal), underlyingBudget);
...
plan.previewedUnderlyingOutput = bestValue;
plan.assetsOut = plan.rawPaid + bestValue;
```

Execution then requires the remaining live value to equal or exceed the exact preview derived residual. A fork test using a local ERC-4626 wrapper over the real Arbitrum Aave pool observed a planned residual of 500012451465260393 and an execution residual of 500012451465260392. The resulting one atom deficit caused `ExitConservationFailure`. The integration matrix also observed this revert for its synthetic Arbitrum USDC and Mainnet WETH Aave wrappers, while other tested Aave cases passed.

The transaction reverts atomically and does not lose principal. A different exit amount may avoid the rounding boundary, and the final supply exit skips the remaining holder conservation check. The evidence therefore establishes a future integration compatibility constraint, not a vulnerability in the currently certified target set.

RECOMMENDATION

Keep Aave targets unsupported unless the exact proposed vault passes the complete qualification process. If Aave support is required, ensure partial exit planning tolerates permitted preview versus execution residual rounding without weakening conservation for remaining holders.

L-01 Aave exit residual can drift by one atom

CONTINUED

PROOF OF CONCEPT <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/commit/a44e4eb3a2f7d50251ae3cfa4d89125e97125889>

— RESOLUTION

Status: Partially resolved.

LOW **L-02**

RESOLVED

Authoritative liquidity blocks exit

CATEGORY	DoS
LOCATION	src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol:_executeExit()
REVIEW	Remediation Review 1

DESCRIPTION

`_executeExit` hard gates on the integration liquidity reading whenever it is marked authoritative:

```

`solidity
(uint256 reportedLiquidShares, bool authoritative) = _integrationLiquidUnderlyingShares();
uint256 liquidShares = Math.min(snapshot.underlyingShares, reportedLiquidShares);
if (authoritative && plan.selectedUnderlyingShares > liquidShares) {
    revert InsufficientActiveLiquidity(plan.selectedUnderlyingShares, liquidShares);
}
`

```

That gate runs before share burn and `yieldVault.redeem`. Later exact share debit, measured receipt, reserve, and conservation checks never run, even though they would safely reject a truly insolvent redeem. Both certified integration modes can under report an executable share lot by about one share, so a boundary `withdraw` or full remaining `redeem` reverts with `InsufficientActiveLiquidity(N+1, N)` (or equivalent) despite a successful same state underlying `redeem`.

****Standard mode (Moonwell Frontier cbBTC).**** `liquidShares` treats a successful `yieldVault.maxRedeem(wrapper)` as authoritative. The certified MetaMorpho vault `0x543257eF2161176D7C8cD90BA65C2d4CaEF5a796` can under report by one share relative to an executable `redeem`. At pinned Base block `49,290,584`, `maxRedeem` returned `N` while `redeem(N+1, ...)` succeeded and paid the next asset atom. On the `IntegrationFork` path, a half `withdraw` of `500,000` cbBTC atoms succeeds, but redeeming the remaining wrapper balance fails with `InsufficientActiveLiquidity(4141408629398389, 4141408629398388)`. Capping the second exit to published `maxRedeem` leaves one residual wrapper share.

****Morpho V2 mode (Steakhouse class).**** `_morphoLiquidShares` measures idle (and optionally adapter) liquid assets, then floor converts with `convertToShares`. When the reading is authoritative (valid egress binding, and either zero adapter liquidity or positive allocations on all three cap ids), the same hard gate applies. Example: wrapper holds all `3,000` underlying shares, managed `totalAssets = 5,000`, idle liquidity `4,999`, zero adapter liquidity. The planner selects `_pinv(4,999) = 3,000` shares while `convertToShares(4,999) = 2,999`, so full exit reverts with `InsufficientActiveLiquidity(3000, 2999)` even though

L-02 Authoritative liquidity blocks exit

CONTINUED

`redeem(3000, ...)` pays `4,999`. Reproduced in `AsyncYieldSplitVaultL05MorphoGateUnderReport.t.sol`. This can affect certified Steakhouse Prime ETH/USDC wrappers whenever their Morpho liquidity reading is authoritative. Impact in both modes is withdrawal liveness only: no funds move, custody and allowances are unchanged, and a slightly smaller or `maxRedeem` capped exit usually succeeds, possibly leaving residual wrapper dust.

— RECOMMENDATION

Do not treat an authoritative liquidity hint as a hard ceiling when the subsequent guarded `yieldVault.redeem` already enforces exact share debit, asset receipt, reserve coverage, and remaining holder conservation. Liquidity views may stay conservative, but execution should not reject an exit planner selected lot solely because Standard `maxRedeem` or Morpho `convertToShares` under reported by a rounding atom.

PROOF OF CONCEPT <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/pull/2/changes>

— RESOLUTION

Status: Resolved.

LOW **L-03**

RESOLVED

Donation blocks same epoch deposits

CATEGORY	DoS
LOCATION	AsyncNavVaultSettlement.sol
REVIEW	Remediation Review 1

— DESCRIPTION

When redemptions targeting an epoch cover the full share supply, deposit settlement rejects any unreserved idle assets above the floor-valued redemption liability:

