// Security Assessment

09.03.2025 - 09.16.2025

Earn V2 Core - Standard Implementation

Blueprint Finance

HALBORN

Earn V2 Core - Standard Implementation - Blueprint Finance

Prepared by: HALBORN

Last Updated 10/10/2025

Date of Engagement: September 3rd, 2025 - September 16th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
12	0	0	0	3	9

TABLE OF CONTENTS

8. Automated Testing

1. Introduction

`Blueprint Finance` engaged `Halborn` to perform a security assessment of their smart contracts from September 3rd, 2025 to September 16th, 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 a modular, upgradeable ERC4626 vault system with strategy allocation, hooks, role-based access control, and factory-managed proxy deployment.

2. Assessment Summary

`Halborn` was allocated 10 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. These were partially addressed by the `Blueprint Finance team`. The primary recommendations were:

- `Integrate the performanceFeeHighWaterMark variable into the performance fee logic.`
- `Require that allocated == 0 before allowing a strategy to be removed, regardless of its status. Remove the exception for Halted status.`
- `Update allocation logic to compare the vault's asset balance before and after the call, and use the actual delta for allocated updates.`
- `Require that a strategy is not present in deallocationOrder before allowing its removal, regardless of its status. Alternatively, automatically remove the strategy from deallocationOrder as part of the removal process to ensure consistency.`

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: b1b7cec

(c) Items in scope:

- src/common/UpgradeableVault.sol
- src/factory/ConcreteFactory.sol
- src/factory/VaultProxy.sol
- src/implementation/ConcreteStandardVaultImpl.sol
- src/lib/storage/ConcreteCachedVaultStateStorageLib.sol
- src/lib/storage/ConcreteFactoryBaseStorageLib.sol
- src/lib/storage/ConcreteStandardVaultImplStorageLib.sol
- src/lib/Constants.sol
- src/lib/Conversion.sol
- src/lib/Hooks.sol
- src/lib/Roles.sol
- src/lib/StateSetterLib.sol
- src/lib/Time.sol
- src/module/AllocateModule.sol

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

REMEDIATION COMMIT ID:

- 4f64163

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	3

INFORMATIONAL
9

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
STRATEGY CAN BE REMOVED WHILE STILL HOLDING ALLOCATED FUNDS	LOW	RISK ACCEPTED - 10/03/2025
LACK OF NON-ZERO OUTPUT CHECKS IN DEPOSIT AND REDEEM CAN RESULT IN USER ASSET LOSS	LOW	SOLVED - 10/03/2025
UNLIMITED APPROVAL RISKS IN ALLOCATEMODULE	LOW	RISK ACCEPTED - 10/03/2025
UNUSED HIGH-WATER MARK IN PERFORMANCE FEE CALCULATION	INFORMATIONAL	ACKNOWLEDGED - 09/26/2025
STRATEGY ALLOCATION ACCOUNTING CAN BE MANIPULATED BY STRATEGY CONTRACTS	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
MISMATCH IN PERFORMANCE FEE PREVIEW VS ACCRUAL AND LIQUIDITY PREVIEW VS EXECUTION	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
HOOKS CAN AFFECT SHARE/ASSET CONVERSION BY ALTERING VAULT BALANCE	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
DEALLOCATION ORDER CAN CONTAIN STALE, MISSING, OR DUPLICATE STRATEGIES	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
COMMENT/CODE MISMATCH	INFORMATIONAL	SOLVED - 10/03/2025
SETDEALLOCATIONORDER WILL REVERT IF MORE THAN 255 STRATEGIES ARE PASSED	INFORMATIONAL	SOLVED - 10/03/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 10/03/2025
UNUSED IMPORTS	INFORMATIONAL	SOLVED - 10/03/2025

