Steakhouse: Leveraged Lending ERC4626 Yield Vault

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Eric Wang, Lead Security Researcher
Jonatas Martins, Security Researcher

December 15, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Steakhouse builds transparent, efficient, and accessible financial primitives to power the next generation of capital markets on public blockchains.

From Nov 3rd to Nov 14th the Cantina team conducted a review of box on commit hash e8356da5. The team identified a total of **35** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	13	5	8
Gas Optimizations	7	7	0
Informational	14	8	6
Total	**35**	**21**	**14**

The Cantina Managed team reviewed Steakhouse's box holistically on commit hash 80a5779f and concluded that all findings were addressed and no new vulnerabilities were identified.

2.1 Scope

The security review had the following components in scope for box on commit hash e8356da5:

```
src
├── Box.sol
├── BoxAdapter.sol
├── BoxAdapterCached.sol
├── factories
│   ├── BoxAdapterCachedFactory.sol
│   ├── BoxAdapterFactory.sol
│   ├── BoxFactory.sol
│   ├── FundingAaveFactory.sol
│   └── FundingMorphoFactory.sol
├── FundingAave.sol
├── FundingMorpho.sol
├── interfaces
│   ├── IBox.sol
│   ├── IBoxAdapter.sol
│   ├── IBoxAdapterFactory.sol
│   ├── IBoxFactory.sol
│   ├── IFunding.sol
│   ├── IOracle.sol
│   └── ISwapper.sol
└── libraries
    ├── Constants.sol
    ├── ErrorsLib.sol
    └── EventsLib.sol
```

3 Findings

3.1 High Risk

3.1.1 Share price manipulation attack in `Box.flash()` callbacks

Severity: High Risk

Context: Box.sol#L650-L672

Description: At the beginning of the `Box.flash()` function, the NAV is cached, so `totalAssets()` remains the same throughout the entire flash operation. However, the `Box` allows deposits (restricted to feeders) and withdrawals (by anyone holding the vault share), which changes `totalSupply()`. This would cause an inconsistency where `totalAssets()` remains the same, while `totalSupply()` is updated to the latest value. As a result, the share price may change during the flash operation. It is possible to exploit such an inconsistency to extract value from the vault. Consider the following example:

1. Initial state: 6000 USDC deposited into the vault, with 6000 shares minted.
2. The attacker (an allocator or anyone during winddown) holds 3000 shares.
3. The attacker calls `flash()` with a flash amount of 3000 USDC.
4. The vault caches the NAV, so `totalAssets` is 6000 USDC.
5. During the callback, the attacker redeems 1500 shares. They get 1500 / 6000 * 6000 = 1500 USDC.
6. During the callback as well, the attacker redeems the remaining 1500 shares. They get 1500 / 4500 * 6000 = 2000 USDC, as `totalSupply()` is decreased to 4500 shares.
7. After the callback, the attacker gets back their original 3000 USDC.

As a result, the attacker redeemed 3000 shares but received 1500 + 2000 = 3500 USDC, more than they would have received from a direct redemption.

Recommendation: Consider disabling deposits or withdrawals during the flash callback by reverting with an error in the `deposit()`, `mint()`, `withdraw()`, and `redeem()` functions so to avoid changes to `totalSupply()`.

Steakhouse: Fixed in PR 6 We resolved this by reverting on mint and redeem when `_cachedNavDepth > 0`.

Cantina Managed: Verified.

3.2 Low Risk

3.2.1 Stale adapter NAV mints free shares

Severity: Low Risk

Context: BoxAdapterCached.sol#L104-L106

Description: `BoxAdapterCached.realAssets()` blindly returns the cached `totalAssets` whenever the adapter's allocation is non-zero and never recomputes the Box balance on read, so NAV queries keep using whatever value was last written during `allocate/deallocate` or `updateTotalAssets` even if hours of yield have accrued in the `Box`. The implementation is literally just returning the cached number:

```
function realAssets() external view returns (uint256) {
    return allocation() != 0 ? totalAssets : 0;
}
```

`updateTotalAssets()` is the only place that refreshes the cache, yet it can only be called by allocators or sentinels unless a full day has passed, which makes long stale periods expected whenever allocators are inactive:

```
function updateTotalAssets() external {
    require(
        IVaultV2(parentVault).isAllocator(msg.sender) ||
            IVaultV2(parentVault).isSentinel(msg.sender) ||
            totalAssetsTimestamp + 1 days < block.timestamp,
```

```
        NotAuthorized()
    );
    _updateTotalAssets();
}
```

`VaultV2`'s deposit/mint path trusts adapters to report live balances inside `accrueInterestView`:

```
for (uint256 i = 0; i < adapters.length; i++) {
    realAssets += IAdapter(adapters[i]).realAssets();
}
uint256 newTotalAssets = MathLib.min(realAssets, maxTotalAssets);
return assets.mulDivDown(newTotalSupply + virtualShares, newTotalAssets + 1);
```

If `Box` accrues yield while the cache stays stale, `realAssets()` keeps returning the old lower value, `newTotalAssets` remains depressed and `previewDeposit/previewMint` mint too many vault shares. Attackers can monitor the `Box` contract off-chain, wait for NAV to drift, then deposit into `VaultV2` to receive underpriced shares and later redeem them for the higher, refreshed NAV, stealing the accumulated yield from honest depositors. Impact: recurring economic loss proportional to the NAV drift while the cache is stale and repeatable whenever allocators neglect to refresh the adapter.