```
```plain text
uint256 sameTargetRedeemShares = _pendingRedeemSharesByTargetEpoch[settlementEpoch];
if (sameTargetRedeemShares < supply) return (true, 0);

uint256 sameTargetRedeemAssets =
_convertToAssetsAtPrice(sameTargetRedeemShares, settlementPrice, Math.Rounding.Floor);
if (available > sameTargetRedeemAssets) return (false, available - sameTargetRedeemAssets);
``
```

Anyone can increase `available` by transferring one underlying atom directly to the vault. A final shareholder can then withhold its redemption claim throughout the target epoch, causing every same-target deposit claim to revert. After the next report, depositors must cancel and requeue. The shareholder can cancel its own expired redemption and repeat the denial with another full-supply request. No principal is taken, but a controller with the final share supply can repeatedly delay otherwise valid deposits at negligible direct cost.

### — RECOMMENDATION

Deposit settlement should not depend on an untrusted token balance exceeding the calculated redemption liability. Preserve prior-cohort residual isolation without allowing unsolicited transfers and an unclaimed redemption to block the next cohort.

### — RESOLUTION

Status: Resolved.

LOW **L-04**

RESOLVED

## Dust redeem blocks funded cohort

CATEGORY	DoS
LOCATION	src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleRedeemRequest()
REVIEW	Remediation Review 1

### — DESCRIPTION

Redemption settlement calculates the caller's payout with a per-request floor, but checks available liquidity by floor-converting all shares pending for the target epoch at once. These operations are not additive. A dust request can have a zero individual payout while still increasing the aggregate requirement by one asset atom:

```
```plain text
```

```
assets = _convertToAssetsAtPrice(shares, settlementPrice, Math.Rounding.Floor);  
if (assets == 0) revert AsyncNavVaultZeroSettlement();
```

```
uint256 aggregateAssets = _convertToAssetsAtPrice(  
    _pendingRedeemSharesByTargetEpoch[settlementEpoch], settlementPrice, Math.Rounding.Floor  
);
```

