// Security Assessment

02.24.2026 - 03.02.2026

Earn V2 Improvements

Blueprint Finance

HALBORN

Earn V2 Improvements - Blueprint Finance

Prepared by: HALBORN

Last Updated 03/19/2026

Date of Engagement: February 24th, 2026 - March 2nd, 2026

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	0	3	1

TABLE OF CONTENTS

1. INTRODUCTION

`Blueprint Finance` engaged `Halborn` to perform a security assessment of their smart contracts from February 24th, 2026 to March 2nd, 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 updates to the Earn V2 core smart contracts, including ID based contract identification, role system simplification, a comprehensive pause system, priority withdrawal queues, external library extraction, and enhancements to deposit/withdraw limits, hooks, and factory implementation registration.

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 fully addressed by the `Blueprint Finance` team. The main recommendations were:

- `Disallow approval of implementations with ID() == 0 by adding an explicit check in approveImplementation() and related flows.`
- `Update WithdrawLib.executeWithdrawFromStrategies() to check that each strategy is present in the strategies set before processing withdrawals. Additionally, enforce that deallocationOrder is updated whenever a strategy is removed.`
- `Add a check in claimPriorityWithdrawal() to ensure that the provided strategy address is a registered and active strategy for the vault, and that it matches the strategy from which the unwind was performed.`

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. CAVEATS

The current security assessment was focused solely on evaluating the changes introduced in the following commit: https://github.com/Blueprint-Finance/earn-v2-core/compare/62430dd..f1e16ee.

It's important to note that despite these caveats, the security assessment aimed to provide a thorough evaluation of the protocol's security posture. However, the limitations mentioned above should be considered when interpreting the findings and recommendations.

5. 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.

5.1 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:

$$E = \prod m_e$$

5.2 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:

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

5.3 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

6. SCOPE

REPOSITORY

(a) Repository: earn-v2-core

(b) Assessed Commit ID: f1e16ee

(c) Items in scope:

- src/common/UpgradeableVault.sol
- src/factory/ConcreteFactory.sol
- src/implementation/ConcreteAsyncVaultImpl.sol
- src/implementation/ConcreteStandardVaultImpl.sol
- src/interface/IAsyncAccounting.sol
- src/interface/IContractId.sol
- src/interface/IDefaultEnforcedVaultConfiguration.sol
- src/interface/IUpgradeableVault.sol
- src/lib/AccessControlDefaultAdminRulesEnumerable.sol
- src/lib/AccessControlDefaultAdminRulesEnumerableUpgradeable.sol
- src/lib/AccessControlLib.sol
- src/lib/AsyncVaultHelperLib.sol
- src/lib/ContractIdEncoding.sol
- src/lib/ERC20Lib.sol
- src/lib/Hooks.sol
- src/lib/LegacyAdminRolesLib.sol
- src/lib/RoleMigrationLib.sol
- src/lib/Roles.sol
- src/lib/StandardVaultHelperLib.sol
- src/lib/StateInitLib.sol
- src/lib/StateSetterLib.sol
- src/lib/WithdrawLib.sol
- src/lib/YieldAccrualLib.sol
- src/lib/storage/ConcreteFactoryBaseStorageLib.sol
- src/lib/storage/ConcreteStandardVaultImplStorageLib.sol
- src/lib/storage/DefaultEnforcedVaultConfigStorageLib.sol
- src/periphery/hooks/HookContainer.sol
- src/periphery/lib/HookContainerStorageLib.sol
- src/periphery/lib/Looping/BaseLoopingHelperLib.sol
- src/periphery/lib/PeripheryRolesLib.sol
- src/periphery/strategies/BaseLoopingStrategy.sol
- src/periphery/strategies/MultisigStrategy.sol
- src/strategy/BaseStrategy.sol

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

REMEDIATION COMMIT ID:

- 23bbbe1

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

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	3

INFORMATIONAL
1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FACTORY ALLOWS APPROVAL AND USE OF IMPLEMENTATION WITH ID ZERO	LOW	SOLVED - 03/13/2026
STALE DEALLOCATION ORDER CAN BREAK WITHDRAWALS AFTER STRATEGY REMOVAL	LOW	SOLVED - 03/13/2026
PRIORITY WITHDRAWAL DOES NOT VALIDATE STRATEGY MEMBERSHIP OR IDENTITY	LOW	SOLVED - 03/13/2026
HOOKCONTAINER CAN REGISTER ITSELF AS A HOOK	INFORMATIONAL	SOLVED - 03/13/2026

8. FINDINGS & TECH DETAILS

8.1 FACTORY ALLOWS APPROVAL AND USE OF IMPLEMENTATION WITH ID ZERO

// LOW

Description

The `approveImplementation()` function in `ConcreteFactory` allows the factory owner to approve an implementation whose `ID()` returns zero. The mapping `vaultImplToImplementation` does not restrict the use of zero as a key, and the `_isValidVaultImpl()` function treats any nonzero address in this mapping as valid, regardless of the ID value.