Recommendation: Consider removing the `BoxAdapterCached` version and always use the `BoxAdapter` implementation.

Steakhouse: Acknowledged. `VaultV2` introduces a maximum interest-rate parameter that ensures value accrues linearly at the vault level, regardless of what happens in the underlying strategy. For complex vaults (particularly those built on `Box`), we intend to set this cap with a small buffer so the vault can safely absorb scenarios where a PT trades down. If the allocator fails to call `updateTotalAssets` appropriately, the vault should be considered unsafe to use. In the extreme case, a malicious curator could even set the max rate to 0%, effectively freezing accrual. We plan to rely on the non-cached version of asset accounting whenever possible. However, we want to preserve the ability to use the cached version, as it reduces gas costs and may provide protection against downside oracle manipulation.

Cantina Managed: Acknowledged.

3.2.2 Invisible Aave isolation mode

Severity: Low Risk

Context: FundingAave.sol#L238

Description: `FundingAave.pledge` unconditionally supplies a collateral token and then calls `pool.setUserUseReserveAsCollateral(token, true)`. If that token is marked as an isolated collateral on Aave, enabling it blocks every other collateral and restricts future borrows to governance-approved stablecoins, yet the `Box` stack never tracks or validates those constraints. An unsuspecting allocator can enable an isolated asset first and accidentally place the whole `Box` position into isolation mode. Later attempts to pledge additional tokens or to borrow any non-approved asset will revert with opaque Aave errors, leaving automation stuck mid-flight. There is also no awareness of isolation debt ceilings: once Aave's cap is reached, `borrow` simply reverts and the strategy has no hint why. Impact: governance and allocators can brick investment workflows or strand collateral with no diagnostics, exposing the vault to outages precisely when new assets are onboarded.

Recommendation: Tag collateral tokens that Aave deploys in isolation mode (or query the Aave pool configuration) and enforce the rules locally: forbid enabling multiple collaterals when one is isolated, restrict `debtToken` choices to the allowed stablecoins, and surface meaningful errors when the global isolation debt limit would be exceeded. Alternatively, block isolated assets entirely unless governance explicitly whitelists them with the corresponding safety rails.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 `CREATE2` salt front-running blocks governance deployments

Severity: Low Risk

Context: BoxFactory.sol#L29

Description: `BoxFactory.createBox` exposes a permissionless `CREATE2` deployment that takes the caller-supplied salt verbatim. The resulting `Box` address depends only on the factory, the salt and the constructor bytecode/parameters. If governance pre-announces a proposal that will call `createBox` with a known salt so the deterministic address can be referenced in the same bundle, an adversary can front-run the execution window and call `createBox` first with the exact same salt and arguments. Because `CREATE2` cannot reuse a salt once the address exists, the scheduled deployment reverts, aborting the governance transaction. The attacker gains no control over the `Box` instance (they had to replicate the parameters), but they do acquire an inexpensive denial-of-service lever that forces governance to re-propose the launch with a fresh salt.

Recommendation: Derived salts from trusted context (e.g. hash the user-provided salt with `msg.sender` or a governance nonce) or restrict `createBox` to vetted callers, preventing outsiders from front-running the deterministic address before the scheduled deployment executes.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.

3.2.4 Donation attack can partially absorb the first deposit

Severity: Low Risk

Context: Box.sol#L213-L218

Description: When the base asset already uses 18 decimals, the `Box` constructor sets `virtualShares = 10 ** 0 = 1`, so the ERC4626 conversion relies almost entirely on real supply. The conversion function shows the problem:

```
function convertToShares(uint256 assets) public view returns (uint256) {
    uint256 supply = totalSupply();
    return assets.mulDiv(supply + virtualShares, totalAssets() + 1, Math.Rounding.Floor);
}
```

Before any feeder seeds the vault, an attacker can donate `D` units of the asset directly to the contract (no role check blocks transfers). This leaves `totalSupply() == 0` but `totalAssets() == D`. When the legitimate feeder deposits `A <= D`, the formula becomes `floor(A * 1 / (D + 1))`, which evaluates to zero because `virtualShares` is only one wei. `_depositMint` still pulls `A` tokens from the feeder and emits the event even though `shares == 0`, so the entire bootstrap deposit is burned and remains stranded in the vault. The attacker cannot withdraw the victim's funds, but they can grief by forcing operators either to bridge in additional capital greater than `D` before seeding or to redeploy/clean the vault. This is purely a denial-of-service vector, no attacker profit is possible.

