Security Review

Cantina Managed

MAPLE FINANCE

Cantina Managed review by:
Alireza Arjmand, Lead Security Researcher
MiloTruck, Lead Security Researcher

CANTINA

October 15, 2026

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3
- 2.2 Codebase context . . . 3
- 2.3 Trust assumptions . . . 4
- 2.4 Cantina Managed Team Statement . . . 4

3 Findings **6**
- 3.1 Low Risk . . . 6
  - 3.1.1 Small Aave amounts can revert and temporarily block strategy operations . . . 6
  - 3.1.2 `MapleSkyStrategy.availableToWithdraw()` could overestimate withdrawal liquidity by 2 units . . . 6
  - 3.1.3 Sky PSM configuration could block `MapleSkyStrategy` operations . . . 7
  - 3.1.4 Unbounded `fallbackTout` can make Sky valuation revert for every attached pool 8
  - 3.1.5 `removeStrategyManager()` checks the rounded position value instead of the share balance . . . 8
- 3.2 Informational . . . 9
  - 3.2.1 Outstanding shares after a total loss permanently block `MapleStrategyManager` deposits . . . 9
  - 3.2.2 `MapleSkyStrategy._gemForUsds()` could underestimate redeemable assets by one unit . . . 9
  - 3.2.3 Sky fee setters can still move the fee checkpoint while the PSM is halted . . . 10
  - 3.2.4 Batched Minor Issues . . . 11

4 Appendix **12**
- 4.1 Aave V3 supplier integration notes . . . 12
  - 4.1.1 Overview . . . 12
  - 4.1.2 Version summary . . . 12
  - 4.1.3 Rounding rules . . . 12
  - 4.1.4 Other integration pitfalls . . . 14
  - 4.1.5 Ways `Pool.supply()` can revert . . . 15
  - 4.1.6 Ways `Pool.withdraw()` can revert . . . 15

1 Introduction

1.1 About Cantina

Cantina is a security platform that pairs a world-class network of security researchers with agentic AI to help teams building onchain find and fix real risk. Through managed reviews, competitions, and bounties, Cantina secures protocols before vulnerabilities can be exploited.
Learn more at cantina.security

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

Maple is redefining asset management for the digital age — onchain, transparent, and built to scale. Through institutional-grade lending and yield strategies, Maple brings the rigor of traditional finance to the transparency and innovation of crypto. This is asset management, evolved for the future, global, onchain financial system.

From Sep 28th to Oct 9th the Cantina team conducted a review of Maple Finance's maple-core-v2, withdrawal-manager-queue, globals-v2, pool-v2, defi-strategies, allocation-strategy and strategy-manager repositories. The commits that were audited, and the final commits containing the fixes, are listed in the table below.

Reviewed Commits

Repository	Audited commit	Final commit (post-fixes)
maple-core-v2	`b60bd5f4`	`835f47da`
withdrawal-manager-queue	`9a49b58f`	`9a49b58f` (unchanged)
globals-v2	`c1b92c14`	`c1b92c14` (unchanged)
pool-v2	`96565d1b`	`96565d1b` (unchanged)
defi-strategies	`6a0e36fa`	`7fe9bdb9`
allocation-strategy	`d4a6d9f3`	`d4a6d9f3` (unchanged)
strategy-manager	`8140a98c`	`8140a98c` (unchanged)

The team identified a total of **9** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	5	3	2
Gas Optimizations	0	0	0
Informational	4	1	3
Total	**9**	**4**	**5**

The Cantina Managed team reviewed the fixes holistically on the final commits listed in the **Reviewed Commits** table above 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, on the audited commits listed in the **Reviewed Commits** table above:

```
├── allocation-strategy
│   └── contracts
│       ├── MapleAllocationStrategy.sol
│       ├── MapleAllocationStrategyFactory.sol
│       ├── MapleAllocationStrategyInitializer.sol
│       ├── MapleAllocationStrategyStorage.sol
│       └── interfaces
│           ├── IMapleAllocationStrategy.sol
│           ├── IMapleAllocationStrategyInitializer.sol
│           ├── IMapleAllocationStrategyStorage.sol
│           └── Interfaces.sol
```

```
├── defi-strategies
│   └── contracts
│       ├── aaveV3
│       │   ├── MapleAaveV3Strategy.sol
│       │   ├── MapleAaveV3StrategyFactory.sol
│       │   ├── MapleAaveV3StrategyInitializer.sol
│       │   └── MapleAaveV3StrategyStorage.sol
│       └── sky
│           ├── MapleSkyStrategy.sol
│           ├── MapleSkyStrategyFactory.sol
│           ├── MapleSkyStrategyInitializer.sol
│           └── MapleSkyStrategyStorage.sol
├── globals-v2
│   └── contracts
│       ├── GovernorTimelock.sol
│       └── MapleGlobals.sol
├── pool-v2
│   └── contracts
│       ├── MaplePool.sol
│       ├── MaplePoolDeployer.sol
│       ├── MaplePoolManager.sol
│       └── proxy
│           ├── MaplePoolManagerFactory.sol
│           ├── MaplePoolManagerInitializer.sol
│           ├── MaplePoolManagerMigratorV500.sol
│           └── MaplePoolManagerStorage.sol
├── strategy-manager
│   └── contracts
│       ├── MapleLiquidityRouter.sol
│       ├── MapleStrategyDeployer.sol
│       ├── MapleStrategyManager.sol
│       ├── MapleStrategyManagerFactory.sol
│       ├── MapleStrategyManagerInitializer.sol
│       └── MapleStrategyManagerStorage.sol
└── withdrawal-manager-queue
    └── contracts
        ├── MapleWithdrawalManager.sol
        ├── proxy
        │   ├── MapleWithdrawalManagerFactory.sol
        │   ├── MapleWithdrawalManagerInitializer.sol
        │   └── MapleWithdrawalManagerStorage.sol
        └── utils
            └── SortedLinkedList.sol
```

