// Security Assessment

04.06.2026 - 04.13.2026

Position Management Helper - Standalone Contract

Blueprint Finance

HALBORN

Position Management Helper - Standalone Contract - Blueprint Finance

Prepared by: HALBORN

Last Updated 04/17/2026

Date of Engagement: April 6th, 2026 - April 13th, 2026

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
13	0	0	0	9	4

TABLE OF CONTENTS

1. INTRODUCTION

`Blueprint Finance` engaged `Halborn` to perform a security assessment of their smart contracts from April 6th, 2026 to April 10th, 2026. The assessment scope was limited to the smart contracts provided to Halborn. Commit hashes and additional details are available in the Scope section of this report.

The `Blueprint Finance` codebase in scope consists of smart contracts for a position management helper system and unified periphery factory, enabling isolated, role-based leveraged lending and borrowing on Aave V3 via lightweight clones, with support for strategy proxy deployment, upgradeability, and strict zero-balance and access control invariants.

2. ASSESSMENT SUMMARY

`Halborn` was allocated 5 days for this engagement and assigned 1 full-time security engineer to conduct a comprehensive review of the smart contracts within scope. The engineer is an expert in blockchain and smart contract security, with advanced skills in penetration testing and smart contract exploitation, as well as extensive knowledge of multiple blockchain protocols.

The objectives of this assessment are to:

- Identify potential security vulnerabilities within the smart contracts.
- Verify that the smart contract functionality operates as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were partially addressed by the `Blueprint Finance` team. The main recommendations were:

- `Treat a dedicated positionOwner with no other helper-managed or unmanaged Aave activity as a hard operational invariant for each helper; if stronger guarantees are desired, add factory-level uniqueness checks for factory-deployed helpers for the same (positionOwner, collateralToken, borrowToken) tuple.`
- `Update getMaxLtvInWad() and getLiquidationThresholdInWad() to check the positionOwner's active eMode category and, when the helper's collateral and borrow assets are both eligible under that category, return the category-specific LTV and liquidation threshold; otherwise, fall back to the base reserve values.`
- `Apply ceiling rounding consistently to all relevant divisions in both flashWithdrawAmountFromTargetLtv() and flashWithdrawAmount().`

3. TEST APPROACH AND METHODOLODY

`Halborn` conducted a combination of manual code review and automated security testing to balance efficiency, timeliness, practicality, and accuracy within the scope of this assessment. While manual testing is crucial for identifying flaws in logic, processes, and implementation, automated testing enhances coverage of smart contracts and quickly detects deviations from established security best practices.

The following phases and associated tools were employed throughout the term of the assessment:

- Research into the platform's architecture, purpose and use.
- Manual code review and walkthrough of smart contracts to identify any logical issues.
- Comprehensive assessment of the safety and usage of critical Solidity variables and functions within scope that could lead to arithmetic-related vulnerabilities.
- Local testing using custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static security analysis of scoped contracts, and imported functions (`Slither`).

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: earn-v2-core

(b) Assessed Commit ID: faa5c6c

(c) Items in scope:

- lib/clones-with-immutable-args
- src/interface/IContractId.sol
- src/periphery/factory/PeripheryFactory.sol
- src/periphery/interface/IFlashModule.sol
- src/periphery/interface/ILenderModule.sol
- src/periphery/interface/IPositionHelperBase.sol
- src/periphery/interface/ISwapModule.sol
- src/periphery/lib/PeripheryFactoryStorageLib.sol
- src/periphery/lib/PeripheryRolesLib.sol
- src/periphery/lib/PositionManagementHelperLib.sol
- src/periphery/lib/PositionManagementHelperStorageLib.sol
- src/periphery/positionhelper/PositionHelperAaveV3.sol
- src/periphery/positionhelper/PositionHelperBase.sol

Out-of-Scope: Third party dependencies and economic attacks.

