// Security Assessment

09.23.2025 - 09.30.2025

Earn V2 Core - Async Implementation

Blueprint Finance

HALBORN

Earn V2 Core - Async Implementation - Blueprint Finance

Prepared by: HALBORN

Last Updated 10/10/2025

Date of Engagement: September 23rd, 2025 - September 30th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	0	0	1	10

TABLE OF CONTENTS

1. Introduction

`Blueprint Finance` engaged `Halborn` to perform a security assessment of their smart contracts from September 23rd, 2025 to September 30th, 2025. 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 implementing an upgradeable vault system with asynchronous withdrawal queuing, multi-strategy asset allocation, fee splitting, and privileged strategy management.

2. Assessment Summary

`Halborn` was allocated 6 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 primary recommendations were:

- `Reorder the logic in claimUsersBatch() to clear the user's claimable state before performing the external transfer.`
- `Enforce a minimum margin between accountingValidityPeriod and cooldownPeriod in both setter functions to ensure a robust and predictable update window.`
- `Restrict unpauseAndAdjustTotalAssets() to only allow adjustments that pass the same validation as adjustTotalAssets(), or remove the function if not strictly necessary.`

3. Test Approach And Methodology

`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: c02454d

(c) Items in scope:

- src/common/UpgradeableVault.sol
- src/implementation/ConcreteAsyncVaultImpl.sol
- src/implementation/ConcreteStandardVaultImpl.sol
- lib/AsyncVaultHelperLib.sol
- lib/ERC20Lib.sol
- lib/storage/ConcreteAsyncVaultImplStorageLib.sol
- periphery/auxiliary/TwoWayFeeSplitter.sol
- periphery/lib/BaseStrategyStorageLib.sol
- periphery/lib/MultisigStrategyStorageLib.sol
- periphery/lib/PeripheryRolesLib.sol
- periphery/lib/PositionAccountingLib.sol
- periphery/lib/PositionAccountingStorageLib.sol
- periphery/lib/SimpleStrategyStorageLib.sol
- periphery/strategies/BaseStrategy.sol
- periphery/strategies/MultisigStrategy.sol
- periphery/strategies/SimpleStrategy.sol

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

REMEDIATION COMMIT ID:

- 01a030b
- 92b192f

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

INFORMATIONAL
10

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNFOLLOWED CHECKS-EFFECTS-INTERACTIONS PATTERN	LOW	SOLVED - 10/03/2025
MISSING INPUT VALIDATION	INFORMATIONAL	PARTIALLY SOLVED - 10/03/2025
INSUFFICIENT MARGIN BETWEEN ACCOUNTING VALIDITY AND COOLDOWN PERIODS	INFORMATIONAL	SOLVED - 10/02/2025
ADMIN CAN BYPASS ACCOUNTING GUARD VIA UNPAUSEANDADJUSTTOTALASSETS	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
ROUNDING MODE MISMATCH IN EPOCH ASSET RESERVATION LOGIC	INFORMATIONAL	SOLVED - 10/03/2025
INCORRECT STORAGE SLOT CONSTANT DECLARATION	INFORMATIONAL	SOLVED - 10/03/2025
LACK OF RECIPIENT VALIDATION IN FEE SPLITTER ENABLES MISLEADING ACCOUNTING	INFORMATIONAL	SOLVED - 10/03/2025
STALE PER-EPOCH TOTALREQUESTEDSHARES VALUE AFTER PROCESSING	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
REENTRANCYGUARDUPGRADEABLE INITIALIZER NOT INVOKED	INFORMATIONAL	SOLVED - 10/03/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
TYPO IN THE CODE	INFORMATIONAL	SOLVED - 10/03/2025
REDUNDANT CODE	INFORMATIONAL	SOLVED - 10/03/2025

7. FINDINGS & TECH DETAILS

7.1 UNFOLLOWED CHECKS-EFFECTS-INTERACTIONS PATTERN

// LOW

Description

The `claimUsersBatch()` function in `AsyncVaultHelperLib` processes batch claims for users in a given epoch. However, it performs the external asset transfer (`IERC20(asset).safeTransfer(user, assets)`) before clearing the user's claimable state (`userEpochRequests[user][epochID] = 0`). This ordering does not follow the checks-effects-interactions pattern, which is a best practice to prevent reentrancy vulnerabilities.

While the current implementation is protected by role-based access and the underlying asset is assumed to be a standard ERC20, future upgrades or integration with tokens supporting hooks (e.g., ERC777) could expose the contract to reentrancy risks, potentially allowing a user to double-claim assets.

BVSS

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

Recommendation

Reorder the logic in `claimUsersBatch()` to clear the user's claimable state before performing the external transfer.

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/01a030bedbd4315141b44e4a6f523dd590a54403

7.2 MISSING INPUT VALIDATION

// INFORMATIONAL

Description

Throughout the codebase, there are several instances where input values are assigned without proper validation. Failing to validate inputs before assigning them to state variables or using them in protocol logic can lead to unexpected system behavior, weakened safety guarantees, or even complete failure.