Proof of Concept: test/BoxDonationFork.t.sol demonstrates the attack on a fork: the test donates USDC to an unseeded Box, invokes `deposit` for less than or equal to the donated amount and asserts that the feeder's transaction pulls assets while minting zero shares, leaving their capital irrecoverably stuck.

Recommendation: Consider reverting whenever `totalSupply() == 0` but `totalAssets() > 0`, preventing deposits until operators drain the donation.

Steakhouse: Acknowledged. As for Morpho, we plan to use a dead deposit to avoid this issue.

Cantina Managed: Acknowledged.

3.2.5 Winddown allows unvetted swappers to skim the slippage budget

Severity: Low Risk

Context: Box.sol#L387-L432

Description: When `shutdownTime` has elapsed and the `Box` enters winddown, `allocate` deliberately relaxes its caller check so "anyone" can source the tokens needed to repay outstanding debt, provided `_debtBalance(token) > 0`. The function still accepts an arbitrary `ISwapper` implementation supplied by the caller and only enforces that the swap delivers at least `minTokens = expectedTokens * (1 - slippageTolerance)`. Because winddown widens the tolerance up to ~1% over time, a malicious helper can register itself as the swapper, keep the base asset it receives, and simply hand back the minimum acceptable number of debt tokens. The transaction succeeds

because the numeric slippage check passes, and the attacker immediately repays the debt with the `Box`'s own tokens to make space for another call. Repeating this loop drains roughly the entire slippage allowance (≈1% of every unwind trade) into the attacker's contract until the shutdown completes. This does not allow draining principal beyond that budget, but it converts every winddown swap into a guaranteed loss equal to the configured tolerance.

Recommendation: Document that winddown swaps are fully trusted and the tolerance represents the maximum economic loss. If that is unacceptable, either (1) keep `allocate` gated to known allocators even during winddown, (2) hardcode/whitelist trusted swappers for the unwind phase, or (3) perform the unwind through a guardian-controlled swap path instead of arbitrary `ISwapper` contracts.

Steakhouse: Acknowledged. Other options were considered, but we believe this approach offers the best overall trade-off. The allowed slippage can increase up to 1%, which reflects the potential fee required by a third party executing the wind-down. In the intended ("happy-path") scenario, the allocator will have already placed the Box in a fully liquid position before the wind-down procedure begins.

Cantina Managed: Acknowledged.

3.2.6 Dust transfers blocks token delisting

Severity: Low Risk

Context: Box.sol#L959

Description: Removing an investment token requires a clean slate: `removeToken` checks both that the caller is the curator and that the Box holds zero balance for the token. The only helper that can sweep balances, `skim`, explicitly rejects whitelisted tokens and the base asset. This means any outsider can front-run a planned delisting by transferring a single wei of the token to the Box; from that point on `removeToken` reverts with `ErrorsLib.TokenBalanceMustBeZero`, and governance must first clear the dust (for example via a manual deallocation) before it can proceed.

Impact: The grief lasts only until the curator runs a tiny `deallocate`/`reallocate` through a trusted swapper to zero the balance, but it does force governance to spend extra transactions and gas to keep the whitelist clean.

Recommendation: Provide a governance-controlled escape hatch, for example, allow the owner/guardian to sweep dust balances of whitelisted tokens before removal, or bundle deallocation and removal into a single transaction that ignores residual dust below a configurable threshold, so an adversary cannot lock tokens on the whitelist with a trivial transfer.

Steakhouse: Acknowledged. Token delisting are not daily operations and we feel adding complexity at this stage is not the right tradeoff. This is something we can take care at a higher level process.

Cantina Managed: Acknowledged.

3.2.7 Include boundary checks on duration-related parameters

Severity: Low Risk

Context: Box.sol#L154-L156

Description: When creating a new `Box` contract, the `_slippageEpochDuration` and `_shutdownSlippageDuration` parameters are checked to be non-zero, while there can be upper bound checks with a reasonable value to prevent configuration errors in deployment scripts, etc.

Similarly, there can be a lower bound of the `shutdownWarmup` parameter to give the guardian sufficient time to notice the shutdown and `recover()` if needed, in case the curator is compromised and initiates an unnecessary shutdown.

Recommendation: Consider adding lower or upper bound checks for the parameters mentioned above.

Steakhouse: Acknowledged. Those parameters are set at the creation of a Box and are immutable, therefore users can verify before using the product or veto when such a Box is added to the vault. Should a Box be badly configured, it shouldn't pass our Q&A processes neither and we would just recreate a new one.

Cantina Managed: Acknowledged.

3.2.8 Incorrect rounding directions

Severity: Low Risk

Context: Box.sol#L416-L418

Description: As best practice, integer divisions should be rounded with a direction that benefits the protocol or enforces stricter checks.