```
``
```

For example, at a `0.6e18`` price, requests for one and four share atoms pay zero and two assets individually, but the aggregate five shares require three assets. If two assets are available, the funded request reverts until the dust controller removes its request or the epoch expires. The victim then has to cancel and requeue at a later NAV.

— RECOMMENDATION

The shared liquidity requirement should equal the sum of executable request liabilities, or zero-output requests should not be allowed to increase the requirement imposed on unrelated controllers.

— RESOLUTION

Status: Resolved.

LOW **L-05**

RESOLVED

Raw-zero orphan classification confiscates yield from accepted but unminted deposit equity

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:_captureOrphanedIdleAssetsIfZeroSupply():L475-L491 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleDepositRequest():L177-L212 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol:_settleRedeemRequest():L214-L255 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultAssetManagement.sol:sweepOrphanedIdleAssets():L35-L48
REVIEW	Remediation Review 1

DESCRIPTION

After the target-epoch NAV report, a deposit request has a fixed-price share entitlement but its shares do not exist until the controller claims. In the ordinary path, the controller claims during the target epoch, becomes a shareholder, and participates pro rata in any gain or loss that occurs after the report. The orphan path produces a different outcome solely because the shares remain unminted. If all incumbent shares are redeemed, a post-report gain can make the residual guard reject the accepted depositor's claim. The incumbents can then finish redeeming and reduce raw totalSupply to zero. The orphan classifier reserves the depositor's principal but treats the post-report gain as ownerless and allows the manager to sweep it. The depositor may later mint the same fixed-price shares, but those shares no longer have exposure to that gain.

Impact: Whether the accepted depositor receives post-report gains depends on whether they successfully materialize shares before the zero-supply transition, rather than on a different contribution or investment decision. If the product intends accepted deposits to begin exposure only once claimed, this is intended cohort policy; if it intends the fixed-price entitlement to carry target-epoch exposure, the gain can be permanently excluded from that cohort.

RECOMMENDATION

Define economic emptiness to include viable target-epoch deposit equity. Materialize accepted deposit supply before a final redemption burn, or track virtual claimable shares/value and prevent orphan capture while such equity exists. Align the full-exit residual guard and orphan reserve calculation to that same economic-supply model.

RESOLUTION

Status: Resolved.

LOW **L-06**

PARTIALLY RESOLVED

Exact withdrawal can burn the full supply while paying only a partial target

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultSettlement.sol:_withdrawQuote():L245-L299 package s/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/contrac ts/AsyncYieldSplitVaultSettlement.sol:_exactWithdrawPlanFromQuote():L308-L354 pa ckages/glider-contracts-audit-team1-1779896254839/src/async-yield-split-vault/co ntracts/AsyncYieldSplitVault.sol:_executeExit():L1107-L1160
REVIEW	Remediation Review 1

— DESCRIPTION

When an exact-asset target is above the greatest budget representable by a nonfinal wrapper-share count but below the full-supply redeem quote, `_withdrawQuote` returns the entire supply. The exact-withdraw plan still sets `assetsOut` to the smaller requested target. Execution consequently debits the full checkpoint and burns all shares, transfers only the target, and isolates the unpaid backing as manager-recoverable orphan value. `maxWithdraw` can advertise such a target during partial underlying liquidity.

Impact: The last shareholder can permanently forfeit nearly all recognized value by following the standard `withdraw(maxWithdraw(owner))` pattern. Once all shares are burned, restored liquidity does not restore the user's claim, and the unpaid backing is recoverable only by the manager.

— RECOMMENDATION

Do not allow an exact withdrawal below the full-supply redeem quote to consume the full supply. Revert or return a burn above the owner's balance for an unrepresentable partial target, or define a terminal rule that pays the full entitlement. Ensure `maxWithdraw` never advertises a partial target whose preview burn is the complete supply.

PROOF OF CONCEPT <https://gist.github.com/GuardianAudits/cbc3dea061ad9f834c2b3fe456ac3fcc>

— RESOLUTION

Status: Partially resolved.

INFORMATIONAL I-01

ACKNOWLEDGED

Operator-executed async deposit claims are logged as controller calls

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:claimDeposit():L149-L180 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:claimMint():L190-L220
REVIEW	Remediation Review 1

DESCRIPTION

The deposit-claim functions authorize either the controller or its approved ERC-7540 operator, but emit the ERC-4626 Deposit event with controller as its indexed sender. DepositRequestClaimed also omits the executing caller. Thus, when an operator claims, the vault's own logs do not identify the actual caller. The PoC performs a real settled request and operator-authorized claim and explicitly decodes the incorrect Deposit sender.

Impact: Event-only consumers cannot distinguish a direct controller claim from an operator-executed claim, impairing delegated-action auditing and potentially misleading integrations that interpret the canonical Deposit sender as the caller. The executor remains recoverable from transaction-level data, and no unauthorized state transition or direct fund loss is shown.

RECOMMENDATION

Emit Deposit(msg.sender, receiver, assets, shares) in both claim paths, while retaining controller in DepositRequestClaimed or extending the custom event to include the executing caller.

RESOLUTION

Status: Acknowledged.

INFORMATIONAL I-02

PARTIALLY RESOLVED

Ceil rounding makes advertised partial withdrawal amounts unexecutable

CATEGORY	Informational
LOCATION	packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:maxWithdraw():L99-L103 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:claimWithdraw():L251-L256 packages/glider-contracts-audit-team1-1779896254839/src/async-nav-vault/contracts/modules/AsyncNavVaultClaims.sol:_requireNoOneSidedClaimResidual():L367-L374
REVIEW	Remediation Review 1

DESCRIPTION

maxWithdraw() advertises the complete asset entitlement without accounting for a non-monotonic partial-withdraw domain. claimWithdraw() ceil-rounds notional share consumption. When one claimable share atom represents 1e6 asset atoms, every positive partial withdrawal consumes that sole share, leaves assets without shares, and reverts in _requireNoOneSidedClaimResidual. The passing production-path PoC proves maxWithdraw reports 1 USDC, withdraw of 0.5 USDC reverts, and the full 1 USDC withdrawal succeeds without intervening state change.

Impact: Exact-asset routers cannot assume amounts at or below maxWithdraw are executable. The blocked withdrawal interval scales with the assets represented by the final share atom and can encompass every partial exact-asset output. This is strictly higher impact than the same-root-cause mint-side issue, whose unspittable remainder is worth less than one asset atom. Principal remains recoverable through a full claim, so severity is Low.

RECOMMENDATION

Preserve the final notional share until the terminal asset claim, retain higher-precision claim accounting, or expose/document executable partial bounds. Keep request/controller/global liabilities paired and add invariants that every positive amount through maxWithdraw executes in unchanged state.

RESOLUTION

Status: Partially resolved.

MEDIUM **M-01****Close prices Morpho pin without live NAV**

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVault.sol#L1041-L1045 https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L356-L357 (https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L356-L357)
REVIEW	Remediation Review 2

DESCRIPTION

F-01 added `morphoNavConformance` after Morpho deposits on intake (hard revert) and optional close compounding (soft fail). `closeEpoch` still prices active yield, the manager fee, the next checkpoint, and pending escrow from `yieldVault.previewRedeem` with no live NAV check, even while the wrapper still has control.

Vault V2 can pin `totalAssets()` / `previewRedeem` for the rest of a transaction after the first state-changing `accrueInterest`. A later Morpho Blue loss can drop adapter `realAssets()` without updating that pin. Intake then correctly advertises zero capacity. Close continues, treats the pinned preview as live, and writes it into accounting.