7. FINDINGS & TECH DETAILS

7.1 STRATEGY CAN BE REMOVED WHILE STILL HOLDING ALLOCATED FUNDS

// LOW

Description

The `removeStrategy(address strategy)` function in `StateSetterLib` is intended to allow the removal of a strategy from the vault only when it is safe to do so. However, if a strategy's status is set to `Halted`, the function allows its removal even if the strategy still has non-zero allocated funds. This is due to the following logic:

```
93  function removeStrategy(address strategy) external {
94      SVLib.ConcreteStandardVaultImplStorage storage $ = SVLib.fetch();
95
96      IConcreteStandardVaultImpl.StrategyData memory strategyDataCached = $.strategyData[strategy]
97
98      require(
99          (strategyDataCached.allocated == 0 && _strategyNotInDeallocationOrder(strategy))
100             || strategyDataCached.status == IConcreteStandardVaultImpl.StrategyStatus.Halted,
101         IConcreteStandardVaultImpl.StrategyHasAllocation()
102     );
103     require($.strategies.remove(strategy), IConcreteStandardVaultImpl.StrategyDoesNotExist());
104
105     delete $.strategyData[strategy];
106
107     emit IConcreteStandardVaultImpl.StrategyRemoved(strategy);
108 }
```

If the strategy is `Halted`, the check passes regardless of the `allocated` value. As a result, the strategy can be removed and its data deleted while it still holds assets, breaking accounting and potentially resulting in loss of funds or inability to recover them.

BVSS

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

Recommendation

Require that `allocated == 0` before allowing a strategy to be removed, regardless of its status. Remove the exception for `Halted` status.

Remediation Comment

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

7.2 LACK OF NON-ZERO OUTPUT CHECKS IN DEPOSIT AND REDEEM CAN RESULT IN USER ASSET LOSS

// LOW

Description

The `deposit()` and `redeem()` functions in `ConcreteStandardVaultImpl` do not enforce that the calculated shares (for deposit) or assets (for redeem) are greater than zero before proceeding. This means that, under certain edge-case conditions, such as when `totalSupply` is very low and `cachedTotalAssets` is very high (which can be caused by a strategy over reporting yield), a user may deposit assets and receive zero shares, or redeem shares and receive zero assets.

In both cases, the user loses value with no compensation.

BVSS

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

Recommendation

Add validations to assert that the `shares` and `assets` are greater than `0` in `deposit()` and `redeem()` to prevent user loss in these scenarios.

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/4f64163e85945785dc6133e6cb0b172dd814f4d0

7.3 UNLIMITED APPROVAL RISKS IN ALLOCATEMODULE

// LOW

Description

The `AllocateModule` contract uses `forceApprove(strategy, type(uint256).max)` before each allocation. While this is reset to zero after the call, a malicious or compromised strategy could, in theory, exploit the approval window to transfer more assets than intended.

BVSS

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

Recommendation

Consider using minimal approvals or validating strategy contracts more strictly. Alternatively, document the trust assumptions for strategy contracts.

Remediation Comment

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

7.4 UNUSED HIGH-WATER MARK IN PERFORMANCE FEE CALCULATION

// INFORMATIONAL

Description

The `performanceFeeHighWaterMark` variable in `ConcreteStandardVaultImplStorageLib` is defined but never read from or updated in the `ConcreteStandardVaultImpl` implementation. As a result, performance fees are charged on any net positive yield within a single accrual period, including gains that merely recover previous losses.

This means the vault can charge performance fees multiple times on the same economic gain if the vault value fluctuates, rather than only on new profits above the previous high.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:N/Y:L (1.7)

Recommendation

Document the fee-on-recovery behavior clearly in user-facing documentation and disclosures, so users understand that performance fees may apply to recovered losses as well as new gains.

Alternatively, if the high-water mark will not be used, consider removing the unused variable to avoid confusion.

Remediation Comment

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

> We consciously opted for this design after carefully considering the options and the industry practices and do not consider it a threat but a design choice. Our design charges directly on the yield (also minted, but net effect is share value appreciation) instead of inflating the shares (net effect is share value depreciation). Thus charging the fees wont drop the share value. We are in line with major defi protocols, who handle the situation similarly.

7.5 STRATEGY ALLOCATION ACCOUNTING CAN BE MANIPULATED BY STRATEGY CONTRACTS

// INFORMATIONAL

Description

The `AllocateModule` contract, used via delegatecall in `ConcreteStandardVaultImpl.allocate`, updates the vault's internal `.allocated` value for each strategy based solely on the return value of `IStrategyTemplate(strategy).allocateFunds()` and `deallocateFunds()`. However, there is no check that the actual asset balance change matches the reported value. A malicious or buggy strategy could over report allocation or under report deallocation, leading to incorrect accounting, fee miscalculation, and potential user loss.

