MorphoWETHLoanProtectionPolicy

Security Review

Cantina Managed review by:
Sujith Somraaj, Lead Security Researcher
Christos Pap, Security Researcher

April 10, 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

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 Absence of post-execution health check allows top-up to feed the liquidator . . . 4
- 3.2 Low Risk . . . 5
  - 3.2.1 Parameter `triggerLtv` lacks minimum buffer from market `lltv`, reducing policy effectiveness . . . 5
  - 3.2.2 No minimum top-up enforcement, executor can waste one-shot with dust amount . . . 6
- 3.3 Informational . . . 6
  - 3.3.1 Third party collateral supply frontrunning can lead to repeated protection policy reverting and delayed execution . . . 6
  - 3.3.2 Parameter `triggerLtv` of zero allows unconditional execution regardless of position health . . . 7
  - 3.3.3 Redundant `idToMarketParams` external call in `_onSingleExecutorInstall` . . . 8
  - 3.3.4 Full reimplementation of parent execute logic in WETH override can lead to silent divergence on future updates . . . 8
  - 3.3.5 `_usedPolicyId` and `_configHashByPolicyId` are never cleared on uninstall . . . 9

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

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From Apr 6th to Apr 8th the Cantina team conducted a review of account-policies on commit hash 13205362. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	5	1	4
Total	**8**	**4**	**4**

2.1 Scope

The security review had the following components in scope for account-policies on commit hash 13205362:

```
src
├── PolicyManager.sol
├── PublicERC6492Validator.sol
└── policies
    ├── MorphoLendPolicy.sol
    ├── MorphoLoanProtectionPolicy.sol
    ├── MorphoWethLoanProtectionPolicy.sol
    ├── Policy.sol
    ├── SingleExecutorAuthorizedPolicy.sol
    ├── SingleExecutorPolicy.sol
    └── accounting
        └── RecurringAllowance.sol
```

3 Findings

3.1 Medium Risk

3.1.1 Absence of post-execution health check allows top-up to feed the liquidator

Severity: Medium Risk

Context: MorphoLoanProtectionPolicy.sol#L244

Description: The `PolicyManager._execute()` implements a three-phase execution flow:

1. Call `policy.onExecute()` to build the account call plan,
2. Execute the call plan against the account.
3. Call `policy.onPostExecute()` for post-execution validation.

This third phase exists specifically to allow policies to verify that the account call achieved its intended outcome.

However, neither `MorphoLoanProtectionPolicy` nor `MorphoWethLoanProtectionPolicy` override `_onPostExecute()` and it remains the base Policy no-op across all five layers of the inheritance chain. The `postCallData` returned by `_onSingleExecutorExecute()` is hardcoded to empty bytes.

As a result, the policy only validates the position's health before the top-up (`currentLtv >= triggerLtv`), but never validates the outcome after the top-up. There is no on-chain guarantee that the supplied collateral actually moves the position out of the liquidation zone.

The NatSpec in `MorphoLoanProtectionPolicy.sol` further understates this risk:

```
/// Even after a successful top-up, the position may become
/// unhealthy again in subsequent blocks.
```

This implies the top-up at least restores the position to a healthy state within the current block, with the risk being future degradation. This is incorrect as the position can remain above LLTV immediately after the top-up within the same transaction, due to market movement between the executor's signature and the transaction's inclusion.

Proof of Concept: The following scenario demonstrates concrete fund loss:

Setup:

- WETH/USDC market: LLTV = 90%, triggerLtv = 80%, maxTopUpAssets = 10 WETH.
- Position: 100 WETH collateral, 72,000 USDC debt.
- WETH = $1,000 → LTV = 72,000 / 100,000 = 72% (healthy).

Step 1: Trigger detected:

- WETH drops to $900 → collateral value = $90,000.
- LTV = 72,000 / 90,000 = 80% → hits triggerLtv.
- Executor signs a 10 WETH top-up and submits the transaction.

Step 2: Price moves before inclusion:

- Before the transaction is mined, WETH crashes to $700.
- Collateral value = 100 × $700 = $70,000.
- LTV = 72,000 / 70,000 = 102.9% → already above LLTV, position is liquidatable.

Step 3: Top-up executes anyway:

- The pre-execution LTV check passes: 102.9% >= 80% triggerLtv .
- The one-shot is consumed: _usedPolicyId[policyId] = true.
- 10 ETH is wrapped to WETH, approved, and supplied as collateral to Morpho.
- Post top-up state: 110 WETH at $700 = $77,000 collateral value.