2.2 Codebase context

The Modular Release replaces the one-to-one link between a Maple Pool and its strategies with shared strategies. A `MaplePoolManager` can now fund Strategy Managers, which combine capital from several Pool Managers into one underlying position. It does this alongside the strategies it owns directly; after the upgrade, the only one left is the Open Term Loan Manager.

Each modular strategy is a pair of contracts, deployed and wired together in one transaction by `MapleStrategyDeployer`. `MapleStrategyManager` tracks each Pool Manager's position with internal, non-transferable shares, enforces caps and handles redemptions. It never holds assets, and reads `assetsUnderManagement()` and `unrealizedLosses()` directly from its underlying strategy. Funding mints shares against current AUM. Redemptions take two steps, a request and then processing, and pay out at AUM minus unrealized losses.

There are three underlying strategies:

- `MapleAaveV3Strategy` supplies to an Aave V3 (or SparkLend) reserve.
- `MapleSkyStrategy` converts the funds asset to USDS through Sky's LitePSM and deposits it

into sUSDS. It values the position net of the PSM's `tout`.

- `MapleAllocationStrategy` sends capital to Maple-managed wallets that deploy it off-chain. Wallets report yield in cash through `realizeYield()`, which takes fees up front and releases the rest into AUM linearly.

The Aave and Sky strategies charge delegate and protocol fees on growth since the last checkpoint, and pay them out on every interaction.

The release also upgrades existing contracts:

- Pool Manager v500 adds Strategy Manager support, per-strategy caps and a rewritten `requestFunds()`, and drops Pool Delegate Cover and cyclical withdrawal managers.
- `MaplePoolManagerMigratorV500` removes fixed-term loan managers and legacy Aave and Sky strategies during the upgrade.
- `MapleGlobals` adds a Strategy Delegate registry, which binds each delegate to one Strategy Manager, and an allowlist for managed wallets.
- The queue `MapleWithdrawalManager` now checks the Pool Permission Manager in `removeSharesById()`.

2.3 Trust assumptions

- Strategy Delegates are trusted, like Pool Delegates. A delegate sets its own fee rate and can fund and operate its strategy even while it is impaired. The Operational Admin is trusted to reassign a delegate only for an agreed handover.
- Liquidity bots and the Liquidity Router are trusted, Maple-operated infrastructure. Caps limit how much capital they move, not when. Strategy Manager redemptions have no queue and are assumed to be processed in the order they were requested.
- Managed wallets are trusted, Maple-operated addresses. Capital deployed to a wallet leaves on-chain control, and the yield a wallet reports through `realizeYield()` is taken as given, within `yieldCap` and the hourly cooldown.
- Impairments, loss recognition, yield cancellation and Sky's `fallbackTout` are discretionary operator inputs, used without proof or delay. They directly set the price at which pools exit a strategy.
- Aave, Sky's LitePSM and sUSDS are assumed to account correctly. Positions are valued from their live balances and conversion rates, with no independent oracle.
- Allocation yield is released into AUM linearly, so it goes to whoever holds Strategy Manager shares while it vests, not to whoever held them when it was earned. Bots are assumed to avoid funding or redeeming pools on a shared Strategy Manager while a large amount of yield is still vesting.
- Strategy Managers and Pool Managers are assumed to be created only through the Strategy Deployer and Pool Deployer, with a non-zero bootstrap set in Globals. The upgrade procedure is assumed to drain the legacy Aave and Sky strategies before each pool is migrated.

2.4 Cantina Managed Team Statement

Maple engaged with Cantina to conduct a security review of the Modular Release. We would like to thank the Maple team for their responsiveness and constructive engagement throughout the review process. No critical, high or medium-severity issues were identified and all of the findings are low-severity and informational. Provided the trust assumptions hold, we expect the release to operate as intended.

3 Findings

3.1 Low Risk

3.1.1 Small Aave amounts can revert and temporarily block strategy operations

Severity: Low Risk.

Context: MapleAaveV3Strategy.sol#L163, MapleAaveV3Strategy.sol#L433-L437.

