// Security Assessment

11.24.2025 - 12.04.2025

AssetCT

Blueprint Finance

HALBORN

AssetCT - Blueprint Finance

Prepared by: HALBORN

Last Updated 01/09/2026

Date of Engagement: November 24th, 2025 - December 4th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	0	0	6	5

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
    - 7.1 Non-atomic yield finalization creates temporary accounting desync
    - 7.2 Missing _disableinitializers() call
    - 7.3 Single step ownership transfer process
    - 7.4 Centralization risks
    - 7.5 Redundant role check in strategy yield finalization
    - 7.6 Vault config update can break yield flow
    - 7.7 Missing input validation in vault yield configuration
    - 7.8 Global qctxtoamount mapping enables cross flow dos
    - 7.9 Missing events
    - 7.10 Floating pragma
    - 7.11 Unused imports

1. INTRODUCTION

Blueprint Finance engaged Halborn to perform a security assessment of their smart contracts from November 24th, 2025 to December 4th, 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 ERC20 token with role-based minting and burning, merchant managed vault integrations, multisig yield strategies, and whitelisting hooks, designed for controlled asset issuance, redemption, and yield accrual.

2. ASSESSMENT SUMMARY

Halborn was allocated 9 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 Low and Informational security risks, which were partially addressed by the Blueprint Finance team. The main recommendations were:

- Consider combining the minting, transfer, and accounting update into a single atomic transaction.
- Consider a domain separation of the transaction hash keys by flow type using a namespace prefix.
- Add the _disableinitializers() function call.

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: ASSETct

(b) Assessed Commit ID: 48a32cb

(c) Items in scope:

- src/hooks/WhitelistHook.sol
- src/implementation/ConcreteAssetCtVault.sol
- src/lib/AssetCtMultisigStrategyStorageLib.sol
- src/lib/AssetCtRolesLib.sol
- src/lib/AssetCtStateInitLib.sol
- src/lib/AssetCtVaultHelperLib.sol
- src/lib/ConcreteMerchantStorageLib.sol
- src/merchant/ConcreteMerchant.sol
- src/strategies/AssetCtMultisigStrategy.sol
- src/AssetCT.sol
- Blueprint-Finance/earn-v2-core/pull/100/files

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

REMEDIATION COMMIT ID:

- e20bfbc
- 66376f9
- 871595c
- e257b13
- a16ef47
- 07d825c
- 9547dc0

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	6

INFORMATIONAL

5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NON-ATOMIC YIELD FINALIZATION CREATES TEMPORARY ACCOUNTING DESYNC	LOW	RISK ACCEPTED - 12/18/2025
MISSING _DISABLEINITIALIZERS() CALL	LOW	SOLVED - 12/18/2025
SINGLE STEP OWNERSHIP TRANSFER PROCESS	LOW	SOLVED - 12/18/2025
CENTRALIZATION RISKS	LOW	RISK ACCEPTED - 12/18/2025
REDUNDANT ROLE CHECK IN STRATEGY YIELD FINALIZATION	LOW	SOLVED - 11/25/2025
VAULT CONFIG UPDATE CAN BREAK YIELD FLOW	LOW	SOLVED - 12/17/2025
MISSING INPUT VALIDATION IN VAULT YIELD CONFIGURATION	INFORMATIONAL	SOLVED - 12/16/2025
GLOBAL QCTXTOAMOUNT MAPPING ENABLES CROSS FLOW DOS	INFORMATIONAL	ACKNOWLEDGED - 12/18/2025
MISSING EVENTS	INFORMATIONAL	SOLVED - 12/16/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	INFORMATIONAL	ACKNOWLEDGED - 12/17/2025
UNUSED IMPORTS	INFORMATIONAL	SOLVED - 12/16/2025

7. FINDINGS & TECH DETAILS

7.1 NON-ATOMIC YIELD FINALIZATION CREATES TEMPORARY ACCOUNTING DESYNC

// LOW

Description

The `ConcreteMerchant.finalizeYieldAccrual()` function mints AssetCT tokens and transfers them to the strategy's multisig. However, the function does not update the vault's internal accounting (`totalAssets`) in the same transaction. Instead, the accounting reconciliation is expected to occur in a separate, subsequent transaction via `AssetCtMultisigStrategy.adjustTotalAssets()` , which must be called by an admin.

This two-step process creates a window where the circulating AssetCT supply increases (held at the strategy multisig) while the vault's reported `totalAssets` remains unchanged or becomes stale due to the strategy's accounting validity period expiring. During this window:

1. The vault's share price (`convertToAssets` / `convertToShares`) does not reflect the newly minted yield.
2. Off-chain systems, UIs, or oracles querying `totalAssets` observe outdated values.
3. Users may experience mispriced share conversions.

BVSS

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

Recommendation

Consider combining the minting, transfer, and accounting update into a single atomic transaction.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** accepted the risk of this finding, stating that:

> The two-step yield accounting process is intentional by design.
>
> MultisigStrategy Safeguards: The MultisigStrategy linked to vaults is responsible for yield reporting and includes extensive validation mechanisms:
>
> - Maximum change bounds limiting yield update magnitude.
> - Transaction cooldown periods between updates.
> - Maximum validity periods preventing stale accounting.