- New LTV = 72,000 / 77,000 = 93.5% → still above LLTV (90%).
- No post-execution check exists to catch this as `_onPostExecute()` is a no-op.

Step 4: Liquidation seizes the freshly added collateral:

- The position remains liquidatable immediately after the top-up.
- A liquidator seizes collateral, including the 10 WETH just supplied.
- The one-shot is permanently consumed as no further protection is available.

Without the top-up, the user keeps 10 ETH in their wallet. Only the existing 100 WETH collateral is exposed to the liquidator. With the top-up, the user loses an additional 10 ETH that is seized by the liquidator for zero benefit.

Recommendation: Use the existing `_onPostExecute()` hook to verify the position exited the liquidation zone after the top-up. Return the necessary context (market ID, market params) via `postCallData` from `_onSingleExecutorExecute()`, and override `_onPostExecute()` in `MorphoLoanProtectionPolicy` to recompute the post-top-up LTV and revert if it remains at or above `lltv`.

This causes the entire transaction to revert and the ETH stays in the wallet, the one-shot is preserved, and the executor can retry with a larger amount or the account can take alternative action.

Additionally, update the NatSpec disclaimer to accurately reflect that a successful top-up does not guarantee the position becomes healthy within the current block.

Coinbase: Fixed in commit 6b0f3f8.

Cantina Managed: Fix verified. The top-up flow now enforces post-execution position health via `_onPostExecute()`, reverting the transaction when the supplied collateral is insufficient to bring the LTV below `lltv`.

3.2 Low Risk

3.2.1 Parameter `triggerLtv` lacks minimum buffer from market `lltv`, reducing policy effectiveness

Severity: Low Risk

Context: MorphoLoanProtectionPolicy.sol#L163

Description: The function `_onSingleExecutorInstall()` in `MorphoLoanProtectionPolicy` is responsible for setting the policy's initial parameters at installation. However, it only enforces a single constraint on `triggerLtv`: that it must be strictly less than the market's `lltv`.

```
if (config.triggerLtv >= marketParams.lltv) revert
↪  TriggerLtvAboveLltv(config.triggerLtv, marketParams.lltv);
```

This effectively permits configurations where `triggerLtv` is set extremely close to `lltv` (e.g., 85.9999% when `lltv` is 86%). In such cases, the executor has almost no practical window to react detecting the risky position, signing the intent, and submitting the transaction before the position becomes liquidatable.

Even minor changes, such as a single block's interest accrual or an oracle price update, can push the LTV past the threshold. As a result, the protection mechanism becomes unreliable: it may trigger too late (after liquidation eligibility) or fail to trigger altogether if the LTV jumps directly beyond the configured threshold.

Recommendation: Enforce a minimum buffer (e.g., 5% in WAD) between `triggerLtv` and `lltv` at install time:

```
uint256 public constant MIN_LTV_BUFFER = 0.05e18;

if (config.triggerLtv + MIN_LTV_BUFFER > marketParams.lltv)
    revert TriggerLtvTooCloseToLltv(config.triggerLtv, marketParams.lltv);
```

This ensures the executor has a meaningful reaction window between the trigger firing and the position becoming liquidatable.

Coinbase: Fixed in commit b5f298ee.

Cantina Managed: Verified fix. The proportional ratio approach is better than the absolute buffer we recommended. An additional recommendation is to enforce constructor bounds on `MAX_TRIGGER_LTV_RATIO` (for example `0 < ratio < 1e18`) and keep an explicit `triggerLtv < lltv` install time check as defense in depth, so misconfiguration cannot weaken the buffer guarantee.

3.2.2 No minimum top-up enforcement, executor can waste one-shot with dust amount

Severity: Low Risk

Context: MorphoWethLoanProtectionPolicy.sol#L125

Description: In the function `_onSingleExecutorExecute()`, the top-up amount is validated as:

```
if (topUpAssets == 0) revert ZeroAmount();
if (topUpAssets > config.maxTopUpAssets) revert TopUpAboveMax(topUpAssets,
↪  config.maxTopUpAssets);
```

There is a ceiling (maxTopUpAssets) but no floor. The executor chooses `topUpAssets` freely and signs over it the config gives the account no way to enforce a minimum useful amount. A malicious or buggy executor can sign a top-up of 1 wei, which:

1. Passes all validation (non-zero, under max, LTV above trigger).
2. Permanently consumes the one-shot (`_usedPolicyId[policyId] = true`).
3. Has negligible effect on the position's LTV.