Description: In `MapleAaveV3Strategy`, there are two places where passing an extremely small asset amount (i.e. a few wei) to Aave V3 functions would cause a revert:

(1) `fundUnderlyingStrategy()` only checks that `assetsIn_` is non-zero before supplying it to Aave:

```
require(assetsIn_ > 0, "MAV3S:FUS:ZERO_ASSETS");

// ...

IAavePoolLike(aavePool_).supply(fundsAsset_, assetsIn_, address(this),
↪  referralCode);
```

As described under "Very small supplies can revert" in Aave V3 supplier integration notes, Aave converts the supplied assets into scaled aTokens. If a very small positive `assetsIn_`, such as a few wei, converts to zero scaled units, `Pool.supply()` reverts.

Consequently, a sufficiently small call to `fundUnderlyingStrategy()` reverts instead of funding Aave. This is unlikely during normal operation because strategy funding amounts should be materially larger than the rounding threshold, and a failed call can be retried with a larger amount.

(2) `_harvestAndWithdraw()` adds accrued fees to the requested withdrawal and calls Aave whenever the sum is non-zero:

```
(uint256 delegateFee_, uint256 protocolFee_) =
↪  _currentAccruedFees(_currentTotalAssets(aaveToken));

uint256 totalToWithdraw_ = assets_ + delegateFee_ + protocolFee_;

if (totalToWithdraw_ != 0) IAavePoolLike(aavePool).withdraw(fundsAsset_,
↪  totalToWithdraw_, address(this));
```

In Aave V3.4 and earlier, a positive withdrawal reverts when its conversion to scaled units produces zero. This behavior is described under "Ways `Pool.withdraw()` can revert" in Aave V3 supplier integration notes.

This can realistically occur when `_harvestAndWithdraw(0, address(0))` is used only to accrue fees shortly after the previous accrual. In that case, `yieldAccrued_` and the resulting fees can be only a few wei, making `totalToWithdraw_` non-zero but too small to burn any scaled aTokens.

The fee-only path is called by `fundUnderlyingStrategy()`, `setStrategyDelegateFeeRate()`, and `setStrategyProtocolFeeRate()`. All three functions are temporarily unavailable until enough additional fees accrue for the scaled withdrawal amount to become nonzero.

The affected deployment surface is limited: of the three identified V3.4-and-earlier markets, two are frozen. The notable remaining market is Aave Horizon, which initially forked Aave V3.3.

Recommendation: Both issues can be fixed operationally.

(1) can be avoided if the operator only performs strategy deposits above a minimum threshold.

(2) will never occur as long as `MapleAaveV3Strategy` is only used for Aave v3.5+ markets.

Maple Finance: Acknowledged our keeper will have minimums set for deposits and withdrawals alongside this AaveV3 pools we have planned to use are on v3.7.

Cantina Managed: Acknowledged.

3.1.2 `MapleSkyStrategy.availableToWithdraw()` could overestimate withdrawal liquidity by 2 units

Severity: Low Risk.

Context: MapleSkyStrategy.sol#L427-L435.

Description: In `MapleSkyStrategy`, `availableToWithdraw()` converts all sUSDS held to `fundsAsset` via one `previewRedeem()` call, and then subtracts the accrued fees:

```
uint256 currentTotalAssets_ = _currentTotalAssets(psm_, savingsUsds_);
uint256 psmPocketLiquidity_ =
↪  IERC20Like(fundsAsset).balanceOf(IPSMLike(psm_).pocket());

uint256 grossAvailable_ = _min(currentTotalAssets_, psmPocketLiquidity_);

(uint256 delegateFee_, uint256 protocolFee_) =
↪  _currentAccruedFees(currentTotalAssets_);
uint256 totalFees_ = delegateFee_ + protocolFee_;

if (grossAvailable_ > totalFees_) availableToWithdraw_ = grossAvailable_ -
↪  totalFees_;
```

However, withdrawals from Sky in the strategy are performed individually. For example, `withdrawFromUnderlyingStrategy()` performs three separate withdrawals:

1. In `_accrueFees()`, withdraw the delegate fee.
2. In `_accrueFees()`, withdraw the protocol fee.
3. Withdraw `assets_`.

ERC-4626 `withdraw()` rounds required shares upward. Rounding the delegate fee, protocol fee, and principal separately can consume up to two more `sUSDS` share units than rounding their aggregate once. Thus, withdrawing the exact value returned by `availableToWithdraw()` could revert due to insufficient `sUSDS` shares.

For example:

- Assume:
  - The strategy holds 1,000,000 `sUSDS`.
  - Sky's `sUSDS <> USDS` exchange rate is 1.05 `USDS` per `sUSDS`.
  - The delegate fee is 1,000 `USDS` and the protocol fee is 500 `USDS`.
- `previewRedeem(1,000,000e18)` returns 1,050,000 `USDS`.
- `availableToWithdraw()` therefore returns `1,050,000 - 1,000 - 500 = 1,048,500` `USDS`.
- Executing `withdrawFromUnderlyingStrategy()` with `assets_ = 1,048,500e18` requires three separately rounded share amounts:

```
delegate: ceil(1,000e18 / 1.05)       =        952.380952380952380953 sUSDS
protocol: ceil(500e18 / 1.05)         =        476.190476190476190477 sUSDS
principal: ceil(1,048,500e18 / 1.05)  =    998,571.428571428571428572 sUSDS
                                          --------------------------------
total:                                  1,000,000e18 + 2 sUSDS
```

The strategy owns only `1,000,000e18` `sUSDS`, so the split withdrawals require 2 more share units and `withdrawFromUnderlyingStrategy()` reverts.