```
``plain text
```

```
_attemptOptionalCompounding();
livePreFeeActiveAssets = _activeLiveAssets();
grossYield = livePreFeeActiveAssets > previousCheckpoint
? livePreFeeActiveAssets - previousCheckpoint
: 0;
targetFeeAssets = Math.mulDiv(grossYield, closingFeeBps, BPS);
```

`_activeLiveAssets` adds raw idle to `previewRedeem(activeUnderlyingShares)`. `quotePendingBatch` uses the same pinned preview for pending mint. This is not the documented $F \setminus *$ residual (Morpho state changing after the wrapper returns).

Illustration: checkpoint 50, 100 raw, 100 Morpho shares whose live adapter value is now 0, pin still showing 90. Close treats live as 190, reserves a 10% fee of 14 from real raw, and can mint pending escrow against that same pinned 90. Those escrow shares later claim remaining raw after the next transaction dumps to live NAV. Team B M-09 already required a settlement guard for this pin, not only an intake/compounding guard.

RECOMMENDATION

M-01 Close prices Morpho pin without live NAV

CONTINUED

When Morpho mode is set, do not take `previewRedeem` as live for close fees, checkpoint, or pending mint if `morphoNavConformance` fails. Price those steps from the independent live sum, or freeze the pending batch into recovery. Keep permissionless close live if that remains a product constraint.

— RESOLUTION

Status: Reported during the fix review period.

MEDIUM **M-02**

Pending redeem shares earn yield on idle reserve

CATEGORY	Logical Error
LOCATION	src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol:deploymentStatus():326-337
REVIEW	Remediation Review 2

— DESCRIPTION

The main review raised “Redeem liquidity is not reserved” (`6a1cb0faeff7ed35bbe080fc`). At base `1ccfa80b`, deployment used `_unreservedIdleAssets()`, which reserved only `pendingClaims + pendingDepositAssets + claimableDepositAssets + orphanedIdleAssets` (`AsyncNavVaultCore.sol:495-509`); pending redeems reserved nothing. This round added `deploymentStatus()` / `fundingTarget` and now withholds `ceil(pendingRedeemShares * pricePerShare())` from deployable idle at every `pushSnapshot` (`AsyncNavVaultCore.sol:326-337`, `:312-323`).

The withheld assets earn nothing, but the requester’s escrowed shares keep accruing. `requestRedeem` escrows shares (`AsyncNavVaultRequests.sol:107,124`) and they are burned only at first settlement (`AsyncNavVaultSettlement.sol:220`), so until then the request is active equity collecting a full pro-rata share of yield produced entirely by other holders’ capital. Nothing bounds the window: requests never expire (`AsyncNavVaultSettlement.sol:392`) and `cancelRedeemRequest` returns all escrowed shares with no fee, cooldown or epoch gate (`AsyncNavVaultRequests.sol:219-224` -> `_decreaseRedeemRequest:329-359`). Measured at `7b4bfd59` (`src/async-nav-vault/test/guardian/pocs/PoC_HX_L243_RealisticUtilizationSuppression.t.sol`, 3/3 passing): with 10% of supply parked, other holders end 1,092.48 USDC (11 bps) below the no-request control while the requester gives up 121.39 — about 9x leverage in the requester’s favour. At 20%: others -1,941.22 (23 bps), requester -485.30. Over 30 reports producing 55,644.08 USDC of yield, the requester collects 5,564.41 (999 bps of all yield) having funded none of it.

The vault stays solvent and per-share returns stay identical, which is why no accounting check flags it. It needs no intent — a holder who forgets to settle collects the same subsidy.

— RECOMMENDATION

Stop escrowed shares accruing from epoch N+1, the point at which their backing stops being deployed, so the reserve and the accrual start and stop together. Two constraints on that fix: the checkpoint must stop accrual with the share count preserved, not freeze the request’s asset value at the N+1 price while cancellation stays free — that would be a free

M-02 Pending redeem shares earn yield on idle reserve

CONTINUED

option on NAV, cancel when it rises and settle at the stale price when it falls; and the non-accruing escrowed shares must be excluded from the supply used for `pricePerShare`, which also moves the reporter's `supply * price` consistency band.

If that is too invasive, bound the window instead: scope the holdback to the claimable window, or allow permissionless settlement after a published number of epochs. A cancellation fee prices the behaviour rather than removing it and does not help the forgetful case.

PROOF OF CONCEPT https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/3a04057fc505e9a147e4bb50d843b10ef03db19f/src/async-nav-vault/test/guardian/pocs/PoC_HX_L243_RealisticUtilizationSuppression.t.sol

— RESOLUTION

Status: Reported during the fix review period.

LOW **L-01**

Zero-tolerance live check skips compounding

CATEGORY Rounding

LOCATION <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L240> <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L109-L162> (https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L240 https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultLifecycle.sol#L109-L162)

REVIEW Remediation Review 2

DESCRIPTION

`compoundActiveIdle` deposits idle active raw assets into the underlying vault during `closeEpoch`, then requires the active live value to be non-decreasing:

```
``plain text
```

```
uint256 activeLiveAfter = _activeLiveAssets(yieldVault, activeRawAssetsAfter,
activeUnderlyingSharesAfter);
if (activeLiveAfter < activeLiveBefore) revert IntegrationConformanceFailure();
```

This is an exact comparison. Any ERC-4626 deposit mints `floor(assets * S / A)` shares and `previewRedeem` of those shares rounds down again, so converting `d` raw assets into shares is worth `d - 1` on roughly half of all inputs. The check therefore fails for ordinary, non-adversarial amounts on conforming vaults. The mandatory `intake` path already recognises this and applies `cappedIntakeLossTolerance(previewRedeem(1), assets)`; the compounding path does not, so the two custody paths disagree about whether a 1-wei rounding loss is a conformance failure. `_attemptOptionalCompounding` swallows `IntegrationConformanceFailure` and lets the epoch close, so the effect is that the deposit is rolled back and the raw balance stays idle until the next epoch's single retry, which again has the same ~50% chance of hitting the rounding edge. No event or return flag exposes the skip.

Fork probe on Base, 40 different manager `returnGrossYieldAssets` amounts each followed by one `closeEpoch`, with no other actor present:

- Steakhouse Prime ETH (Morpho V2): compounding skipped 15/40; each skip had `activeLiveAfter == activeLiveBefore - 1` and `morphoNavConformance` still true.

- Aave stata USDC: skipped 6/40.

- Moonwell Frontier cbBTC: skipped 0/40.

Impact is yield drag: manager-returned yield and other active idle raw balances sit uninvested for one or more extra one-day epochs on a random basis, and there is no in-epoch retry entryptpoint. It is not a stuck state and does not misprice shares. This is a different root cause from the existing "Momentary venue-cap exhaustion consumes the epoch's only compounding opportunity" finding (attacker-driven zero capacity read): here the wrapper's own check rejects a correctly executed deposit

L-01 Zero-tolerance live check skips compounding

CONTINUED

— RECOMMENDATION

Use the same tolerance model as `intake`: allow `activeLiveBefore - activeLiveAfter` up to `cappedIntakeLossTolerance(yieldVault.previewRedeem(1), depositedAssets)` before treating the drop as `IntegrationConformanceFailure`. Alternatively compare `previewRedeem(receivedUnderlyingShares)` against `depositedAssets` with a one-share-quantum allowance. Consider emitting an event when optional compounding is skipped so operators can see idle raw carrying across epochs.

— RESOLUTION

Status: Reported during the fix review period.

LOW L-02

Pending Morpho burns do not stop intake

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-yield-split-vault/contracts/AsyncYieldSplitVaultIntegration.sol#L204-L238>

REVIEW Remediation Review 2

DESCRIPTION

MORPHO_V2 capacity checks the current adapter topology, cap headroom, and live NAV. It never reads the canonical adapter's pending timelock map. A curator can `submit` `burnShares(marketId)` while Glider still advertises capacity and accepts `requestDeposit`.

Vault V2 then allocates the whole deposit into `liquidityAdapter` plus `liquidityData`. If that market is the one queued for burn and its caps still have headroom, the adapter's `supplyShares[marketId]` rises by the new mint. Executing the already scheduled burn sets that entire tracked balance to zero, including shares bought with the new deposit. Other markets keep pricing leftover NAV, so this is a write off of the allocated book, not a guaranteed total loss on a multi market Steakhouse vault.