FILE

(a) Submitted File: earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3.zip

(b) Items in scope:

- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/doc/spec/PositionManagementHelperAuditScope.md
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/doc/spec/ThePierreLtvAssistent.md
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/lib/clones-with-immutable-args/Clone.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/lib/clones-with-immutable-args/ClonesWithImmutableArgs.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/interface/IContractId.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/interface/IPositionHelperBase.sol

- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/interface/ILenderModule.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/interface/IFlashModule.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/interface/ISwapModule.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/positionhelper/PositionHelperAaveV3.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/positionhelper/PositionHelperBase.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/factory/PeripheryFactory.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/lib/PositionManagementHelperLib.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/lib/PositionManagementHelperStorageLib.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/lib/PeripheryFactoryStorageLib.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/src/periphery/lib/PeripheryRolesLib.sol
- /earn-v2-core-faa5c6c0e69b2edeb8918a28d8b6e53a3ad02ad3/doc/spec/PeripheryFactory.md

REMEDIATION COMMIT ID:

- 7c7768b
- ad3d2ff
- 79bc19b
- e658a21
- 9a2a76a
- 4a97013
- 515d985
- d63e563
- 82a13a8

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	9

INFORMATIONAL
4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
EMODE NOT CONSIDERED IN LTV CHECKS	LOW	RISK ACCEPTED - 04/14/2026
AAVE POSITION ISOLATION IS NOT ENFORCED PER-HELPER	LOW	RISK ACCEPTED - 04/15/2026
INCONSISTENT ROUNDING IN WITHDRAW LOGIC	LOW	SOLVED - 04/14/2026
REPAY FLOWS CAN STRAND BORROW TOKENS IN THE HELPER	LOW	RISK ACCEPTED - 04/14/2026
UNINITIALIZED HELPERS CAN BE HIJACKED	LOW	SOLVED - 04/14/2026
ATOMIC FULL UNWIND TO ZERO LTV IS BLOCKED IN _RETRACT()	LOW	SOLVED - 04/14/2026
FACTORY BLOCKS REGISTRATION OF NON-UPGRADEABLE STRATEGIES	LOW	SOLVED - 04/15/2026

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SINGLE STEP OWNERSHIP TRANSFER PROCESS	LOW	RISK ACCEPTED - 04/14/2026
STRATEGY UPGRADE PERMISSIONS MISALIGNED	LOW	SOLVED - 04/15/2026
LACK OF ZERO-PRICE CHECK IN ORACLE	INFORMATIONAL	SOLVED - 04/14/2026
INCORRECT STORAGE SLOT CALCULATION	INFORMATIONAL	SOLVED - 04/14/2026
IFLASHMODULE.FLASH() RETURN VALUE IS IGNORED	INFORMATIONAL	SOLVED - 04/14/2026
REPAY EVENT INACCURACY	INFORMATIONAL	SOLVED - 04/14/2026

7. FINDINGS & TECH DETAILS

7.1 EMODE NOT CONSIDERED IN LTV CHECKS

// LOW

Description

The `getMaxLtvInWad()` and `getLiquidationThresholdInWad()` functions in `PositionHelperAaveV3` read the base reserve configuration from `DATA_PROVIDER.getReserveConfigurationData(COLLATERAL_TOKEN())`, ignoring the positionOwner's active eMode category.

However, when a `positionOwner` enables eMode and the helper's collateral and borrow assets are eligible under the active category, the effective max LTV and liquidation threshold are derived from that eMode category, which may be significantly higher than the base reserve values.

As a result, the helper can incorrectly reject valid leverage targets in `deploy()` and valid withdrawals in `_retract()` for users in eMode, even though Aave would allow these operations. This creates an inconsistency between the helper's target and withdrawal checks and the effective Aave risk parameters applied to the `positionOwner` under the active eMode category, and can break the helper for users whose active eMode category applies to the helper pair.

For example:

1. A user enables eMode for a supported asset category in Aave V3.
2. The operator calls `deploy()` or `_retract()` with a target LTV that is valid under eMode but above the base config.
3. The helper reverts, preventing the user from managing their position as Aave allows.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (3.4)

Recommendation

Update `getMaxLtvInWad()` and `getLiquidationThresholdInWad()` to check the `positionOwner`'s active eMode category and, when the helper's collateral and borrow assets are both eligible under that category, return the category-specific LTV and liquidation threshold; otherwise, fall back to the base reserve values.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> We do not utilize E-Mode in Aave V3, as it is not aligned with our strategy.
> Our use case involves supplying blue-chip volatile assets (e.g., ETH, BTC) as collateral and borrowing stablecoins (e.g., USDC) against them. E-Mode is designed to provide enhanced capital efficiency only when both collateral and borrowed assets belong to the same correlated category (e.g., stablecoin-to-stablecoin).