The account loses their protection and must uninstall, re-install (or replace), and obtain a new executor signature to restore coverage all while the position remains at risk.

Recommendation: Rather than a static `minTopUpAssets` (which is hard to size correctly across market conditions), consider enforcing a target LTV in the config that the post-top-up position must satisfy. The `_onPostExecute()` hook (currently unused) can verify that the position's LTV after the top-up is below a configured threshold (e.g., `triggerLtv` or a separate `targetLtv`). If the top-up is insufficient to reach the target, the entire transaction reverts, preserving the one-shot and the account's ETH.

Coinbase: Fixed in commit 6b0f3f8.

Cantina Managed: Verified fix. This only protects against dust top-ups when the position is near or above LLTV. If the position is at 80% LTV (well below 90% LLTV), a dust top-up would still pass the post-hook (post-top-up LTV ≈ 80%, still below LLTV). But in that scenario, the one-shot waste is less harmful and the position isn't at immediate liquidation risk. The fix handles the dangerous case correctly.

3.3 Informational

3.3.1 Third party collateral supply frontrunning can lead to repeated protection policy reverting and delayed execution

Severity: Informational

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

Description: Morpho Blue's `supplyCollateral` accepts an `onBehalf` parameter, allowing anyone to supply collateral to any account's position. A malicious actor can monitor the mempool for the executor's `PolicyManager.execute` transaction and front-run it by supplying a small amount of collateral to the target account's Morpho position. This temporarily lowers the account's LTV below `triggerLtv`, causing the protection policy to revert with `HealthyPosition`. Although the one shot is not consumed (as the entire transaction reverts), the protection is delayed. If repeated, the position can reach LLTV and be liquidated while the policy never successfully fires.

The pre-execution LTV check in both `MorphoLoanProtectionPolicy` and `MorphoWethLoanProtectionPolicy` compares the current on-chain LTV at execution time against `triggerLtv`:

`src/policies/MorphoLoanProtectionPolicy.sol`

```
uint256 currentLtv = _computeCurrentLtv(config, marketParams, account);
if (currentLtv < config.triggerLtv) revert HealthyPosition(currentLtv,
↪  config.triggerLtv);
```

The `_computeCurrentLtv` function reads the position's collateral directly from Morpho Blue's state, which reflects any collateral supplied by third parties in the same block:

`src/policies/MorphoLoanProtectionPolicy.sol`

```
morphoBlue.accrueInterest(marketParams);

Position memory position = morphoBlue.position(config.marketId, account);
Market memory market = morphoBlue.market(config.marketId);

uint256 collateralBefore = uint256(position.collateral);
```

Morpho Blue's `supplyCollateral` is permissionless for the `onBehalf` parameter and anyone can supply collateral to any position:

`IMorphoBlue.supplyCollateral`

```
function supplyCollateral(
    MarketParams calldata marketParams,
    uint256 assets,
    address onBehalf,   // ← any address, no authorization required
    bytes calldata data
) external;
```

Recommendation: It's recommended to configure the executor bot to use private mempools (for example Flashbots Protect) to prevent front-running visibility, and implement retry logic on `HealthyPosition` reverts, and consider adding a small buffer below `triggerLtv` in the LTV check to absorb minor collateral donations without reverting.

Coinbase: Acknowledged, no-op. We noted this scenario with our product team.

Cantina Managed: Acknowledged.

3.3.2 Parameter `triggerLtv` of zero allows unconditional execution regardless of position health

Severity: Informational

Context: MorphoWethLoanProtectionPolicy.sol#L128

Description: In function `_onSingleExecutorExecute()` of `MorphoWethLoanProtectionPolicy`, the LTV trigger check is:

```
if (currentLtv < config.triggerLtv) revert HealthyPosition(currentLtv,
↪  config.triggerLtv);
```

Since `currentLtv` is a `uint256`, the condition `currentLtv < 0` is never true. A `triggerLtv` of zero therefore means the executor can trigger the one-shot top-up at any time, even when the position is perfectly healthy. The install validation only enforces `triggerLtv < lltv`, zero passes this check for any valid market.

An account setting `triggerLtv = 0` gives the executor unconditional authority to wrap the account's ETH into collateral at any moment.

Recommendation: Enforce `triggerLtv > 0` at install time in `_onSingleExecutorInstall()`, or document that zero means "always trigger" so account signers are aware of the semantics.

Coinbase: Fixed in PR 28.