Recommendation: The simplest fix is operational - avoid withdrawing exactly `availableToWithdraw()` when calling `withdrawFromUnderlyingStrategy()`. Under normal operation, this shouldn't occur since a single pool will usually not own all assets in a strategy deployed under the strategy manager.

Otherwise, refactor the strategy to perform fee and asset withdrawals together, in a single call to `sUSDS.withdraw()`, similar to `MapleAaveV3Strategy._harvestAndWithdraw()`.

Maple Finance: Fixed in commit 7fe9bdb.

Cantina Managed: Verified, assets and fees are now withdrawn from Sky in a single call.

3.1.3 Sky PSM configuration could block `MapleSkyStrategy` operations

Severity: Low Risk.

Context: *(No context files were provided by the reviewer)*.

Description: There are two ways in which `MapleSkyStrategy` operations could be blocked due to changes in the configuration of `psm_` by Sky governance, which are currently not checked for.

`fundUnderlyingStrategy()` relies on `sellGem()` to reject a halted `tin`: `DssLitePsm.sellGem()` reverts if `tin == type(uint256).max`:

```
function sellGem(address usr, uint256 gemAmt) external returns (uint256 daiOutWad) {
    uint256 tin_ = tin;
    require(tin_ != HALTED, "DssLitePsm/sell-gem-halted");
    daiOutWad = _sellGem(usr, gemAmt, tin_);
}
```

`fundUnderlyingStrategy()` checks whether `tout` is set to `type(uint256).max`, but does not perform the equivalent check for `tin` before calling `sellGem()`:

```
require(IPSMLike(psm_).tout() != type(uint256).max, "MSS:FUS:PSM_HALTED");

// ...

uint256 usdsOut_ = IPSMLike(psm_).sellGem(address(this), assetsIn_);
```

Therefore, if `tin` is set to `type(uint256).max` in the PSM, deposits in `MapleSkyStrategy` are blocked.

This does not create a liveness issue. The revert propagates through the wrapper and prevents the strategy from being funded, which is the desired behavior.

A caged `DaiJoin` would allow deposits but prevent withdrawals: `_withdraw()` calls the PSM wrapper's `buyGem()` function without checking the liveness of the legacy `DaiJoin`:

```
sharesBurned_ = IERC4626Like(savingsUsds_).withdraw(requiredUsds_, address(this),
↪  address(this));

require(ERC20Helper.approve(usds, psm_, requiredUsds_), "MSS:W:USDS_APPROVE_FAIL");

IPSMLike(psm_).buyGem(destination_, assets_);
```

`UsdsPsmWrapper.buyGem()` calls `legacyDaiJoin.exit()`, which requires `DaiJoin.live == 1`:

```
function exit(address usr, uint wad) external {
    require(live == 1, "DaiJoin/not-live");
    vat.move(msg.sender, address(this), mul(ONE, wad));
```

```
        dai.mint(usr, wad);
        emit Exit(usr, wad);
    }
```

In contrast, the wrapper's deposit path calls `legacyDaiJoin.join()`, which does not check `live`. If `DaiJoin.cage()` were called without also halting the PSM, the strategy could continue accepting deposits while all withdrawals would revert.

Recommendation: For the second issue, a possible fix would be to block deposits when withdrawals are not possible, by adding a `DaiJoin.live == 1` check in `fundUnderlyingStrategy()`.

Maple Finance: Fixed in commit e25a820.

Cantina Managed: Verified, the recommended fix was implemented. `availableToWithdraw()` now also returns `0` when `live != 1`.

3.1.4 Unbounded `fallbackTout` can make Sky valuation revert for every attached pool

Severity: Low Risk.

Context: MapleSkyStrategy.sol#L253.

Description: `setFallbackTout()` only rejects `type(uint256).max` (L244-L258), while `LitePSM` itself only accepts a `tout` up to `WAD` (DssLitePsm L288). During a halt, `_gemForUsds()` computes `to18ConversionFactor * (WAD + fallbackTout)` (L459-L464). That overflows for large values, for example `type(uint256).max - 1`.

`assetsUnderManagement()` and `unrealizedLosses()` then revert. That makes `MaplePoolManager.totalAssets()` revert (L474-L482), which blocks deposits and withdrawals in every pool attached to the Strategy Manager. The Maple Sky docs mention that `fallbackTout` exists so that AUM doesn't revert during a halt, and Pool Manager invariant C mentions that `totalAssets()` never reverts. Smaller values above `WAD` don't revert, but they aren't valid `LitePSM` fees and only misvalue the position.

Recommendation: Add `require(fallbackTout_ <= 1e18)`, matching `LitePSM`.

Maple Finance: Fixed in commit badb46e.