```
``plain text
interface IMorphoMarketV1AdapterV2Integration {
// ...
function realAssets() external view returns (uint256);
}
`
```

The intake fingerprint stops at `realAssets()` and `_morphoLiveNav`. It has no `executableAt` read, so a publicly announced burn looks healthy until it executes. Morpho recommends zeroing the market caps before submitting a burn and does not enforce it. The curator must leave those caps and the liquidity route usable. A sentinel or curator can still `revoke` before the timelock expires, and an open Glider request can still `cancelDepositRequest` until `closeEpoch`.

RECOMMENDATION

Return zero Morpho intake capacity whenever `burnShares` is pending for the configured liquidity market or any market included in the adapter's live NAV.

RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-01

Redeem claim invalidates prepared report

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol#L674-L680 https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol#L204-L220 (https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-nav-vault/contracts/modules/AsyncNavVaultCore.sol#L674-L680 https://github.com/GuardianOrg/glider-contracts-audit-team1-1779896254839/blob/7b4bfd5947282ab302037f8b49ce3fe498b2acbd/src/async-nav-vault/contracts/modules/AsyncNavVaultSettlement.sol#L204-L220)
REVIEW	Remediation Review 2

DESCRIPTION

`pushSnapshot` takes a calldata `shareBackedNav` and checks it against live `totalSupply()` at inclusion. The report does not bind that NAV to the supply used when it was prepared. A first `claimRedeem` or `claimWithdraw` on an already eligible, funded request still burns the whole escrow in `_settleRedeemRequest` before the payout slice is sent:

```

`plain text
pendingClaims += assets;
_burn(address(this), shares);
`

```

Deposit collection no longer mints, so that half of remediations-v1 I-01 is gone. Same-epoch `requestRedeem` cannot do this; the request is not claimable until this report lands. An underfunded claim reverts and leaves supply unchanged.

If supply is 100 and the reporter submits `shareBackedNav` for 100, then Carol settles 40 shares first, the band becomes the value of 60 shares and `NavOracleInconsistentReport` rolls back the whole boundary. She is taking her own exit at the current price. The reporter recomputes `shareBackedNav` from the new supply and resubmits. Staging several eligible requests can force one retry each.

RECOMMENDATION

Treat live `totalSupply()` as an inclusion input: recompute `shareBackedNav` immediately before the report, and consider deriving the supply band onchain so stale calldata cannot revert the epoch.

RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-02

Misleading escrow claim in reserve check

CATEGORY	Informational
LOCATION	AsyncYieldSplitVault.sol:1438-1450
REVIEW	Remediation Review 2

— DESCRIPTION

The NatSpec of `_assertReserveCoverage` says it covers every pending, fee, orphan, and escrow reserve. The wrapper-share check only compares `balanceOf(address(this))` against `totalReservedProtocolWrapperShares`, so escrow shares owed to depositors are never included and the check can never revert. This is confusing for readers and reviewers; no shortfall is reachable.

— RECOMMENDATION

Either include outstanding escrow shares in the wrapper-share check, or remove "escrow" from the NatSpec.

— RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-03

Unused imports and errors

CATEGORY	Superfluous Code
LOCATION	AsyncNavVaultErrors.sol:10
REVIEW	Remediation Review 2

DESCRIPTION

Declared errors with no revert site anywhere in scope. Four of the five had one at base `1ccfa80b` and lost it in this diff.

- `AsyncNavVaultErrors.sol:10` `AsyncNavVaultDefaultAdminNotAuthorized`, already dead at base.
- `AsyncNavVaultErrors.sol:46` `AsyncNavVaultEmptyVaultIdleAssets`, orphaned by this diff.
- `AsyncNavVaultErrors.sol:56` `AsyncNavVaultInsufficientIdle`, orphaned by this diff.
- `AsyncNavVaultErrors.sol:108` `AsyncNavVaultRequestLocked`, orphaned by this diff.
- `AsyncNavVaultErrors.sol:122` `AsyncNavVaultRequestStale`, orphaned by this diff.

Unused imports, where the importing file never references the name it pulled in.

- `AsyncNavVaultAssetManagement.sol:6` imports `AsyncNavVaultInsufficientIdle`.
- `AsyncNavVaultClaims.sol:8` imports `AsyncNavVaultZeroAddress`.
- `AsyncNavVaultRequests.sol:7` imports `AsyncNavVaultZeroAddress`.
- `MultiOwnerDeterministicValidator.sol:11` imports `SIG_VALIDATION_SUCCESS_UINT`.

Scanned and clean. No declared-but-unemitted events, no unused constants or immutables, and no unused internal or private functions. `_convertToShares` reads as unused but is an OpenZeppelin `ERC4626` override called by the base contract.

RECOMMENDATION

Delete the five dead errors and the four unused imports.

RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-04

minShares check can never fail

CATEGORY	Superfluous Code
LOCATION	AsyncNavVaultClaims.sol:195-219
REVIEW	Remediation Review 2

— DESCRIPTION

`claimMint` added a `minShares` parameter this round, documented as the minimum escrowed shares the claim must deliver. The delivered amount is `shares`, which the caller passes in, so `shares < minShares` only reverts when the caller contradicts its own input. The ERC-7540 `mint()` entry point passes `minShares = 0`, so the check only exists on direct `claimMint` calls, where it is tautological. The parameter never protects against anything and is misleading for integrators.

— RECOMMENDATION

Remove `minShares` from `claimMint`, or document that it is redundant for this function and kept only for signature symmetry with `claimDeposit`.

— RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-05

Unreachable clamp in `_maxWithdraw`

CATEGORY	Superfluous Code
LOCATION	AsyncYieldSplitVaultSettlement.sol:646-726
REVIEW	Remediation Review 2

— DESCRIPTION

This round's `_maxWithdraw` refactor routes every sole holder (`ownerBalance == snapshot.supply`) into the dedicated full-burn branch and adds a `_clampedNonFinalAssets` fallback for everyone else. That fallback needs `requiredShares == snapshot.supply` with `ownerBalance < snapshot.supply`, but line 646 already reverts whenever `requiredShares > ownerBalance`, so the clamp at line 724 can never execute. If that case were reachable, `maxWithdraw` would revert instead of clamping, violating ERC-4626.

The protocol fuzz suite confirms it. In the yield-split campaign, line 646 was hit 509,147 times, the full-supply check 313,905 times and the sole-holder branch 25,248 times. Line 724 and the whole `_clampedNonFinalAssets` body were hit 0 times.

— RECOMMENDATION

Remove the unreachable fallback and `_clampedNonFinalAssets`, or, if non-sole holders are meant to be clamped, move the check at line 646 after the full-supply branch so the clamp can run.

— RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-06

claimWithdraw pays assets for 0 shares

CATEGORY	Events
LOCATION	AsyncNavVaultClaims.sol:257-266
REVIEW	Remediation Review 2

DESCRIPTION

A partial `claimWithdraw` derives the shares it consumes from the assets requested, then clamps that result to `request.claimableShares - 1` so a partial claim can never retire the request outright (`AsyncNavVaultClaims.sol:261-263`). When the settled request holds exactly one claimable share atom, that clamp is zero, so the minimum is zero and the call consumes no shares at all. The vault still transfers the assets.

The state stays coherent throughout. `_consumeRedeemClaim` subtracts the assets from both `request.claimableAssets` and `pendingClaims`, so no caller can withdraw more than the request was settled for, and the final full claim retires the remaining share. Two narrower defects survive.