Instances of this issue include:

- In `PositionAccountingLib.setMaxAccountingChangeThreshold()`, the `maxAccountingChangeThreshold_` parameter is assigned directly without checking that it is less than or equal to `BASIS_POINTS` (10,000).
- In `PositionAccountingLib.setCooldownPeriod()`, the `cooldownPeriod_` parameter is not validated for a sensible minimum value (e.g., greater than zero), which could allow disabling cooldown protection.
- In `ConcreteAsyncVaultImpl.toggleQueueActive()`, there is no validation to prevent disabling the queue while there are pending requests, which could lead to user confusion or inconsistent withdrawal behavior.
- In management and performance fee setters (`updateManagementFee()`, `updatePerformanceFee()`), there are no hard-coded upper bounds on fee rates, allowing privileged roles to set excessive fees and dilute user shares.
- In `PositionAccountingStorageLib.initialize()`, the `maxAccountingChangeThreshold_` parameter is assigned directly without checking that it is less than or equal to `BASIS_POINTS` (10,000).
- In `PositionAccountingStorageLib.initialize()`, the `accountingValidityPeriod_` parameter is not validated to ensure it is greater than `cooldownPeriod_`, which can lead to immediate expiry or liveness issues.
- In `TwoWayFeeSplitter.initialize()`, the `feeType` parameter is not validated or restricted to a known set of values, which can lead to inconsistent or meaningless fee type labeling across deployments.

BVSS

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

Recommendation

Add proper validation to ensure that input values are within expected ranges and that addresses are not the zero address.

Remediation Comment

PARTIALLY SOLVED: The **Blueprint Finance team** partially solved this finding in the specified commit by adding input validation to several of the aforementioned instances.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/01a030bedbd4315141b44e4a6f523dd590a54403

7.3 INSUFFICIENT MARGIN BETWEEN ACCOUNTING VALIDITY AND COOLDOWN PERIODS

// INFORMATIONAL

Description

The `setAccountingValidityPeriod()` and `setCooldownPeriod()` functions in `PositionAccountingLib` enforce that `accountingValidityPeriod` must be strictly greater than `cooldownPeriod`. However, the contract does not enforce a minimum margin between these two values.

If `accountingValidityPeriod` is set only slightly greater than `cooldownPeriod` (e.g., by 1 second), there will be an extremely narrow window to perform accounting updates after the cooldown expires. This fragility can lead to missed updates due to block time variance or transaction delays, potentially causing the protocol to revert with `AccountingValidityPeriodExpired()` even when the cooldown has passed.

For example, if `cooldownPeriod` is set to 100 seconds and `accountingValidityPeriod` to 101 seconds, there is only a 1-second window to perform the next update after cooldown, which is impractical and unreliable.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:M/I:N/D:N/Y:N (1.0)

Recommendation

Enforce a minimum margin between `accountingValidityPeriod` and `cooldownPeriod` in both setter functions to ensure a robust and predictable update window.

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/92b192fce8c39655eefbd38feff4b4aa7e15de63

7.4 ADMIN CAN BYPASS ACCOUNTING GUARD VIA UNPAUSEANDADJUSTTOTALASSETS

// INFORMATIONAL

Description

The `MultisigStrategy.unpauseAndAdjustTotalAssets()` function allows an address with the STRATEGY_ADMIN role to unpause the strategy and directly adjust the reported total assets by any amount, without passing through the accounting validation logic enforced by `PositionAccountingLib.isValidAccountingChange()`. This bypasses the intended controls on asset reporting and nonce/timestamp advancement.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:M/D:N/Y:N (1.0)

Recommendation

Restrict `unpauseAndAdjustTotalAssets()` to only allow adjustments that pass the same validation as `adjustTotalAssets()`, or remove the function if not strictly necessary.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** made a business decision to acknowledge this finding and not alter the contracts.

7.5 ROUNDING MODE MISMATCH IN EPOCH ASSET RESERVATION LOGIC

// INFORMATIONAL

Description

The `ConcreteAsyncVaultImpl` contract is designed to reserve assets during epoch processing to guarantee that all user withdrawal claims can be fulfilled. According to the technical documentation (`ConcreteAsyncVaultImpl-doc.md`, section 7.2), the calculation for reserving assets should use rounding up (`Math.Rounding.Ceil`) when converting shares to assets.

However, the actual implementation in the contract uses rounding down (`Math.Rounding.Floor`) when calculating the share price and, by extension, the reserved assets.

This discrepancy means that, for each epoch, a small amount of value may be left unreserved, causing users to receive slightly less than their fair share when claiming withdrawals. Over time, this dust can accumulate in the vault, diverging from the intended behavior described in the documentation.

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 implementation to use `Math.Rounding.Ceil` when reserving assets for processed epochs, aligning the code with the documented specification.

Alternatively, if the current behavior is preferred, update the documentation to reflect the use of rounding down.