Cantina Managed: Verified the fixes, `fallbackTout` now has a cap of `WAD`.

3.1.5 `removeStrategyManager()` checks the rounded position value instead of the share balance

Severity: Low Risk.

Context: *(No context files were provided by the reviewer).*

Description: `removeStrategyManager()` only requires `assetsOf(address(this)) == 0` (L217), which the docs describe as the pool's position being zero. `assetsOf()` rounds down, so it is also zero when the pool still holds dust shares, or when the strategy's AUM is zero after a full write-off.

After removal those shares still exist, but they are no longer counted in the pool's `totalAssets()`.

Recommendation: Consider requiring the pool's share balance to be zero, however with this mitigation it should be ensured strategy removal is not blocked. Otherwise, document this as a known issue.

Maple Finance: This is acknowledged and by design. Whilst it may be possible for a Pool to have shares in a strategy and `assetsOf()` rounds down to 0 this is required for us to be able to remove strategies without being blocked over if we checked for shares to be zero as well. It is valid for a strategy have a loss leading to shares being worthless and needing to be removed from a Pool.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Outstanding shares after a total loss permanently block `MapleStrategyManager` deposits

Severity: Informational.

Context: MapleStrategyManager.sol#L418-L424.

Description: `MapleStrategyManager.fundStrategyManager()` records the strategy's AUM before funding and uses it to calculate the shares minted for the deposit:

```
uint256 totalAssetsBefore_ = assetsUnderManagement();

IPoolManagerLike(poolManager_).requestFunds(underlyingStrategy_, assetsIn_);

uint256 assetsDeposited_ =
↪   IUnderlyingStrategyLike(underlyingStrategy_).fundUnderlyingStrategy(assetsIn_,
↪   data_);
uint256 sharesMinted_    = _convertToShares(assetsDeposited_, totalAssetsBefore_);
```

`_convertToShares()` only checks if `totalShares` is zero:

```
function _convertToShares(uint256 assets_, uint256 totalAssets_) internal view
↪   returns (uint256 shares_) {
    uint256 totalShares_ = totalShares;

    if (totalShares_ == 0) return assets_;

    shares_ = assets_ * totalShares_ / totalAssets_;
}
```

Assume a `MapleStrategyManager` has the following state:

- `totalShares != 0`
- The underlying strategy has an unrealized loss greater or equal to 100% of AUM.

If it realizes that loss, `assetsUnderManagement()` becomes zero while `totalShares` remains non-zero. All future deposits via `fundStrategyManager()` revert due to a division-by-zero in `_convertToShares()`, so the strategy manager becomes permanently DOSed.

This could realistically occur if the amount of assets in the strategy is small.

Recommendation: Strategy managers whose underlying strategy realizes a complete loss must be shut down and removed from all pools permanently. This is enforced in code of the `Aave` and `Sky` strategies. The allocation strategy does not enforce this, therefore, it must be handled operationally.

Maple Finance: Acknowledged and will be clearly documented as the recommendation is the intended procedure.

Cantina Managed: Acknowledged.

3.2.2 `MapleSkyStrategy._gemForUsds()` could underestimate redeemable assets by one unit

Severity: Informational.

Context: MapleSkyStrategy.sol#L459-L464.

Description: `MapleSkyStrategy._gemForUsds()` estimates the amount of `fundsAsset` that will be received from `buyGem()` for a given `usdsAmount_`, by applying the conversion factor and fee in one calculation:

```
function _gemForUsds(address psm_, uint256 usdsAmount_) internal view returns
↪   (uint256 gemAmount_) {
    uint256 tout_                 = _getTout(psm_);
    uint256 to18ConversionFactor_ = IPSMLike(psm_).to18ConversionFactor();

    gemAmount_ = (usdsAmount_ * WAD) / (to18ConversionFactor_ * (WAD + tout_));
}
```

However, `UsdsPsmWrapper.buyGem()` first rounds the fee down and then adds it to the base `USDS` amount:

```
uint256 gemAmt18 = gemAmt * to18ConversionFactor;
usdsInWad = gemAmt18 + gemAmt18 * psm.tout() / WAD;
```

When the fee calculation has a remainder, the PSM can therefore sell one more gem unit than `_gemForUsds()` reports for the same `USDS` amount. For example, consider the following example, which has:

- Gem as `USDC`, which means `to18ConversionFactor = 1e12`.
- A fee of approximately 0.1% (`tout = 0.001e18 + 500_000`).

For `gemAmt = 100e6 + 1`, the calculation in `buyGem()` would be:

```
gemAmt = 100e6 + 1
gemAmt18 = (100e6 + 1) * 1e12 = (100e18 + 1e12)

usdsInWad = (100e18 + 1e12) + (100e18 + 1e12) * (0.001e18 + 500_000) / 1e18 =
↪   100_100_001_001_050_000_000
```

For the same `USDS` amount, the calculation in `_gemForUsds()` is:

```
(100_100_001_001_050_000_000 * 1e18) / (1e12 * (1e18 + (0.001e18 + 500_000))) =
↪   100_000_000  // 100 USDC
```

Thus, `buyGem()` can purchase `100.000001` `USDC` with the supplied `USDS`, while `_gemForUsds()` reports only `100` `USDC`, underestimating the position by one `USDC` unit.