Take `Box.sol#L416` as an example. When calculating the expected tokens from a swap, it is recommended to round up to avoid underestimating the expected amount. Note that the `minTokens` calculation at L417 rounds up, which follows best practice.

Below are the divisions that need to be rounded up:

- Box.sol#L407-L408.
- Box.sol#L416.
- Box.sol#L424.
- Box.sol#L464.
- Box.sol#L515-L516.
- Box.sol#L525.
- Box.sol#L1360: The raw `/` division.

Recommendation: Consider rounding up the divisions as mentioned.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.2.9 Compromised curator can abdicate timelock functions immediately

Severity: Low Risk

Context: Box.sol#L889-L895

Description: The `Box.abdicateTimelock()` function does not require a timelock. Assume the curator is compromised, they can abdicate the timelock functions immediately. The functions cannot be executed anymore, even after the owner removes the compromised curator since abdicating the functions is irreversible.

The timelock functions that could be affected include:

- `setGuardian()`.
- `decreaseTimelock()`.
- `setIsFeeder()`.
- `setMaxSlippage()`.
- `addToken()`.
- `changeTokenOracle()`.
- `addFunding()`.
- `addFundingFacility()`.
- `addFundingCollateral()`.
- `addFundingDebt()`.

Recommendation: Consider adding a timelock to the `abdicateTimelock()` function so the guardian can revoke `abdicateTimelock()` calls if the curator is compromised.

Steakhouse: Acknowledged. Our rationale is to have a timelock only for action that are increase risk for users. In such case, the solution would be to create a new Box under the VaultV2 and migrate. While being an inconvenience, it would be minor compared to the fact of having the curator msig compromised.

Cantina Managed: Acknowledged.

3.2.10 Risks of unset or insufficiently long timelock periods

Severity: Low Risk

Context: Box.sol#L746-L748

Description: Several critical functions of the `Box` contract require a timelock to prevent a compromised curator from immediately changing settings and putting users at risk without them or the guardian noticing in time.

Note that the functions' timelock period is 0 by default. Users and the box owner should ensure that a sufficiently long timelock period is set for all critical functions, especially for the `setGuardian()` function, since the guardian can protect users if the curator is compromised.

Recommendation: Consider adding a note in the docs or comments to alert users to verify that the timelock periods for critical functions are enough. Also, if applicable, consider adding a check in the `submit()` function at the contract level to revert if the `delay` value is 0, which likely indicates the timelock is unset.

Steakhouse: Fixed in PR 6. Timelocks are expected to be set to a minimum of three days, whereas the guardian decision-making process (via the AragonDAO) is designed to operate on a one-day turnaround. Users can verify it before using the product and then veto any change in the timelocks.

Cantina Managed: Verified. A comment is added to the `Box.submit()` function as recommended.

3.2.11 Revert if an oracle is unset when calculating NAV

Severity: Low Risk

Context: Box.sol#L1406-L1413, FundingAave.sol#L331-L335, FundingMorpho.sol#L279, FundingMorpho.sol#L320-L322, FundingMorpho.sol#L331-L334

Description: To calculate the current NAV, the `Box._nav()` function sums up the current values of all the tokens based on the oracle price. The function skips the case if `address(oracle) == address(0)` (see L1408). However, we suggest removing the `if` condition because:

1. Given the current checks in the `addToken()` and `changeTokenOracle()` functions, it is not possible for a token's oracle to be `address(0)`.
2. Even if the oracle address is zero, it is considered better to raise an error instead of skipping it, since it is likely a programming error and could result in an incorrect NAV.

The same logic applies to the `nav()` function in `FundingAave` and `FundingMorpho`. It applies to the `ltv()` function in `FundingMorpho` as well, as it is unlikely that a Morpho market has an oracle set to `address(0)`.

Recommendation: Consider removing the `if (address(oracle) == address(0))` condition and replacing it with an explicit check to revert with an error if the oracle is `address(0)`.

Steakhouse: Fixed in PR 6

Cantina Managed: Verified. Note that while there are no explicit zero-address checks on the oracle address when calculating the NAV, this does not pose a security issue, since calling the zero address returns an empty byte, which causes the ABI decoder to revert when decoding an `uint256` value from it.

3.2.12 Possible duplicate markets in `FundingMorpho`

Severity: Low Risk

Context: FundingMorpho.sol#L344-L347

Description: The `FundingMorpho` contract assumes that `facilityData` input can be decoded into the `MarketParams` struct. Note that appending arbitrary bytes to a properly encoded `facilityData` would decode into the same `MarketParams` since `abi.decode()` ignores the appended bytes. Since `isFacility` compares the entire `facilityData` bytes, it is possible to add the same market to `FundingMorpho` multiple times, which could result in an incorrect NAV.