Additionally, when a strategy is toggled to `Halted`, the vault stops updating its `.allocated` value for that strategy. If the real value of the strategy decreases (e.g., due to a hack or loss) after being halted, the vault continues to use the old, higher value in its accounting. This allows users to withdraw or redeem at an inflated share price, extracting more than their fair share and pushing hidden losses onto remaining holders.

BVSS

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

Recommendation

Update allocation logic to compare the vault's asset balance before and after the call, and use the actual delta for allocated updates.

Consider excluding halted strategies from total assets until reconciled.

Remediation Comment

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

7.6 MISMATCH IN PERFORMANCE FEE PREVIEW VS ACCRUAL AND LIQUIDITY PREVIEW VS EXECUTION

// INFORMATIONAL

Description

The `ConcreteStandardVaultImpl` contract exhibits inconsistencies between preview and execution logic in two areas:

- **Performance fee preview:**
  The `_previewAccrueYieldAndFees()` function subtracts the management fee amount from total assets before calculating the performance fee, while the actual accrual path (`accruePerformanceFee()`) uses the full total assets value. This results in the previewed performance fee being slightly lower than the fee actually minted, potentially confusing users and integrators.

- **Liquidity preview:**
  The `maxWithdraw` and `maxRedeem` functions preview available liquidity by considering all active strategies, but actual withdrawals only use strategies listed in `deallocationOrder`. If `deallocationOrder` omits any active, liquid strategies, the previewed maximum withdrawable amount may be higher than what can actually be withdrawn, leading to failed transactions and user confusion.

BVSS

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

Recommendation

Standardize the asset base used for performance fee calculations in both preview and accrual paths, and align the liquidity preview logic with the actual withdrawal execution logic to ensure consistency.

Remediation Comment

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

7.7 HOOKS CAN AFFECT SHARE/ASSET CONVERSION BY ALTERING VAULT BALANCE

// INFORMATIONAL

Description

The `ConcreteStandardVaultImpl` contract supports pre-action hooks (such as `preDeposit`, `preMint`, `preWithdraw`, and `preRedeem`) via the `HooksLibV1` library. These hooks are invoked before the vault calculates the number of shares or assets for a user action, but after the vault's asset balance is cached.

If a hook implementation transfers assets into or out of the vault during its execution, the cached asset value used for conversion will be stale. This could result in value extraction or dilution, as the effective price at which shares are minted or redeemed will be manipulated.

BVSS

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

Recommendation

Recompute or recheck the vault's asset balance after executing pre-action hooks.

Remediation Comment

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

7.8 DEALLOCATION ORDER CAN CONTAIN STALE, MISSING, OR DUPLICATE STRATEGIES

// INFORMATIONAL

Description

The `deallocationOrder` array in `ConcreteStandardVaultImpl` determines the order in which strategies are used to fulfill withdrawals. However, the protocol does not enforce that this array contains all and only the active strategies, nor does it prevent duplicates. Notably, when a strategy is removed, it is not automatically purged from `deallocationOrder`, leaving stale entries.

As a result, `deallocationOrder` can contain stale (removed) strategies, omit active ones, or include duplicates. This can cause withdrawals to fail even when sufficient funds exist, or waste gas on unnecessary or invalid withdrawal attempts.

BVSS

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

Recommendation

Ensure that `deallocationOrder` always contains each active strategy exactly once, with no duplicates or stale entries.

Alternatively, automatically update `deallocationOrder` when strategies are added or removed, or validate its integrity before processing withdrawals.

Remediation Comment

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

7.9 COMMENT/CODE MISMATCH

// INFORMATIONAL

Description

The `_withdraw()` function in `ConcreteStandardVaultImpl.sol` currently transfers assets to the receiver before burning the user's shares, while the accompanying comment states that the transfer should occur after burning shares for safer ERC777 handling.

```
715 // Conclusion: we need to do the transfer after the burn so that any reentrancy would happen aft
716 // shares are burned and after the assets are transferred, which is a valid state.
717
718 SafeERC20.safeTransfer(IERC20(asset()), receiver, assets);
719 CachedVaultStateLib.fetch().cachedTotalAssets -= assets;
720 _burn(owner, shares);
```

Although the function is protected by the `nonReentrant` modifier, which currently prevents reentrancy attacks, this mismatch between the comment and implementation could lead to confusion or introduce vulnerabilities if the function is refactored or the guard is removed in the future.