Remediation Comment

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

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/92b192fce8c39655eefbd38feff4b4aa7e15de63

7.6 INCORRECT STORAGE SLOT CONSTANT DECLARATION

// INFORMATIONAL

Description

The `ConcreteAsyncVaultImplStorageLib` library defines the storage slot constant `ConcreteAsyncVaultImplStorageLocation` as:

```
/// @dev keccak256(abi.encode(uint256(keccak256("concrete.storage.ConcreteAsyncVaultImplStorage")) -
bytes32 private constant ConcreteAsyncVaultImplStorageLocation =
    0xd3b5f67b5a9bb5c5a5b5c5a5b5c5a5b5c5a5b5c5a5b5c5a5b5c5a5b5c5a5b500;
```

However, the correct value, as computed by the documented formula `keccak256(abi.encode(uint256(keccak256("concrete.storage.ConcreteAsyncVaultImplStorage")) - 1)) & ~bytes32(uint256(0xff))` using Foundry's Chisel tool is `0xada5b606f7944319310c49c0f9f30d6272793a991bd2b9c3db8049867746700`:

```
➜keccak256(abi.encode(uint256(keccak256("concrete.storage.ConcreteAsyncVaultImplStorage")) - 1)) & ~bytes32(uint256(0xff))
Type: uint256
├ Hex: 0xada5b606f7944319310c49c0f9f30d6272793a991bd2b9c3db8049867746700
├ Hex (full word): 0x0ada5b606f7944319310c49c0f9f30d6272793a991bd2b9c3db8049867746700
└ Decimal: 490893180476634012638832172745795792923919693863582992281704942483034946944O
```

While this does not currently break storage access if the incorrect slot is used consistently throughout the codebase, it may cause confusion for future maintainers or integrators who expect the slot to match the documented formula. This inconsistency could lead to integration issues or errors if other contracts or tools rely on the documented calculation.

BVSS

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

Recommendation

Update the `ConcreteAsyncVaultImplStorageLocation` constant to match the documented formula.

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/01a030bedbd4315141b44e4a6f523dd590a54403

7.7 LACK OF RECIPIENT VALIDATION IN FEE SPLITTER ENABLES MISLEADING ACCOUNTING

// INFORMATIONAL

Description

The `TwoWayFeeSplitter` contract allows either the `mainRecipient` or `secondaryRecipient` to be set to the splitter contract's own address (`address(this)`), or for both recipients to be set to the same address.

When this occurs, calling `distributeFees()` will transfer vault tokens to the splitter itself or to the same address twice, leaving the contract's balance unchanged and/or inflating the `feesDistributed` metric. This can mislead off-chain systems or dashboards that rely on these metrics.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.5)

Recommendation

Add checks to prevent either recipient from being set to `address(this)` and to ensure that `mainRecipient` and `secondaryRecipient` are not identical.

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/01a030bedbd4315141b44e4a6f523dd590a54403

7.8 STALE PER-EPOCH TOTALREQUESTEDSHARES VALUE AFTER PROCESSING

// INFORMATIONAL

Description

`AsyncVaultHelperLib.processEpoch()` burns `requestingShares` and sets `epochPricePerShare` but leaves `totalRequestedSharesPerEpoch[epochID]` untouched. Post-processing logic never uses it, yet off-chain indexers may misinterpret the non-zero value as still-queued requests.

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (0.5)

Recommendation

Set `totalRequestedSharesPerEpoch[epochID] = 0` .

Remediation Comment

ACKNOWLEDGED: The `Blueprint Finance` team made a business decision to acknowledge this finding and not alter the contracts.

7.9 REENTRANCYGUARDUPGRADEABLE INITIALIZER NOT INVOKED

// INFORMATIONAL

Description

The `TwoWayFeeSplitter` contract inherits from `ReentrancyGuardUpgradeable` but does not call `__ReentrancyGuard_init()` in its `initialize()` function.

While this omission does not currently impact the contract's security or functionality, it deviates from OpenZeppelin's recommended upgradeable contract initialization pattern.

BVSS

AO:A/AC:L/AX:H/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.2)

Recommendation

Add a call to `__ReentrancyGuard_init()` in the `initialize()` function.

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/01a030bedbd4315141b44e4a6f523dd590a54403

7.10 TYPO IN THE CODE

// INFORMATIONAL

Description

In the `ConcreteAsyncVaultImplStorage` struct of the `ConcreteAsyncVaultImplStorageLib` library, there is a typo, where the word *Unclaimed* is misspelled as *Unlcaimed.* The same case can be found in the `pastEpochsUnlcaimedAssets()` function of the `ConcreteAsyncVaultImpl` contract.

While this typo does not affect the functionality of the code, it can make the codebase harder to read and understand.

BVSS

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

Recommendation

It is recommended to fix all typos to improve the readability of the codebase.

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/01a030bedbd4315141b44e4a6f523dd590a54403

7.11 REDUNDANT CODE

// INFORMATIONAL

Description

The `ConcreteAsyncVaultImpl._executeWithdraw()` function contains two identical checks: `require(shares > 0, ZeroShares());`. One at the start and another inside the `if ($.isQueueActive)` block. Since the first check already reverts if `shares == 0`, the second is redundant and unreachable.

BVSS

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

Recommendation

Remove the second `require(shares > 0, ZeroShares());` inside the `if ($.isQueueActive)` block to simplify the code.

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/01a030bedbd4315141b44e4a6f523dd590a54403