Recommendation: Consider adding a check

```
require(facilityData.length == FACILITY_DATA_LENGTH, ...)
```

in the `decodeFacilityData()` function with `FACILITY_DATA_LENGTH = 160` to ensure a Morpho market can only be added once.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.2.13 Risks of allowing anyone to execute `Box.changeTokenOracle()`

Severity: Low Risk

Context: Box.sol#L982-L988

Description: The `Box.changeTokenOracle()` function can be executed by anyone after the timelock has passed, and if the box is not in winddown mode. Changing the token's oracle would instantly update its asset value, changing the NAV and potentially creating an arbitrage opportunity. Arbitrageurs can try to sandwich the oracle update atomically by depositing and withdrawing (or vice versa) to profit.

This is a general concern that could also occur with oracle price updates, especially when the price difference is large enough to make arbitrage profitable. This issue highlights the specific `changeTokenOracle()` case, since anyone in the regular case can call the function.

Recommendation: Curators should be cautious about whether changing token oracles could create profitable arbitrage opportunities. The waiting timelock period needs to be considered as well.

On the other hand, besides having a timelock, the `changeTokenOracle()` could also be restricted to the curator. The curator could then submit the transactions to services with MEV protection to reduce the risks.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified. The `changeTokenOracle()` function is now restricted to the curator (if the Box is not in winddown mode).

3.3 Gas Optimization

3.3.1 Cache asset address in `FundingAave.nav`

Severity: Gas Optimization

Context: FundingAave.sol#L327

Description: `FundingAave.nav()` compares each collateral and debt token against the base asset by calling `oraclesProvider.asset()` inside the loops. Because the oracle provider is external to the funding module, every iteration makes another external call even though the asset address remains constant throughout the function. When many collateral/debt tokens are registered this wastes gas: each token incurs an extra `STATICCALL` solely to re-fetch the same address. Caching the result once before the loops would eliminate two external calls per token without changing semantics.

Recommendation: Read `address asset = oraclesProvider.asset();` once at the top of `nav()` and compare against that local when iterating over `collateralTokens` and `debtTokens`.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.3.2 Cache facilityHash in `FundingAave` loops

Severity: Gas Optimization

Context: FundingAave.sol#L132-L140

Description: `FundingAave.isFacility` and `_findFacilityIndex` recompute `keccak256(facilityData)` inside each loop iteration when walking `facilities`. The calldata argument never changes through the loop, so hashing it once before iterating and comparing to the

cached value would avoid re-running keccak256 for every entry. Gas impact grows linearly with the number of facilities; caching removes one Keccak per element in both functions.

Recommendation: Store `bytes32 facilityHash = keccak256(facilityData);` before the loop in both functions and compare `facilityHash` against `keccak256(facilities[i])`.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.3.3 Skip oracle call when LTV has no collateral

Severity: Gas Optimization

Context: FundingMorpho.sol#L279

Description: `FundingMorpho.ltv()` fetches each facility's collateral amount, and even when that amount is zero it still queries the oracle address to pull a price before concluding the LTV is zero. The function already documents "returns 0 if there is no collateral," so it can short-circuit earlier: after reading `collateralAmount`, check for zero and return immediately instead of making a needless external call and `mulDivDown`. With multiple collateral tokens registered, the current code burns gas calling oracles even though the branch ends up returning zero.

Recommendation: After computing `collateralAmount`, insert `if (collateralAmount == 0) return 0;` so the function exits before touching oracles in the no-collateral case.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.3.4 Reuse debtToken in `FundingMorpho.repay`

Severity: Gas Optimization

Context: FundingMorpho.sol#L239

Description: `FundingMorpho.repay` verifies that the `debtToken` argument matches `market.loanToken`, yet it later redeclares `IERC20(market.loanToken)` solely to call `forceApprove`. Casting again is redundant once the equality check passes. Using the original `debtToken` parameter would save a stack slot and keep the code clearer.

Recommendation: Replace `IERC20(market.loanToken).forceApprove(...)` with `debtToken.forceApprove(...)`, reusing the validated argument instead of re-wrapping the same address.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.3.5 Return cached totalAssets after update

Severity: Gas Optimization

Context: BoxAdapterCached.sol#L108-L126

Description: `BoxAdapterCached._updateTotalAssets()` stores `box.previewRedeem()` into `totalAssets` and updates the timestamp, but it does not return the freshly computed value. Callers like `allocate`, `deallocate` and even `updateTotalAssets` itself immediately SLOAD `totalAssets` after invoking `_updateTotalAssets()` just to use the value they already calculated. Forwarding the updated amount as a return value would let callers use the in-memory result instead of wasting gas re-reading storage.

Recommendation: Have `_updateTotalAssets()` return the updated total (e.g., `return totalAssets = ...;`) and update call sites to consume that return value instead of performing an extra SLOAD right after the update.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.3.6 Use `EnumerableSet` for constant-time lookups and removals

Severity: Gas Optimization

Context: FundingAave.sol#L383-L391

Description: In the `FundingAave` and `FundingMorpho` contracts, the data structures for `facilities`, `collateralTokens`, and `debtTokens` are maintained as arrays, with a time complexity of O(n) for lookups and removals.