Since `_gemForUsds()` is used to calculate the strategy's current total assets, `assetsUnderManagement()` and `availableToWithdraw()` can be understated by one `fundsAsset` unit. However, this does not seem to have more severe impacts since underestimating both values are safe.

Recommendation: After calculating the conservative estimate, check whether the available `USDS` can purchase one additional gem unit using the PSM's exact rounding order:

```
function _gemForUsds(address psm_, uint256 usdsAmount_) internal view returns
↪   (uint256 gemAmount_) {
    uint256 tout_                 = _getTout(psm_);
    uint256 to18ConversionFactor_ = IPSMLike(psm_).to18ConversionFactor();

    gemAmount_ = (usdsAmount_ * WAD) / (to18ConversionFactor_ * (WAD + tout_));

    uint256 nextGemAmt18_ = (gemAmount_ + 1) * to18ConversionFactor_;
    uint256 nextUsdsAmount_ = nextGemAmt18_ + nextGemAmt18_ * tout_ / WAD;

    if (nextUsdsAmount_ <= usdsAmount_) gemAmount_++;
}
```

Maple Finance: Acknowledged, we'll keep the code as is. Our view is in the case of a +ve tout the strategy will not be used and unwound and as the rounding causes 1 wei of precision loss this is

acceptable without over complicating the calculation.

Cantina Managed: Acknowledged.

3.2.3 Sky fee setters can still move the fee checkpoint while the PSM is halted

Severity: Informational.

Context: MapleSkyStrategy.sol#L307.

Description: While the PSM is halted, funding and withdrawals revert, but both fee setters still reset `lastRecordedTotalAssets` to a `fallbackTout`-based valuation when no fees are due (L307, L331). When `_accrueFees()` has a non zero fee to pay, they revert with an arithmetic panic in `_usdsForGem()` (L477-L483) instead of a clear error.

Recommendation: Block both fee setters while the PSM is halted, as `fundUnderlyingStrategy()` already does (L148).

Maple Finance: Fixed in commit 46378ce0.

Cantina Managed: Verified the fixes.

3.2.4 Batched Minor Issues

Severity: Informational.

Context: *(No context files were provided by the reviewer)*.

Description:

1. **`_usdsForGem()` overflows during a PSM halt.** It reads the raw `tout()` instead of `fallbackTout` (L477-L483), so a fee harvest during a halt reverts with an arithmetic panic rather than a clear error.
2. **A `Strategy Manager` is stuck if its delegate is de-listed before activation.** De-listing is only blocked once the `Strategy Manager` is active (L294). Before that, it succeeds, and the `Strategy Manager` can't be activated (L169) or handed over (L416) until the same address is re-listed.

Recommendation:

1. Revert with a clear error in `_usdsForGem()` while the PSM is halted.
2. Let the OA hand over an inactive `Strategy Manager`.

Maple Finance:

1. We are happy with the arithmetic panic whilst the PSM is paused, our system won't be sending txns during this period and a PSM pause will be monitored.
2. This will be operationally managed.

Cantina Managed: Acknowledged.

4 Appendix

4.1 Aave V3 supplier integration notes

4.1.1 Overview

This document covers Aave V3.0.x through V3.7. It assumes that a strategy:

- supplies underlying assets through `Pool.supply()`;
- holds aTokens to earn interest;
- withdraws through `Pool.withdraw()`; and
- uses `aToken.balanceOf(address(this))` to value its position.

Check the implementation deployed on the target market. Aave markets can be upgraded at different times, so the repository's latest version may not match a particular deployment.

4.1.2 Version summary

Version	aToken rounding	Available-liquidity accounting
V3.0.x	Round half up	Physical underlying balance of the aToken
V3.1-V3.3	Round half up	Virtual accounting is optional per reserve
V3.4	Round half up	Virtual accounting is always active
V3.5-V3.7	Supply and balances round down; withdrawals round up	Virtual accounting is always active

V3.0.x refers to the final generation in the archived `aave-v3-core` repository. V3.1 and later refer to `aave-v3-origin`.

4.1.3 Rounding rules

V3.0.x-V3.4: round half up: These versions round each conversion to the nearest integer. An exact half rounds up.

```
scaled amount minted on supply  = roundHalfUp(amount * RAY / index)
visible aToken balance           = roundHalfUp(scaled balance * index / RAY)
scaled amount burned on withdraw = roundHalfUp(amount * RAY / index)
```

See the V3.4 scaled mint and burn code and `WadRayMath.rayDiv()`.

V3.5-V3.7: explicit rounding:

```
RAY = 1e27

scaled amount minted on supply  = floor(amount * RAY / index)
visible aToken balance           = floor(scaled balance * index / RAY)
scaled amount burned on withdraw = ceil(amount * RAY / index)
```

See V3.5 `TokenMath`.

Supply can mint a different amount of visible value: Applies to: V3.0.x-V3.7, with different rounding behavior.