7.2 AAVE POSITION ISOLATION IS NOT ENFORCED PER-HELPER

// LOW

Description

The `PositionHelperAaveV3` contract models each clone as an isolated position for a specific `(collateralToken, borrowToken, positionOwner)` tuple. However, the helper's core accounting is keyed to the `positionOwner`'s Aave account rather than to the helper itself. In particular, `getCollateralAmount()` and `getDebtAmount()` read the `positionOwner`'s full balances for this helper's collateral and borrow reserves, while `getCurrentLtvInWad()` reads the `positionOwner`'s aggregate Aave account data.

This means that if a single `positionOwner` address is used for multiple helpers or has other Aave positions, the helper's accounting reads shared owner-level reserve balances, and its public LTV view reads aggregate owner-level Aave account data, rather than isolated per-helper state.

Any party able to initialize and operate another approved helper for the same `positionOwner`, or the `positionOwner` itself through unmanaged Aave activity, can unintentionally or maliciously cause one helper to repay debt or withdraw collateral associated with another helper managed position for that owner. Even when no cross helper action is taken, the helper's reserve balance views and public LTV view can reflect owner level Aave state rather than per-helper state when other Aave activity exists, which can mislead operators or automation. This breaks the intended isolation and can result in unintended position changes, unnecessary unwind or swap costs, or failed operations.

For example:

1. User deploys two helpers for the same `positionOwner` and overlapping Aave reserves (for example, the same collateral/borrow pair).
2. Helper A borrows and manages a position.
3. Helper B, seeing the aggregate debt, can call `selfLiquidate(type(uint256).max)` and repay debt belonging to Helper A.
4. This can cause the shared `positionOwner` account to lose collateral or incur unnecessary unwind / swap costs

Proof of Concept

In the following scenario, two different helpers read and manipulate the same state for the position owner:

Copy Code

```
function testPoC_crossHelperStateLeak_sameOwnerSamePair() public {
  _scenarioSetup();

  // helper2 is a fresh clone for the same owner + same Aave pair.
  bytes memory cloneArgs = abi.encodePacked(positionOwner, WETH, USDT, WETH_DECIMALS, USDT_DECIMALS);
  PositionHelperAaveV3 helper2 = PositionHelperAaveV3(ClonesWithImmutableArgs.clone(address(implement
```

```
    helper2.initialize(admin, operator, address(targetVault), flashModule, swapModule, HEALTHY_LTV, HEA

    // Approvals for helper2 so it can act too.
    vm.startPrank(positionOwner);
    IERC20(WETH).approve(address(helper2), type(uint256).max);
    IERC20(_aToken()).approve(address(helper2), type(uint256).max);
    IERC20(USDT).forceApprove(address(helper2), type(uint256).max);
    ICreditDelegationToken(_debtToken()).approveDelegation(address(helper2), type(uint256).max);
    vm.stopPrank();

    // Create the position only through helper1.
    deal(WETH, positionOwner, 10 ether);
    vm.prank(operator);
    helper.deploy(10 ether, 500_000_000_000_000_000); // 50%

    console.log("helper1 debt", helper.getDebtAmount());
    console.log("helper2 debt seen without using helper2", helper2.getDebtAmount());
    console.log("helper1 collateral", helper.getCollateralAmount());
    console.log("helper2 collateral seen without using helper2", helper2.getCollateralAmount());
    console.log("helper1 LTV", uint256(helper.getCurrentLtvInWad()));
    console.log("helper2 LTV seen without using helper2", uint256(helper2.getCurrentLtvInWad()));

    // Fresh helper2 sees the exact same Aave state.
    assertEq(helper2.getDebtAmount(), helper.getDebtAmount(), "helper2 should not see helper1 debt, but
    assertEq(helper2.getCollateralAmount(), helper.getCollateralAmount(), "helper2 should not see helpe
    assertEq(helper2.getCurrentLtvInWad(), helper.getCurrentLtvInWad(), "helper2 should not see helper1

    // And helper2 can actively manage the position created through helper1.
    uint256 ownerWethBefore = IERC20(WETH).balanceOf(positionOwner);

    vm.prank(operator);
    helper2.retract(0, 1 ether, 600_000_000_000_000_000); // 60%

    uint256 ownerWethReceived = IERC20(WETH).balanceOf(positionOwner) - ownerWethBefore;

    console.log("owner WETH received via helper2", ownerWethReceived);
    console.log("collateral after helper2.retract (as seen by helper1)", helper.getCollateralAmount());

    assertApproxEqAbs(ownerWethReceived, 1 ether, 1e9, "helper2 should be able to withdraw from helper1
    assertLt(helper.getCollateralAmount(), 10 ether, "helper2 changed the same underlying Aave position
}
```

```
Ran 1 test for test/periphery/unit-test/PositionHelperAaveV3Test.t.sol:PositionHelperAaveV3Test
[PASS] testPoC_crossHelperStateLeak_sameOwnerSamePair() (gas: 1913972)
Logs:
  helper1 debt 10000000004
  helper2 debt seen without using helper2 10000000004
  helper1 collateral 9999999999999999999
  helper2 collateral seen without using helper2 9999999999999999999
  helper1 LTV 500000000100250000
  helper2 LTV seen without using helper2 500000000100250000
  owner WETH received via helper2 1000000000000000000
  collateral after helper2.retract (as seen by helper1) 8999999999999999999

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 575.42ms (8.63ms CPU time)
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:L/Y:N (3.1)

Recommendation

Treat a dedicated `positionOwner` with no other helper-managed or unmanaged Aave activity as a hard operational invariant for each helper; if stronger guarantees are desired, add factory-level uniqueness checks for factory-deployed helpers for the same `(positionOwner, collateralToken, borrowToken)` tuple.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> There is no way to avoid that the position owner approves the position to anyone, whether it is yet another position management helper or another person. it is the responsibility of the owner to approve only those that it trusts. having said that, approving two PMH for the same tripple (owner, collateral, debt) does not achieve such a devastating effect as mentioned, since the helper contract never actually holds the position, it is just approved. so from the point of view of the PMH it doesnt really matter whether the position owner itself tampers with the position or another PMH. it is just an extended interface that allows automatization, but it is not stateful in that sense or a proxy. so the only state that is exists is at the lender who does not care whether a PMH helps with part or the whole of the position. If one wants to cover more complex scenarios with multiple colateral assets etc one would have to write a more complex helper. So having 2 or more PMH even for the same tuple is not really causing harm by itself but also is not really intended behaviour and cannot be avoided either through a factory cloning guard, since a non-factory PMH can also be approved by the position owner.

7.3 INCONSISTENT ROUNDING IN WITHDRAW LOGIC

// LOW

Description

The `flashWithdrawAmountFromTargetLtv()` and `flashWithdrawAmount()` functions in `PositionManagementHelperLib` are responsible for computing the minimum collateral to withdraw for flash self-liquidation, either to reach a target LTV or to repay a fixed debt amount. However, there are two related rounding issues:

1. In `flashWithdrawAmountFromTargetLtv()`, the cost factor `F` is computed using floor rounding, while `flashWithdrawAmount()` uses ceiling rounding. This inconsistency can cause the target-LTV self-liquidation path to underestimate the required collateral withdrawal in edge cases, leading to failed swaps and reverted transactions at the slippage boundary.
2. In `flashWithdrawAmount()`, ceiling rounding is applied only to the final division, not to the intermediate division of `y * F / WAD`. This can result in underestimating the required collateral when the intermediate division truncates a fractional result, because the later ceiling round does not recover the earlier truncation. As a consequence, the swap in the self-liquidation flow may revert due to insufficient collateral, breaking the intended atomic unwind behavior at the slippage boundary.

For example, if the intermediate calculation of `y * F / WAD` truncates a fractional value, the final collateral withdrawal can be understated, causing the swap to revert and the self-liquidation to fail.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:L/Y:N (3.1)

Recommendation

Apply conservative rounding consistently across both the library helpers and the target-LTV caller path. Specifically, use ceiling rounding for the cost factor `F` in `flashWithdrawAmountFromTargetLtv()`, avoid intermediate truncation in `flashWithdrawAmount()`, and keep the `debtToFlash` reconstruction in `PositionHelperBase._selfLiquidate()` aligned with the same rounding convention.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/7c7768bbb058650e433beb0173c9bc2730367d85

7.4 REPAY FLOWS CAN STRAND BORROW TOKENS IN THE HELPER

// LOW

Description

The `getDebtAmount()` function in `PositionHelperAaveV3` adds a small buffer (`REPAY_RAY_ROUNDING_BUFFER`) to the actual debt. When that buffered amount is used as a repayment target, most visibly in self-liquidation flows such as `selfLiquidate(type(uint256).max)`, the helper can transfer in or flash-borrow slightly more borrow tokens than Aave actually accepts in `POOL.repay()`. The excess borrow tokens are not swept back to the owner, breaking the zero-balance invariant and potentially trapping funds in the helper. This excess is not automatically returned to the owner and, unless later consumed by another helper operation, requires admin intervention via `rescueFunds` to be withdrawn.

This is not limited to self-liquidation: in the normal repay path, `_repay()` only sweeps residuals strictly greater than `REPAY_RAY_ROUNDING_BUFFER`, so a residual equal to the buffer can also remain in the helper.

For example:

1. Operator calls `selfLiquidate(type(uint256).max)` on a position with small debt.
2. The helper flash-borrows and attempts to repay slightly more than the actual debt.
3. The excess borrow tokens remain in the helper contract.
4. These tokens are not automatically returned to the position owner and, unless later consumed by another helper operation, may require admin recovery.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:L/Y:N (2.5)

Recommendation

Do not assume the requested repayment amount is fully consumed. Use the actual amount returned by `POOL.repay()` for accounting and events, and ensure any post-repay residual borrow tokens are returned to the `positionOwner` once they are no longer needed for the current flow, including after accounting for flash-loan repayment obligations in self-liquidations.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> The getDebtAmount() intentionally adds a 2-wei REPAY_RAY_ROUNDING_BUFFER to ensure full debt repayment. In the selfLiquidate(type(uint256).max) path, the same 2-wei residual remains in the

> helper after Aave accepts only the actual debt. This is by design — it is a fixed constant (not cumulative), worth fractions of a cent.

7.5 UNINITIALIZED HELPERS CAN BE HIJACKED

// LOW

Description

The `PeripheryFactory` allows deploying PMH clones, and strategy proxies whose implementations expose an externally callable initializer, without requiring atomic initialization (`initData` can be empty). If initialization is deferred, the first caller to invoke `initialize()` on the new contract sets the admin/operator and other critical configuration. Any account can win the race to initialize the contract, permanently hijacking it and taking control of the privileged configuration established during initialization.

For example:

1. User deploys a PMH clone, or a strategy proxy whose implementation exposes an externally callable initializer, without `initData`.
2. Attacker observes the deployment and calls `initialize()` first.
3. Attacker sets themselves as admin/operator.
4. The legitimate user cannot recover the contract or its privileges.

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:N/I:C/D:N/Y:N (2.5)

Recommendation