OpenZeppelin has a EnumerableSet library that supports O(1) lookups and removals, and therefore is more gas efficient on average than the array implementation due to fewer SLOAD/SSTORE. On the other hand, insertion requires an additional SSTORE. The benefit of using the library, in terms of gas savings, would be more significant as the number of elements in the array grows.

Recommendation: Consider using the `EnumerableSet` library to maintain the data structures in the mentioned contracts for gas optimization or code simplification purposes.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified. The `FundingBase.facilitiesArray()`, `collateralTokensArray()`, and `debtTokensArray()` functions could be further simplified with `facilitiesSet.values()`, etc.

```
function facilitiesArray() external view returns (bytes32[] memory) {
    return facilitiesSet.values();
}
```

3.3.7 Avoid `this.nav()` external calls

Severity: Gas Optimization

Context: FundingAave.sol#L211, FundingMorpho.sol#L174

Description: The `skim()` function of `FundingAave` and `FundingMorpho` calls `this.nav()` before and after the skim operation. The `nav()` function could be declared `public`, allowing the `skim()` function to reduce external calls and therefore save gas.

Recommendation: Consider making the `nav()` function `public` rather than `external`, and replace `this.nav()` with `nav()` in the `skim()` function.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4 Informational

3.4.1 Factory mapping can be overwritten

Severity: Informational

Context: BoxAdapterFactory.sol#L18-L25

Description: `BoxAdapterFactory.createBoxAdapter` is public and stores the new adapter in the `boxAdapter[parentVault][box]` mapping. There's no guard against redeploying for an existing pair, so a third party can overwrite the mapping with another adapter. The on-chain vault never consults this mapping for authorization, privileges are bound to whatever address governance added via `vault.addAdapter`. Consequently, overwriting the mapping doesn't interfere with live operations, but off-chain tooling that treats the factory's mapping as canonical could pick up the wrong adapter address and lose track of the active deployment.

Recommendation: Document that the factory mapping is informational only. If external systems rely on it, add caller checks or refuse to overwrite non-empty entries to prevent accidental pointer churn.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.2 Winddown slippage ramp can block exits

Severity: Informational

Context: Box.sol#L400

Description: During winddown both `allocate` and `deallocate` insist the trade output meets a minimum derived from `_winddownSlippageTolerance()`. The allocator check at the top of `allocate` falls back to that helper, then enforces `tokensReceived >= minTokens`, causing any shortfall to revert:

```
uint256 slippageTolerance = winddown ? _winddownSlippageTolerance() : maxSlippage;
// ...
uint256 minTokens = _calculateMinAmount(expectedTokens, slippageTolerance);
require(tokensReceived >= minTokens, ErrorsLib.AllocationTooExpensive());
```

The sell path mirrors that logic and reverts on `ErrorsLib.TokenSaleNotGeneratingEnoughAssets` when the base asset received violates the same tolerance. `_winddownSlippageTolerance()` then starts the ramp at zero and only climbs linearly to `MAX_SLIPPAGE_LIMIT`, while the constant itself hard caps at 1% in `Constants.sol`. Because the deployment scripts pass `shutdownSlippageDuration = 10 days`, the system refuses to tolerate any slippage for an entire day and stays below 1% for the remainder of the ramp. Realistic distressed unwinds often need more room, so the `require` checks above keep reverting through most of the period, blocking debt repayments and exits until the ramp finishes.

Importantly, chopping the trade into smaller clips does not help: the `require` compares per-trade execution price (`tokensReceived/expectedTokens`) against the same zero-to-1% tolerance, so any market that clears 0.5% below oracle will cause every sub-trade to revert until the ramp exceeds 0.5%.

Impact: In some cases, emergency shutdown cannot unwind positions when the market demands >0-1% slippage, leaving allocators unable to repay debt or exit during the crisis window. We can't assume price convergence between the oracle and the pool price in the time window that matters. During a shutdown you are trying to unwind under duress: liquidity dries up, spreads blow out, and the best executable price can sit several percent away from the oracle for hours or days. The oracle might refresh quickly but still reflect a TWAP, price feed bias, or governance delay, while the on-chain pool you're actually hitting remains skewed because LPs fled or routing is one-sided. Even if both eventually realign, the protocol has to service redemptions before 10 days passes (`shutdownSlippageDuration`); being forced to wait until markets "settle" defeats the purpose of an emergency exit. That's why a fixed 0->1% ramp is too tight, there are realistic windows where the oracle shows the "correct" price yet the only available exit path is 3-5% off and the system still bricks until tolerance finally catches up (if it ever does).

Proof of Concept: test/WinddownSlippageFork.t.sol forks Base mainnet, deploys a fresh Box, allocates stUSD through the live swapper, then forces shutdown while its oracle is intentionally 20% stale. The first unwind (immediately after shutdown while tolerance ≈ 0 bps) reverts, and even after fast-forwarding 11 days so the ramp caps at 1% the second unwind still reverts, showing that the `Box` cannot exit as long as the market/oracle gap exceeds the ramp ceiling.