Version	Behavior
V3.0.x-V3.4	The visible aToken balance increase can be slightly less or slightly more than the supplied amount.

V3.5-V3.7	The scaled amount is rounded down. The visible balance increase does not receive the older round-up benefit and can be less than the supplied amount.

Do not assume:

```
aToken balance increase == supplied amount
```

Measure the balance change if the exact result matters.

Very small supplies can revert: Applies to: V3.0.x-V3.7, with different thresholds and errors.

Version	Behavior
V3.0.x-V3.3	Half-up conversion that produces zero scaled aTokens reverts with numeric code `24` (`InvalidMintAmount`).
V3.4	The same half-up conversion reverts with `InvalidMintAmount()`.
V3.5-V3.7	Floor conversion that produces zero scaled aTokens is rejected by the Pool with `InvalidAmount()`.

The V3.5-V3.7 rule is stricter. For example, when the index is just above `1e27`, supplying one smallest unit produces zero shares in V3.5-V3.7. Under half-up rounding, it normally still produces one share until the index is greater than about `2e27`.

Partial withdrawals can change the aToken balance by a different amount: Applies to: V3.0.x-V3.7, with different rounding behavior.

Version	Behavior
V3.0.x-V3.4	The scaled burn uses half-up rounding. The claim value burned can be slightly less or slightly more than the underlying received. A positive amount that converts to zero scaled units reverts with `InvalidBurnAmount`.
V3.5-V3.7	The scaled burn rounds up. The aToken balance can fall by slightly more than the underlying received. A positive explicit amount always converts to at least one scaled unit.

Do not assume:

```
aToken balance decrease == withdrawn amount
```

This matters most for very small withdrawals. Batch withdrawals where possible.

Withdrawing everything: Applies to: V3.0.x-V3.7. The implementation changed in V3.5.

Passing `type(uint256).max` to `Pool.withdraw()` requests the user's entire position.

Version	Behavior
V3.0.x-V3.4	The Pool calculates the user's visible balance with half-up rounding, uses it as the withdrawal amount, and converts it back to scaled units during the burn. With a normal liquidity index of at least `RAY`, this round trip burns the full scaled balance.
V3.5-V3.7	The Pool burns the exact stored scaled balance and calculates the underlying amount with floor rounding. This avoids a second scaled conversion.

See the V3.4 full-withdraw path and the V3.5 full-withdraw path.

Calling `withdraw(aToken.balanceOf(user))` in the same state should also burn the full scaled balance in all reviewed versions.

`balanceOf()` rounding: Applies differently across versions.

Version	Behavior
V3.0.x-V3.4	`balanceOf()` rounds half up. It can slightly overstate or understate the exact mathematical value. There is no fixed ordering between the sum of user balances and `totalSupply()`.
V3.5-V3.7	`balanceOf()` and `totalSupply()` round down. Each visible balance is conservative, and `sum(balanceOf(users)) <= totalSupply()`.

Scaled accounting remains exact. If a strategy must never overstate assets on an older deployment, it should not assume that the old `balanceOf()` result is a lower bound.

4.1.4 Other integration pitfalls

First supply may enable collateral: Applies to: V3.0.x-V3.7, but collateral eligibility changed.

A first supply can automatically mark the asset as collateral when the asset is eligible for the user's current configuration.

Version	Important differences
V3.0.x-V3.5	Eligibility uses the older reserve-level LTV, isolation-mode, and eMode rules. Reserve-isolated collateral is not normally auto-enabled. Some older releases allow a permissioned supplier role to auto-enable it.
V3.6	Supply still auto-enables eligible collateral, but reserve-isolated collateral is never auto-enabled. Effective LTV and LTV-zero status can depend on the user's eMode.
V3.7	Reserve isolation mode was removed. Supply auto-enables only when the asset has a nonzero effective LTV. In an isolated eMode, assets outside its collateral bitmap have effective LTV zero and are not auto-enabled.

See the V3.6 automatic-collateral changes and the V3.7 isolated-eMode rules.

V3.5 also fixed an older collateral-flag edge case. In V3.0.x-V3.4, a partial withdrawal could round to all remaining shares without clearing the collateral flag because the requested amount was not equal to the previously calculated visible balance. V3.5-V3.7 clear the flag based on whether the scaled balance is actually zero after the burn.

If the same strategy address has Aave debt, withdrawals may be blocked by health-factor or LTV-zero rules. A dedicated supplier strategy should normally avoid borrowing from the same address.

Raw aToken-contract balance is not always Aave liquidity: Applies differently across versions.

```
IERC20(asset).balanceOf(aToken)
```

Version	Which balance represents accounted available liquidity?
V3.0.x	The physical underlying balance of the aToken. Virtual accounting does not exist. Direct donations increase the liquidity seen by the standard interest-rate strategy, although they do not mint aTokens.
V3.1-V3.3	Check `getIsVirtualAccActive(asset)`. If true, use `getVirtualUnderlyingBalance(asset)`. If false, the reserve uses its special non-virtual behavior and the virtual value should not be treated as available liquidity.
V3.4-V3.7	Use `getVirtualUnderlyingBalance(asset)`. Virtual accounting is always active. Direct donations increase physical cash but not Aave-accounted liquidity.