Require atomic initialization during deployment by enforcing non-empty `initData` in `deployStrategy()` and `clonePMH()`, or implement a mechanism that restricts `initialize()` to the intended deployer/owner.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in commits `ad3d2ff` and `1369464` by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/ad3d2ffc38900bb9f6178f504caaf6c5b52ed5b2

7.6 ATOMIC FULL UNWIND TO ZERO LTV IS BLOCKED IN _RETRACT()

// LOW

Description

The `PositionHelperBase._retract()` function is intended to allow an operator to fully unwind a leveraged position by repaying all debt and withdrawing all collateral in a single transaction, with `maxPermissibleLtv` set to zero. However, the internal math helper `PositionManagementHelperLib.withdrawable()` returns zero immediately when `ltv == 0`, even if the debt has already been fully repaid in the same transaction.

As a result, any attempt to fully repay and withdraw all collateral in one call with `maxPermissibleLtv = 0` will revert, forcing users to use multiple transactions or leave a nonzero LTV. This breaks the expected atomic unwind flow for position operators.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:L/I:N/D:N/Y:N (2.5)

Recommendation

Update the logic in `_retract()` and/or `withdrawable()` to allow full collateral withdrawal when the debt is fully repaid, even if `maxPermissibleLtv` is zero.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/79bc19bc15b78956daa1c4404766595279c29a64

7.7 FACTORY BLOCKS REGISTRATION OF NON-UPGRADEABLE STRATEGIES

// LOW

Description

The `PeripheryFactory.registerStrategy()` function reverts if `proxyAdmin` is set to `address(0)`, even though both the code comments and the factory specification explicitly state that `address(0)` should be allowed for non-upgradeable strategies or those not managed by the factory.

This inconsistency prevents users from registering externally deployed non-upgradeable strategies as intended, and leaves dead code paths in deregistration logic that expect `proxyAdmin` to be zero in such cases. The bug is not just a documentation mismatch: the storage struct and deregistration logic both anticipate `proxyAdmin = address(0)` as a valid scenario, but the function currently rejects it.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Allow `registerStrategy()` to accept `proxyAdmin = address(0)` for non-upgradeable strategies, as documented. Ensure all related logic (including deregistration) correctly handles the zero address case.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/e658a21772916f75bafe71544abe004e0292ebd7

7.8 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

The `PeripheryFactory` contract inherits OpenZeppelin's `OwnableUpgradeable`, which restricts certain functions to the contract owner. The `OwnableUpgradeable` pattern allows the contract owner to transfer ownership to another address using the `transferOwnership()` function. However, the `transferOwnership()` function does not include a two-step process to transfer ownership.

This creates operational risk if ownership is transferred to an incorrect or unresponsive address. In that case, the contract could be left in a state where administrative changes can no longer be made.

Additionally, the `renounceOwnership()` function allows the owner to renounce ownership permissions. Renouncing ownership before transferring it would result in the contract having no owner, eliminating the ability to call privileged functions.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (2.0)

Recommendation

Consider using OpenZeppelin's `Ownable2StepUpgradeable` contract over the `OwnableUpgradeable` implementation. `Ownable2StepUpgradeable` provides a two-step ownership transfer process, which adds an extra layer of security to prevent accidental ownership transfers.

Additionally, if permanent ownerless operation is not an intended end state, consider disabling or separately gating `renounceOwnership()`.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** made a business decision to accept the risk of this finding and not alter the contracts, stating:

> In our deployment, the factory owner is a multisig: any ownership change requires multiple independent signers and internal review before it can execute on-chain. That operational control gives us the same practical assurance as a two-step transfer pattern, so we do not plan to replace Ownable with a two-step variant for this contract.

7.9 STRATEGY UPGRADE PERMISSIONS MISALIGNED

// LOW

Description

For ID-compliant strategies, the `upgradeStrategy()` and `batchUpgradeStrategies()` functions in `PeripheryFactory` require the caller to hold the `STRATEGY_UPGRADER` role on the target strategy proxy; legacy strategies instead require `STRATEGY_ADMIN`. However, under the documented deployment flow, this role is expected to be granted to the factory contract itself (as `msg.sender` in the proxy's initializer), not to the factory owner EOA.

Since the factory does not internally call `upgradeStrategy()` on its own behalf, any factory owner EOA that is not separately granted the required strategy role will revert with `NotStrategyUpgrader` when attempting to upgrade via the factory interface.