Additionally, in `ConcreteFactory`, the NatSpec documentation for the `create` and `predictVaultAddress` functions claims that if `salt == 0`, a deterministic salt will be computed from the deployer, version, and owner. In reality, the implementation simply forwards zero as the salt, and no computation occurs. This discrepancy may confuse users expecting automatic salt derivation.

BVSS

AO:A/AC:L/AX:M/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.4)

Recommendation

Update comments and documentation to match the actual implementation, or update the code to match the documented behavior.

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/4f64163e85945785dc6133e6cb0b172dd814f4d0

7.10 SETDEALLOCATIONORDER WILL REVERT IF MORE THAN 255 STRATEGIES ARE PASSED

// INFORMATIONAL

Description

The `setDeallocationOrder(address[] calldata order)` function in `StateSetterLib.sol` uses a `uint8` loop index.

```
147 function setDeallocationOrder(address[] calldata order) external {
148     SVLib.ConcreteStandardVaultImplStorage storage $ = SVLib.fetch();
149
150     delete $.deallocationOrder;
151
152     uint256 orderLength = order.length;
153     for (uint8 i = 0; i < orderLength; i++) {
154         address strategy = order[i];
155         require($.strategies.contains(strategy), IConcreteStandardVaultImpl.StrategyDoesNotExist
156         require(
157             $.strategyData[strategy].status == IConcreteStandardVaultImpl.StrategyStatus.Active,
158             IConcreteStandardVaultImpl.StrategyIsHalted()
159         );
160
161         $.deallocationOrder.push(strategy);
162     }
163
164     emit IConcreteStandardVaultImpl.DeallocationOrderUpdated();
165 }
```

If more than 255 strategies are passed, the function will revert due to overflow.

BVSS

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

Recommendation

Change the loop index from `uint8` to `uint256` to match the array length type. Alternatively, limit the amount of strategies that can be added to be less than 256.

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/4f64163e85945785dc6133e6cb0b172dd814f4d0

7.11 FLOATING PRAGMA

// INFORMATIONAL

Description

The contracts in scope currently use floating pragma version `^0.8.0` which means that the code can be compiled by any compiler version that is greater than these version, and less than `0.9.0`.

However, it is recommended that contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively.

Additionally, from Solidity versions `0.8.20` through `0.8.24`, the default target EVM version is set to `Shanghai`, which results in the generation of bytecode that includes `PUSH0` opcodes. Starting with version `0.8.25`, the default EVM version shifts to `Cancun`, introducing new opcodes for transient storage, `TSTORE` and `TLOAD`.

In this aspect, it is crucial to select the appropriate EVM version when it's intended to deploy the contracts on networks other than the Ethereum mainnet, which may not support these opcodes. Failure to do so could lead to unsuccessful contract deployments or transaction execution issues.

BVSS

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

Recommendation

Lock the pragma version to the same version used during development and testing (for example: `pragma solidity 0.8.28;`), and make sure to specify the target EVM version when using Solidity versions from `0.8.20` and above if deploying to chains that may not support newly introduced opcodes.

Additionally, it is crucial to stay informed about the opcode support of different chains to ensure smooth deployment and compatibility.

Remediation Comment

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

7.12 UNUSED IMPORTS

// INFORMATIONAL

Description

Throughout the code, there are several instances of unused components that could be removed to improve code readability and maintainability.

Instances of this issue include:

- In `ConcreteStandardVaultImpl`:

```
import {IConcreteFactory} from "../interface/IConcreteFactory.sol";
```

- In `StateSetterLib`:

```
import {IStrategyTemplate} from "../interface/IStrategyTemplate.sol";
```

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 unused imports from the files.

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/4f64163e85945785dc6133e6cb0b172dd814f4d0

8. AUTOMATED TESTING

Description

Halborn used automated testing techniques to increase coverage of specific areas within the smart contracts under review. Among the tools used was Slither , a Solidity static analysis framework. After Halborn successfully verified the smart contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was executed against the contracts. This tool performs static verification of mathematical relationships between Solidity variables to identify invalid or inconsistent usage of the contracts' APIs throughout the entire codebase.