As a result, an implementation with `ID() == 0` can be registered, blocked, migrated, and used for vault creation, which may break assumptions or integrations.

BVSS

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

Recommendation

Disallow approval of implementations with `ID() == 0` by adding an explicit check in `approveImplementation()` and related flows.

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/23bbbe181234363d17a7771080c99c8a93bf2bf9

8.2 STALE DEALLOCATION ORDER CAN BREAK WITHDRAWALS AFTER STRATEGY REMOVAL

// LOW

Description

The `WithdrawLib.executeWithdrawFromStrategies()` function iterates the `deallocationOrder` array to process withdrawals from strategies. However, it only checks if a strategy's status is `Active` and does not verify that the strategy is still present in the vault's `strategies` set.

If a strategy is removed from the vault (via `StateSetterLib.removeStrategy()`), but remains in `deallocationOrder`, its storage entry is deleted and its status defaults to `Active`. This allows the withdrawal logic to attempt withdrawals from a stale strategy address, which can cause underflow, revert, or unexpected behavior.

An account with the `STRATEGY_MANAGER` role can remove a strategy without updating the deallocation order, leaving stale entries. When a user attempts to withdraw, the vault will process these stale entries, potentially causing withdrawals to fail and locking up user funds.

BVSS

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

Recommendation

Update `WithdrawLib.executeWithdrawFromStrategies()` to check that each strategy is present in the `strategies` set before processing withdrawals. Additionally, enforce that `deallocationOrder` is updated whenever a strategy is removed.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in commits `23bbbe1` and `23bbbe1` by following the mentioned recommendation.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/23bbbe181234363d17a7771080c99c8a93bf2bf9

8.3 PRIORITY WITHDRAWAL DOES NOT VALIDATE STRATEGY MEMBERSHIP OR IDENTITY

// LOW

Description

The `claimPriorityWithdrawal()` function in `ConcreteAsyncVaultImpl` allows the `PRIORITY_WITHDRAWAL_EXECUTOR` role to fulfill a user's withdrawal request by specifying a `strategy` address and an `unwindCost`. However, the function does not verify that the provided `strategy` address is actually a registered or active strategy for the vault, nor that it is the strategy from which the unwind occurred.

This could allow the executor to call `IAsyncAccounting(strategy).adjustTotalAssets(-int256(unwindCost))` on any contract implementing the interface, or to skip proper accounting if an incorrect or dummy address is provided.

BVSS

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

Recommendation

Add a check in `claimPriorityWithdrawal()` to ensure that the provided `strategy` address is a registered and active strategy for the vault, and that it matches the strategy from which the unwind was performed.

Remediation Comment

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

> We cannot do more than verify that the strategy belongs to the vault (registered and active). The unwind occurs in another operational flow that is also executed by the operator, so there is no on-chain easy way to link the unwind to its real origin. Correct attribution is the operational responsibility of the executor.

Remediation Hash

https://github.com/Blueprint-Finance/earn-v2-core/commit/23bbbe181234363d17a7771080c99c8a93bf2bf9

8.4 HOOKCONTAINER CAN REGISTER ITSELF AS A HOOK

// INFORMATIONAL

Description

The `addHook()` function in the `HookContainer` contract allows an account with the `HOOK_MANAGER_ROLE` to register any address as a hook, including the container contract itself.

However, if `HookContainer` is added as its own sub hook, any call to a hook entrypoint (such as `preDeposit`, `preWithdraw`, etc.) will result in recursive self-calls until the call stack limit or gas is exhausted, causing the transaction to revert. This configuration would brick all hook dispatched vault operations until the self-registration is removed.

BVSS

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

Recommendation

Consider adding a check in `addHook()` to prevent the contract from registering itself as a hook.

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/23bbbe181234363d17a7771080c99c8a93bf2bf9

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.