Recommendation: Allow winddown operators to raise the slippage ceiling during emergencies for example by letting guardians set a larger cap for a specific shutdown, bypass `_winddownSlippageTolerance()` for privileged callers, or introduce a higher configurable limit once the vault enters winddown, so that necessary trades can execute even when markets move more than 1%.

Steakhouse: Acknowledged. In extreme situations, the guardian can change the oracle. We thought about it since the start of the project, and we don't want to increase slippage beyond 1%.

Cantina Managed: Acknowledged.

3.4.3 No oracle scale normalization

Severity: Informational

Context: DeployEthereum.s.sol#L62-L86

Description: `ORACLE_PRECISION` is hard-coded to 1e36 and every valuation, `allocate/deallocate` slippage math, `_nav()` and both. funding modules' NAV/ltv calculations, blindly multiplies balances by `oracle.price()` and divides by `1e36`. The code never factors in token/base decimals, so the conversion only works if each oracle already emits a price scaled by `10^(36 - tokenDecimals + baseDecimals)`. With USDC (6 decimals) and an 18-decimal Pendle token, a 24-decimal feed happens to satisfy this, but

nothing enforces it at smart contract level. Supplying a true 36-decimal feed for that token would make `neededTokens * price / 1e36` return 18-decimal units (1e12 higher than USDC), while a 6-decimal token would be off by 1e30. Because share pricing, deposit previews, slippage limits, and funding NAV reuse the same assumption, any mis-scaled feed immediately corrupts the vault's accounting, allowing underpriced share mints or blocking swaps entirely.

Recommendation: Normalize prices explicitly. Consider enforcing that each oracle to expose its own decimals and rescale on-chain (e.g., divide by `10^(oracleDecimals - (36 - tokenDecimals + baseDecimals))`).

Steakhouse: Acknowledged. This is a non-issue, because this is how Morpho oracles work (see IOracle.sol#L10-L14). We're using the same oracles we would be using in Morpho markets.

Cantina Managed: Acknowledged.

3.4.4 Predicted funding address risk

Severity: Informational

Context: Box.sol#L998-L1020

Description: The factory for `FundingMorpho` is permissionless: anyone can deploy a module and pick both the `owner_` and the `morpho_` backend. `Box.addFunding` later checks only that the module lists the `Box` as owner and that the module is empty. If governance ever queues `addFunding` with a precomputed factory address, rather than deploying the module first, an outsider can front-run the deployment, instantiate the module themselves, set `owner = Box` and point `morpho` to arbitrary logic. When the timelock executes, the malicious module becomes whitelisted. While the documented flow deploys before queuing (so the risk is mitigated), relying on predicted addresses reopens the hijack window. This is largely an operational concern: the code is permissive by design, so governance needs to stick to "deploy first, queue later" or introduce additional verification.

Recommendation: Document clearly that funding modules must be deployed before they are scheduled for `addFunding`, or add light-weight safeguards (e.g. enforcing a registry or verifying a code hash) so the timelock cannot be pointed at a front-run deployment.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Skim ETH transfer uses 2300 gas

Severity: Informational

Context: Box.sol#L370

Description: `Box.skim` routes native balances to `skimRecipient` with Solidity's `transfer`, which enforces the historical 2300 gas stipend. A recipient implemented as a Gnosis Safe or any contract that needs more than 2300 gas in its fallback will revert, so the entire skim call fails and the vault cannot recover ETH that was meant to be swept. Because only the owner may update `skimRecipient`, an attacker cannot steal funds, but the vault's administrators lose the ability to claim native fees whenever the configured recipient is a Safe that executes complex logic (the default Gnosis Safe setup does). In practice this strands the ETH in `Box` until governance changes the recipient to an EOA or redeploys the Safe.

Recommendation: Send native assets with a low-level `call` that forwards all remaining gas and checks the return flag, e.g.

```
(bool ok, ) = skimRecipient.call{value: balance}("");
require(ok, ErrorsLib.TransferFailed());
```

so Safe-based recipients can accept the sweep without reverts.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.6 Unused code

Severity: Informational

Context: ErrorsLib.sol#L4

Description: Several artifacts are included but never exercised, adding bytecode or maintenance overhead without providing functionality. `BoxAdapter` imports `MathLib` and declares `using MathLib for uint256;` despite not calling any `MathLib` helpers in the adapter; removing the directive has no effect. `FundingAave` defines `RAY` but never reads it, so the constant can be dropped. Finally, `ErrorsLib` exposes custom errors `InvalidOwner`, `CannotDepositZero`, `CannotMintZero`, `NoOracleForToken`, `CannotSkimAsset` and `AlreadySet` that have no throw sites in the repo, meaning they should be deleted to keep the error catalog lean. Keeping unused code around increases bytecode size for no benefit.