First, the ERC-4626 `Withdraw` event carries `shares` as zero while assets leave the vault. An indexer reconstructing share flow from events sees value move with no corresponding burn on that log, and only recovers the correct total after the request's final claim.

Second, the entry point is inconsistent with its siblings. `claimRedeem` and `claimDeposit` both reject a zero computed output with `AsyncNavVaultZeroClaimOutput`; `claimWithdraw` carries no equivalent check, and no shipped document explains the difference.

RECOMMENDATION

Add `if (shares == 0) revert AsyncNavVaultZeroClaimOutput();` after the partial-branch computation in `claimWithdraw`, matching the guard already present in `claimRedeem` and `claimDeposit`.

RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-07

Return retry can land on a new cohort

CATEGORY	Trust Assumptions
LOCATION	AsyncYieldSplitVault.sol:550-556
REVIEW	Remediation Review 2

— DESCRIPTION

The remediation for "Close redirects manager return" added an epoch guard:

`returnGrossYieldAssets(uint256 expectedEpoch, uint256 assets)` reverts with `EpochMismatch` before moving tokens (:550-556), and `returnGrossYieldVaultShares` does the same (:567-574). The front-run itself is now blocked.

The guard turns the attack into a failure that invites an unsafe retry. `closeEpoch` is permissionless, so an attacker holding a dust request can advance the epoch at the boundary. The manager's return reverts, and the error names the epoch only. Re-signing with the current epoch is the natural response, but the epoch moved precisely because a new cohort was minted, so supply is no longer zero and the returned value is credited as active yield to that cohort instead of becoming orphan backing. There is no attacker gain without that retry, and the cost of the grief is gas.

— RECOMMENDATION

Bind the return to the condition the manager actually relies on, not only to the epoch: accept an `expectZeroSupply` flag, or check `totalSupply() == 0` when the return is intended as orphan backing, so a retry after a front-run reverts instead of crediting the new cohort. Alternatively, state in the operations runbook that an `EpochMismatch` retry must re-check `totalSupply() == 0`, since the revert reason does not reveal why the epoch advanced.

— RESOLUTION

Status: Reported during the fix review period.

INFORMATIONAL I-08

Safe owners cannot reuse replayable signatures

CATEGORY	Trust Assumptions
LOCATION	src/glider-kernel-validator/contracts/MultiOwnerDeterministicValidator.sol:_verifyAddressOwner():1490-1513
REVIEW	Remediation Review 2

— DESCRIPTION

The single `allow1271Multichain` flag permits a configured ERC-1271 owner on both the chain-bound and the replayable multichain paths. In replayable mode `_verifyAddressOwner` selects the chain-independent raw Glider digest (L1512) and passes it unchanged to the owner's `isValidSignature` (L1526), deliberately preserving cross-chain reuse.

A Safe owner reintroduces the chain binding internally. Safe's

`CompatibilityFallbackHandler.isValidSignature` hashes `abi.encode(_dataHash)` inside a `SafeMessage` envelope whose domain comes from `Safe.domainSeparator()`, which incorporates `chainid()`. Even with identical addresses, code, owners, generation and Glider authorization, chains A and B therefore produce different Safe signing hashes, so ordinary off-chain Safe signatures collected for chain A do not authorize chain B.

The configuration is reachable and not excluded anywhere: a deployed Safe can be installed as an address owner with `allowEoa=false`, `allow1271=true`, `allow1271Multichain=true`, and there is no Safe exclusion or implementation allowlist. The existing test at

`test/MultiOwnerDeterministicValidator.t.sol:784` installs exactly that configuration and asserts replayable-mode validation, but only ever on the current chain; it never varies the chain ID, so the suite does not exercise this case. The cross-chain ERC-1271 test at `:895` uses `MockERC1271Configurable` instead of a real Safe.

Impact: an integrator can enable the only ERC-1271 multichain opt-in for a Safe and successfully exercise replayable mode on the source chain, yet a pre-collected cross-chain batch fails on every destination. Recovery or automation is delayed until the Safe threshold supplies separate per-chain signatures. Funds are not at risk, the failure is closed, and ordinary chain-bound Safe authorization remains available. Safe's per-chain on-chain message/hash approvals can also make identical signature bytes validate after a separate approval transaction on each chain, which still does not deliver the intended sign-once workflow.

— RECOMMENDATION

Document that replayable mode requires an owner whose ERC-1271 check verifies the raw digest unchanged, and that Safe-style owners re-wrap it in a chainid-bound domain and are

I-08 Safe owners cannot reuse replayable signatures

CONTINUED

therefore chain-bound. The docs currently describe the raw-digest forwarding (docs/CORE_FLOWS.md:249-253 , docs/supplemental/VALIDATOR_DECISIONS.md:137-141) but do not state this owner-side requirement; the identical-code warning at docs/ASSUMPTIONS_AND_TRADEOFFS.md:117-125 covers a different failure mode. Optionally reject or warn on allow1271Multichain for such owners at configuration time rather than failing silently on the destination chain, and add a chain-ID-varying replayable test with a real Safe owner.

PROOF OF CONCEPT <https://github.com/GuardianOrg/SentryRepo-Glider-Remediations-Round-2-7b4bfd59-01a08a6e24af79c8873ecaa6bca06188/blob/refinery/poc-01a08a6e-3deb-783f-9967-64743ef65a35/test/refinery-poc/safe-backed-replayable-mode-silently-requires-a-new-signature-8f5901ec.sol>

— RESOLUTION

Status: Reported during the fix review period.

§ 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.