The security team reviewed all findings reported by the Slither software; however, findings related to external dependencies have been excluded from the results below to maintain report clarity.

Output

Most findings identified by Slither were proved to be false positives and therefore were not added to the issue list in this report.

```
INFO:Detectors:
ConcreteStandardVaultImpl._previewStrategyYield(address).loss (src/implementation/ConcreteStandardVaultImpl.sol#856) is a local variable never initialized
ConcreteStandardVaultImpl._previewStrategyYield(address).yield (src/implementation/ConcreteStandardVaultImpl.sol#855) is a local variable never initialized
ConcreteStandardVaultImpl._accrueYield().totalPositiveYield (src/implementation/ConcreteStandardVaultImpl.sol#596) is a local variable never initialized
ConcreteStandardVaultImpl._accrueYield().totalNegativeYield (src/implementation/ConcreteStandardVaultImpl.sol#597) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ConcreteStandardVaultImpl.allocate(bytes) (src/implementation/ConcreteStandardVaultImpl.sol#60-70) ignores return value by allocateModule().functionDelegateCall(delegateData) (src/implementation/ConcreteStandardVaultImpl.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
UpgradeableVault._initialize(uint64,address,bytes).owner (src/common/UpgradeableVault.sol#70) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl.constructor(address).factory (src/implementation/ConcreteStandardVaultImpl.sol#57) shadows:
        - UpgradeableVault.factory (src/common/UpgradeableVault.sol#15) (state variable)
        - IUpgradeableVault.factory() (src/interface/IUpgradeableVault.sol#14) (function)
ConcreteStandardVaultImpl.withdraw(uint256,address,address).owner (src/implementation/ConcreteStandardVaultImpl.sol#239) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl.redeem(uint256,address,address).owner (src/implementation/ConcreteStandardVaultImpl.sol#282) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl.previewDeposit(uint256).totalSupply (src/implementation/ConcreteStandardVaultImpl.sol#388) shadows:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-108) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
ConcreteStandardVaultImpl.previewMint(uint256).totalSupply (src/implementation/ConcreteStandardVaultImpl.sol#396) shadows:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-108) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
ConcreteStandardVaultImpl.previewWithdraw(uint256).totalSupply (src/implementation/ConcreteStandardVaultImpl.sol#411) shadows:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-108) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
ConcreteStandardVaultImpl.previewRedeem(uint256).totalSupply (src/implementation/ConcreteStandardVaultImpl.sol#426) shadows:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-108) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
ConcreteStandardVaultImpl.maxRedeem(address).owner (src/implementation/ConcreteStandardVaultImpl.sol#449) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl.maxWithdraw(address).owner (src/implementation/ConcreteStandardVaultImpl.sol#458) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl._initialize(uint64,address,bytes).asset (src/implementation/ConcreteStandardVaultImpl.sol#533) shadows:
        - ERC4626Upgradeable.asset() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC4626Upgradeable.sol#132-135) (function)
        - IERC4626.asset() (node_modules/@openzeppelin/contracts/interfaces/IERC4626.sol#30) (function)
ConcreteStandardVaultImpl._initialize(uint64,address,bytes).name (src/implementation/ConcreteStandardVaultImpl.sol#535) shadows:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#71-74) (function)
        - IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#15) (function)
ConcreteStandardVaultImpl._initialize(uint64,address,bytes).symbol (src/implementation/ConcreteStandardVaultImpl.sol#536) shadows:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#80-83) (function)
        - IERC20Metadata.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#20) (function)
ConcreteStandardVaultImpl._executeWithdraw(address,address,address,uint256,uint256).owner (src/implementation/ConcreteStandardVaultImpl.sol#656) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl._withdraw(address,address,address,uint256,uint256).owner (src/implementation/ConcreteStandardVaultImpl.sol#702) shadows:
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