Recommendation: Delete the unused `using` directive, constant and error definitions (or wire them up if future work needs them) so the deployed contracts only contain reachable code paths and the error list reflects real behaviors.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.7 Oracles must return prices in asset denomination

Severity: Informational

Context: Box.sol#L1411

Description: The oracle set in the `Box` contract is used in `nav` calculations. The return value is expected to be in asset price (token/asset). This isn't guaranteed when adding oracles and should be verified before. Using an oracle that doesn't return asset-denominated prices will mix asset units, resulting in incorrect values, inflated or deflated depending on the oracle.

Recommendation: Verify that every oracle returns the price in token/asset format, by checking it before adding or checking the oracle returns in asset price.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 More detailed custom errors

Severity: Informational

Context: Box.sol#L395-L396

Description: In the `Box.allocate()` function, if the contract is in winddown mode, but the debt balance of the `token` is 0, an error `OnlyAllocatorsOrWinddown()` is raised. However, the error name does not reflect that the debt balance is 0, which is the actual cause of the failure. Raising a separate custom error, e.g., `NoDebtBalance()`, would be more accurate in this case.

The check can be updated to:

```
if (winddown) {
    require(_debtBalance(token) > 0, ErrorsLib.NoDebtBalance());
} else {
    _onlyAllocatorNotWinddown();
}
```

Similar changes can be applied to the check in the `Box.deallocate()` function as well.

Recommendation: Consider applying the above suggested changes.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 Explicitly check whether the target function is abdicated in `Box.submit()`

Severity: Informational

Context: Box.sol#L815-L817

Description: If the curator abdicates a function, its corresponding timelock value is set to `TIMELOCK_DISABLED`, which is `type(uint256).max`. Subsequent `submit()` calls with `data` targeting the abdicated function will revert with an arithmetic overflow error when calculating `executableAt[data]` at L817.

Reverting is expected behavior. On the other hand, for better clarity, consider adding an explicit check `if (delay == TIMELOCK_DISABLED)`, and revert with a custom error, e.g., `FunctionDisabled()`.

Recommendation: Consider implementing the above suggestions.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.10 Custom error name could be more accurate

Severity: Informational

Context: Box.sol#L858

Description: In the `Box.increaseTimelock()` function, a `TimelockDecrease()` error is raised if the new timelock period is not larger than the current one. The error could be renamed to `TimelockDecreaseOrSame()` or `TimelockNotIncreasing()` to be more accurate. Similar for the `TimelockIncrease()` error at L876.

Recommendation: Consider renaming the errors for clarity.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.11 Suggestions on the `IBox` interface

Severity: Informational

Context: IBox.sol#L16

Description:

1. Consider making the `IBox` interface inherit from `IOracleCallback`. It is to ensure the `Box` contract implements the correct interface for the `IFunding.nav(IOracleCallback oraclesProvider)` function at the programming level.
2. Consider adding the `skimFunding()` and `multicall()` functions in the `IBox.sol` interface.

Recommendation: Consider implementing the above suggestions.

Steakhouse: Fixed in PR 6.

Cantina Managed: Verified.

3.4.12 Reduce code duplication in Funding modules

Severity: Informational

Context: FundingAave.sol#L71

Description: `FundingAave` and `FundingMorpho` have several identical function implementations. Code duplication could be reduced by creating, e.g., a `FundingBase` contract that implements common functions, and by having `FundingAave` and `FundingMorpho` override module-specific functions, such as the `nav()` calculation.

Recommendation: Consider implementing a base contract for funding modules and refactoring the codebase.

Steakhouse: Fixed in PR 6. Wrote a `FundingBase` to inherit from.

Cantina Managed: Verified.

3.4.13 Suggestions on code comments and docs

Severity: Informational

Context: Box.sol#L647, Box.sol#L1296

Description:

1. Box.sol#L647: Update the comment "@dev Caller must implement IBoxFlashCallback and return tokens within same transaction" since the `caller` does not return the tokens, but the `Box` contract does.
2. Box.sol#L1296: The comments can be updated to "Checks if token is whitelisted or is the base asset" to be more accurate.
3. FundingAave.sol#L78: The `owner` variable is misleading, usually the `owner` can be changed through functionalities and never be set as immutable. For better consistency of the naming, it could be changed to `box`.

Recommendation: Consider implementing the above suggestions.

Steakhouse: Fixed in PR 6

Cantina Managed: Verified fix for items 1 and 2, acknowledged item 3.

3.4.14 Non-existent Morpho markets can be added as facilities

Severity: Informational

Context: FundingMorpho.sol#L62-L71

Description: The `FundingMorpho.addFacility()` function does not check whether the provided market exists on Morpho, so it is possible to add a non-existent market as a facility. Note that it is also possible to call `removeFacility()` to remove the market.

Recommendation: If adding a non-existent market as a facility is not an expected use case, consider adding `morpho.market(market_id)` and checking `lastUpdate != 0` in the `addFacility()` function.

Steakhouse: Acknowledged.

Cantina Managed: Acknowledged.