This breaks the documented upgrade path for factory-managed strategies deployed under that role-grant pattern, preventing emergency or routine upgrades by the factory owner unless the owner EOA is separately granted the required role or the factory implements an owner bypass.

Additionally, the documentation on `PeripheryFactory.md` states that the required role is `ROLE_ADMIN`, while the code checks `STRATEGY_UPGRADER` for ID-compliant strategies and `STRATEGY_ADMIN` for legacy strategies, further increasing the risk of misconfiguration and missed upgrades due to this mismatch.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:C/D:N/Y:N (2.0)

Recommendation

Ensure that the expected external upgrade caller is granted the required role during strategy deployment and registration, or otherwise align the factory's role check with the actor that is intended to trigger upgrades. Additionally, update documentation to clarify the required roles and expected upgrade flow.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/9a2a76a45e3c3a7f2962b02b7fb2c68f62a6aaea

7.10 LACK OF ZERO-PRICE CHECK IN ORACLE

// INFORMATIONAL

Description

The `collateralPriceInBorrowTokensInRay()` function in `PositionHelperAaveV3` does not validate that the prices returned by the oracle are non-zero. If the oracle returns zero for the collateral token, the function returns zero; if it returns zero for the borrow token, the division reverts inside `mulDiv`. This can propagate through position math and cause critical operations such as `deploy()`, `_retract()`, or `selfLiquidate()` to produce partial no-op behavior or revert in non-obvious ways.

This can lead to confusing denial-of-service or partial no-op scenarios if the oracle is misconfigured or unavailable.

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:M/I:N/D:N/Y:N (1.3)

Recommendation

Add explicit checks for zero prices in `collateralPriceInBorrowTokensInRay()` and revert with a descriptive error if either price is zero.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/4a970132799b4bc82efef87d34361de900f632b0

7.11 INCORRECT STORAGE SLOT CALCULATION

// INFORMATIONAL

Description

The `PeripheryFactoryStorageLib` contract defines a private constant for its custom storage slot, intended to follow the ERC-7201 pattern: `keccak256(abi.encode(uint256(keccak256("concrete.storage.PeripheryFactoryStorage")) - 1)) & ~bytes32(uint256(0xff))`. However, the assigned slot value does not match the correct calculation.

Specifically, the slot is set to `0x938627f798558987333b9a8337806955a8a10d7794ab4306915141ada51d2c00` but should be `0xc57b9181734711449c6174643141c6b5854d4e37846989b9214d7aff93543d00`.

The main risk is upgrade and operational safety: future developers, auditors, or tooling may assume the documented namespace formula is authoritative and “correct” the slot constant in a later implementation. If that happens, the upgraded contract would read and write a different storage location than the one already in use, which could break configuration, access control, or other persistent state.

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.8)

Recommendation

Update the custom storage slot constant to match the correct formula for the intended namespace.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/515d985d3543b9f6b25d571d0a58270a44c5d3cb

7.12 IFLASHMODULE.FLASH() RETURN VALUE IS IGNORED

// INFORMATIONAL

Description

The `_selfLiquidate()` function in `PositionHelperBase` initiates a flash loan by calling `IFlashModule.flash(...)`, which returns a boolean indicating success. However, the return value is not checked, so if a flash module implementation signals failure by returning `false` instead of reverting, the self-liquidation call can still return successfully even though the flash request was reported as unsuccessful.

This can cause automation or monitoring systems to misinterpret a reported flash failure as a successful self-liquidation based on transaction status alone, potentially leaving positions at risk.

BVSS

AO:A/AC:L/AX:L/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.6)

Recommendation

Require that `IFlashModule.flash()` returns `true` and revert otherwise, or document and enforce that all flash module implementations must revert on failure rather than returning `false`.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/d63e5631aae86a4b8f6755d9d80d9fe65e9a43dc

7.13 REPAY EVENT INACCURACY

// INFORMATIONAL

Description

The `PositionHelperAaveV3._repayLender()` function emits `Repaid(borrowAmount)` using the requested repayment amount rather than the actual amount accepted by `POOL.repay()` function. If the requested amount exceeds the outstanding debt (e.g., due to rounding buffer), Aave will only accept the true debt amount, but the event will still report the higher requested value. This can cause off-chain systems relying on events to misreport the actual debt reduction.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Capture the return value of `POOL.repay()` and emit the `Repaid` event with the actual amount repaid, ensuring event accuracy for downstream consumers.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/82a13a8baa0109625bc74fab1271e94d59eccdc7