ConcreteStandardVaultImpl.previewPerformanceFee(uint256,uint256,uint256,uint256).totalSupply (src/implementation/ConcreteStandardVaultImpl.sol#915) shadows:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-108) (function)
        - IERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#27) (function)
ConcreteStandardVaultImpl._maxWithdraw(address).owner (src/implementation/ConcreteStandardVaultImpl.sol#946) shadows:
```

```
        - OwnableUpgradeable.owner() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#73-76) (function)
IConcreteStandardVaultImpl.updateManagementFee(uint16).managementFee (src/interface/IConcreteStandardVaultImpl.sol#339) shadows:
        - IConcreteStandardVaultImpl.managementFee() (src/interface/IConcreteStandardVaultImpl.sol#449-452) (function)
IConcreteStandardVaultImpl.updatePerformanceFee(uint16).performanceFee (src/interface/IConcreteStandardVaultImpl.sol#356) shadows:
        - IConcreteStandardVaultImpl.performanceFee() (src/interface/IConcreteStandardVaultImpl.sol#489) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
ConcreteFactory._upgrade(address,uint64,bytes) (src/factory/ConcreteFactory.sol#241-256) has external calls inside a loop: require(bool,error)(msg.sender == OwnableUpgradeable(vault).owner(),revert NotOwner()()) (src/factory/ConcreteFactory.sol#243)
ConcreteFactory._upgrade(address,uint64,bytes) (src/factory/ConcreteFactory.sol#241-256) has external calls inside a loop: currentVaultVersion = IUpgradeableVault(vault).version() (src/factory/ConcreteFactory.sol#245)
ConcreteFactory._upgrade(address,uint64,bytes) (src/factory/ConcreteFactory.sol#241-256) has external calls inside a loop: IVaultProxy(vault).upgradeToAndCall(getImplementationByVersion(newVersion),abi.encodeCall(IUpgradeableVault.upgrade,(newVersion,d
ata))) (src/factory/ConcreteFactory.sol#251-253)
AllocateModule.allocateFunds(bytes) (src/module/AllocateModule.sol#17-45) has external calls inside a loop: IERC20(IERC4626(address(this)).asset()).forceApprove(params[i].strategy,type()(uint256).max) (src/module/AllocateModule.sol#31)
AllocateModule.allocateFunds(bytes) (src/module/AllocateModule.sol#17-45) has external calls inside a loop: amount = IStrategyTemplate(params[i].strategy).allocateFunds(params[i].extraData) (src/module/AllocateModule.sol#33)
AllocateModule.allocateFunds(bytes) (src/module/AllocateModule.sol#17-45) has external calls inside a loop: IERC20(IERC4626(address(this)).asset()).forceApprove(params[i].strategy,0) (src/module/AllocateModule.sol#35)
AllocateModule.allocateFunds(bytes) (src/module/AllocateModule.sol#17-45) has external calls inside a loop: amount = IStrategyTemplate(params[i].strategy).deallocateFunds(params[i].extraData) (src/module/AllocateModule.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in ConcreteFactory._upgrade(address,uint64,bytes) (src/factory/ConcreteFactory.sol#241-256):
        External calls:
        - IVaultProxy(vault).upgradeToAndCall(getImplementationByVersion(newVersion),abi.encodeCall(IUpgradeableVault.upgrade,(newVersion,data))) (src/factory/ConcreteFactory.sol#251-253)
        Event emitted after the call(s):
        - Migrated(vault,newVersion) (src/factory/ConcreteFactory.sol#255)
Reentrancy in AllocateModule.allocateFunds(bytes) (src/module/AllocateModule.sol#17-45):
        External calls:
        - amount = IStrategyTemplate(params[i].strategy).allocateFunds(params[i].extraData) (src/module/AllocateModule.sol#33)
        - amount = IStrategyTemplate(params[i].strategy).deallocateFunds(params[i].extraData) (src/module/AllocateModule.sol#39)
        Event emitted after the call(s):
        - AllocatedFunds(params[i].strategy,params[i].isDeposit,amount,params[i].extraData) (src/module/AllocateModule.sol#43)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
UpgradeableVault._initialize(uint64,address,bytes) (src/common/UpgradeableVault.sol#70) is never used and should be removed
UpgradeableVault._upgrade(uint64,uint64,bytes) (src/common/UpgradeableVault.sol#78) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
ConcreteFactory._computeBytecode(uint64,address,bytes) (src/factory/ConcreteFactory.sol#126-141) uses literals with too many digits:
        - abi.encodePacked(type()(VaultProxy).creationCode,abi.encode(implementation,constructorData)) (src/factory/ConcreteFactory.sol#140)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:. analyzed (59 contracts with 100 detectors), 36 result(s) found
```

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.