Cantina Managed: Fix verified. `_onSingleExecutorInstall()` now rejects `triggerLtv = 0` at install time, and the same guard is inherited by the `MorphoWethLoanProtectionPolicy` path.

3.3.3 Redundant `idToMarketParams` external call in `_onSingleExecutorInstall`

Severity: Informational

Context: MorphoWethLoanProtectionPolicy.sol#L96

Description: In `MorphoWethLoanProtectionPolicy._onSingleExecutorInstall`, `IMorphoBlue(MORPHO).idToMarketParams(config.marketId)` is called twice during installation: once inside `super._onSingleExecutorInstall` (via `_requireMarketParams`) and again immediately after to check `collateralToken == WETH`. The second call is redundant since the market params are immutable in Morpho Blue and were already fetched and validated by the parent.

The parent call fetches and validates market params:

`src/policies/MorphoLoanProtectionPolicy.sol`

```
function _onSingleExecutorInstall(...) internal override {
    // ...
    MarketParams memory marketParams = _requireMarketParams(config.marketId);
    // _requireMarketParams calls IMorphoBlue(MORPHO).idToMarketParams(config.marketId)
    ↪  internally
}
```

The child calls it again to check WETH:

`src/policies/MorphoWethLoanProtectionPolicy.sol`

```
function _onSingleExecutorInstall(...) internal override {
    super._onSingleExecutorInstall(policyId, account, singleExecutorConfig,
    ↪  policySpecificConfig);

    LoanProtectionPolicyConfig memory config = abi.decode(policySpecificConfig,
    ↪  (LoanProtectionPolicyConfig));
    MarketParams memory marketParams =
    ↪  IMorphoBlue(MORPHO).idToMarketParams(config.marketId); // ← redundant external
    ↪  call
    if (marketParams.collateralToken != WETH) revert
    ↪  CollateralNotWeth(marketParams.collateralToken, WETH);
}
```

Recommendation: It's recommended to keep the current implementation as-is. Given the modular design where the child override delegates to `super` and then adds its own validation, refactoring to pass the already-fetched `marketParams` down would require changing the parent's internal API.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3.4 Full reimplementation of parent execute logic in WETH override can lead to silent divergence on future updates

Severity: Informational

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

Description: `MorphoWethLoanProtectionPolicy._onSingleExecutorExecute` completely overrides the parent `MorphoLoanProtectionPolicy._onSingleExecutorExecute` without calling `super`. It reimplements all the mechanisms, including the one-shot guard, config decoding, market param resolution, top-up validation, and LTV trigger check and only differs in the final call plan construction (ETH wrapping instead of zero approve).

Any future security fixes or enhancement to the parent's validation logic must be manually mirrored in the WETH variant, or the child silently diverges. This is not an issue today, as the reimplementation is done correctly, but represents a maintenance risk worth noting.

Recommendation: No changes are needed at this time. It's advised to keep this duplication in mind, if any security fix or validation change is applied to `MorphoLoanProtectionPolicy._onSingleExecutorExecute`

in the future, the same change must be manually applied to `MorphoWethLoanProtectionPolicy._onSingleExecutorExecute` as well.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.3.5 `_usedPolicyId` and `_configHashByPolicyId` are never cleared on uninstall

Severity: Informational

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

Description: When a policy is uninstalled, `_clearInstallState` clears `activePolicyByMarket` and `marketKeyByPolicyId` but does not clear policy-local state such as `_configHashByPolicyId[policyId]`, and (for loan-protection policies) `_usedPolicyId[policyId]`. These storage slots can remain populated after uninstall.

This is not exploitable because `policyId`s are permanently "retired" by the `PolicyManager`: once marked uninstalled, a `policyId` cannot be reinstalled or executed. As a result, stale `_configHashByPolicyId` entries and stale `_usedPolicyId` flags are not reachable through meaningful execution paths.

One minor side effect is that `cancelNonces` remains callable against uninstalled policies, since it only checks `_requireConfigHash` (which can pass against stale config hash state) and `msg.sender == executor`. However, cancelling nonces on an uninstalled policy has no practical effect. Execution is blocked by the `PolicyManager`'s `policyRecord.uninstalled` check before control reaches policy execution logic.

The only real cost is storage growth, which is minimal.

Recommendation: No changes are required from a security perspective. The stale storage is effectively unreachable under the permanent `policyId` model.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged. Stale state is intentionally preserved, as retaining these entries is useful for external monitoring systems.