V3.1 introduced optional virtual accounting. V3.4 made it active for every reserve. See the V3.1 design and V3.4 virtual-accounting change.

For V3.4-V3.7, the physical balance should normally be at least the virtual balance. This is one of the Pool's invariants.

No built-in slippage arguments: Applies to: V3.0.x-V3.7.

`Pool.supply()` and `Pool.withdraw()` do not accept minimum-share or maximum-share arguments.

If the strategy needs this protection, it can measure balances before and after the call and revert when the result is outside its allowed range. Reverting the strategy transaction also reverts the Aave operation.

4.1.5 Ways `Pool.supply()` can revert

Aave validation and rounding errors:

Semantic error	Applicable versions	Cause and version differences
`InvalidAmount`	V3.0.x-V3.7	The input amount is zero. In V3.5-V3.7, this also covers a positive supply whose floor-rounded scaled amount is zero.
`InvalidMintAmount`	V3.0.x-V3.4	A positive supply's half-up scaled amount is zero. The Pool catches this earlier as `InvalidAmount()` from V3.5 onward.
`ReserveInactive`	V3.0.x-V3.7	The reserve is inactive.
`ReservePaused`	V3.0.x-V3.7	The reserve is paused.
`ReserveFrozen`	V3.0.x-V3.7	The reserve is frozen.
`SupplyToAToken`	V3.1-V3.7	`onBehalfOf` is the reserve's own aToken address. V3.0.x has no guard and can mint aTokens to the aToken itself.
`SupplyCapExceeded`	V3.0.x-V3.7	The supply would put the reserve over its supply cap.

Configured Aave component failure: Applies to: V3.0.x-V3.7.

Supply can also revert if:

- the interest-rate strategy reverts;
- the incentives or rewards controller reverts while handling the aToken mint;
- the aToken or variable-debt token implementation is broken or misconfigured; or
- an upgrade introduces incompatible behavior.

V3.4 made the standard rewards controller and several other addresses immutable, but external component calls can still fail.

4.1.6 Ways `Pool.withdraw()` can revert

Aave validation and rounding errors:

Semantic error	Applicable versions	Cause and version differences
`WithdrawToAToken`	V3.1-V3.7	The receiver is the reserve's own aToken address. V3.0.x has no guard: the burn can succeed while the aToken deliberately skips transferring underlying to itself.
`InvalidAmount`	V3.0.x-V3.7	The explicit amount is zero, or a full withdrawal is requested by an account with no balance.

`InvalidBurnAmount`	V3.0.x-V3.4	A positive withdrawal's half-up conversion produces zero scaled units. V3.5-V3.7 use ceiling conversion, so a positive explicit withdrawal cannot round to zero.
`NotEnoughAvailableUserBalance`	V3.0.x-V3.7	The requested withdrawal exceeds the user's position. V3.0.x-V3.4 compare unscaled half-up balances; V3.5-V3.7 compare the ceiling-rounded scaled burn against the stored scaled balance.
`ReserveInactive`	V3.0.x-V3.7	The reserve is inactive.
`ReservePaused`	V3.0.x-V3.7	The reserve is paused.
`HealthFactorLowerThan LiquidationThreshold`	V3.0.x-V3.7	The withdrawal would make an account with debt unhealthy.
`LtvValidationFailed`	V3.0.x-V3.7	LTV-zero collateral must be withdrawn or disabled before non-LTV-zero collateral. V3.6 makes effective LTV-zero status eMode-specific. V3.7 also treats assets outside an isolated eMode's collateral bitmap as LTV-zero..

A frozen reserve does not block withdrawals in any reviewed version.

For a dedicated supplier strategy with no Aave debt, the health-factor and LTV errors should not occur.

Not enough accounted liquidity: Applies to: V3.0.x-V3.7, but the liquidity source changed.

Version	Failure path
V3.0.x	Liquidity is based on the physical underlying balance. The standard interest-rate calculation or the final token transfer can revert when cash is insufficient. There is no dedicated "not enough withdrawal liquidity" error.
V3.1-V3.3	For a reserve with virtual accounting enabled, subtracting more than its virtual balance can cause an arithmetic panic. For a non-virtual reserve, failure depends on its special rate strategy and the physical transfer.
V3.4-V3.7	If the amount is greater than `getVirtualUnderlyingBalance(asset)`, virtual-liquidity subtraction or rate calculation can underflow and produce an arithmetic panic rather than a specific liquidity error.

This can happen even when `aToken.balanceOf(strategy)` is large enough, because aToken balance is a claim and not a promise that all underlying cash is currently available.

Configured Aave component failure: Applies to: V3.0.x-V3.7.

Withdrawal can also revert if:

- the interest-rate strategy reverts;
- the incentives or rewards controller reverts while handling the aToken burn;
- the aToken or variable-debt token implementation is broken or misconfigured; or
- an upgrade introduces incompatible behavior.