> The QC yield accrual process handles swapping and depositing yield assets into the QC account. This operation produces a deterministic amount of slippage that must be reconciled in the vault's accounting. An independent, distributed backend is responsible for updating the vault's totalAssets via adjustTotalAssets().
>
> This separation is purposeful to:
>
> - Enhance system decentralization by avoiding tight coupling between yield finalization and accounting updates.
> - Prevent race conditions when asset updates originate from multiple sources.
> - Allow the backend to batch and validate accounting changes independently.
>
> The brief window between yield backed assetCT minting and vault accounting reconciliation (slippage) is expected and mitigated by the strategy's bounded update parameters, ensuring any temporary discrepancy remains within acceptable limits.

7.2 MISSING _DISABLEINITIALIZERS() CALL

// LOW

Description

The `ConcreteMerchant` and `AssetCtMultisigStrategy` contracts follow the upgradeable proxy pattern and inherit from OpenZeppelin's `Initializable` contract. According to OpenZeppelin's best practices for upgradeable contracts, implementation contracts should include a constructor that calls `_disableInitializers()` to prevent the implementation contract itself from being initialized.

While the proxy's storage isolation prevents direct exploitation of an initialized implementation, leaving the implementation unprotected deviates from security best practices and increases the attack surface. An attacker could initialize the implementation contract, which may cause confusion or enable social engineering attacks.

The `AssetCT` token contract correctly includes `_disableInitializers()` in its constructor, demonstrating the intended pattern.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:L/I:L/D:N/Y:N (2.1)

Recommendation

Add the `_disableinitializers()` function call in the aforementioned 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/ASSETct/commit/e20bfbc66e16a924a34fda12b96fe7144377c784

7.3 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

The `WhitelistHook` contract inherits the `Ownable` contract implementation from OpenZeppelin's library and is used to restrict access to certain functions to the contract owner. The `Ownable` 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.

Regarding this, it is crucial that the address to which ownership is transferred is verified to be active and willing to assume ownership responsibilities. Otherwise, the contract could be locked in a situation where it is no longer possible to make administrative changes to it.

Additionally, the `renounceOwnership()` function allows renouncing to the owner permission. 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 `Ownable2Step` contract over the `Ownable` implementation. `Ownable2Step` provides a two-step ownership transfer process, which adds an extra layer of security to prevent accidental ownership transfers.

Additionally, it is recommended that the owner cannot call the `renounceOwnership()` function to avoid losing ownership of the contract.

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/ASSETct/commit/66376f9c2d0b3acce13b6cfafc3acabf92e05f7a

7.4 CENTRALIZATION RISKS

// LOW

Description

The code in scope concentrates significant control over core economic functions in privileged roles without on-chain validation mechanisms. Specifically:

1. **Arbitrary AssetCT minting by OPERATOR_ROLE**: The `finalizeYieldAccrual()` function allows operators to mint any amount of AssetCT tokens independent of the recorded pending yield in `pendingYieldByNonce[nonce]`. There is no on-chain enforcement ensuring the minted `amountInAsset` corresponds to actual yield accrued. A compromised or malicious operator could inflate token supply without corresponding backing, breaking the economic peg between yield and AssetCT.

2. **Unconstrained configuration changes by MERCHANT_MANAGER_ROLE**: Admins can modify vault yield configurations (`setVaultYieldConfig()`), user vault mappings (`setUserVault()`), and core vault references (`setAssetCTVault()`) without timelock or multi-signature requirements. Changes to vault configurations while nonces are pending can brick yield finalization and mismatched multisig or asset configurations can redirect funds to unintended addresses.

3. **Arbitrary vault accounting adjustments**: The `AssetCtMultisigStrategy.adjustTotalAssets()` function allows admins to modify vault accounting (`totalAssets`) by any amount without validation against minted tokens or actual yield. Combined with arbitrary minting (point 1), this enables independent manipulation of both token supply and accounting, making share prices entirely dependent on operator correctness.

BVSS

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

Recommendation

Consider implementing some of the following safeguards to reduce centralization risks:

- Add reasonable bounds for operator minting (e.g., require minted amounts to be within a reasonable range of the pending yield).
- Use timelocks for critical admin operations (configuration changes, contract upgrades) to allow users time to react to malicious changes.
- Transition control of privileged roles to multi-signature wallets or DAO governance to eliminate single points of failure.

Remediation Comment

RISK ACCEPTED: The **Blueprint Finance team** accepted the risk of this finding, stating that:

> The concerns raised are inherent to the design of AssetCT, which operates within a permissioned Qualified Custodian (QC) ecosystem.
>
> Existing Mitigations:
>
> - All critical roles (Admin, Merchant Manager) utilize multi-signature wallets with multi-layer signature processes involving MPC wallets and fine-grained transaction policies.
> - AssetCT minting is economically inconsequential in isolation—tokens cannot be redeemed outside our closed QC ecosystem, limiting the impact of unauthorized minting.
> - Yield updates at the strategy level are protected by maximum change bounds, cooldown periods, and staleness checks, preventing abrupt or stale yield manipulation.
>
> Future Considerations: We will evaluate adding timelocks for sensitive configuration changes to further enhance risk mitigation.

7.5 REDUNDANT ROLE CHECK IN STRATEGY YIELD FINALIZATION

// LOW

Description

The `finalizeMerchantYieldAccrual()` function in the `AssetCtMultisigStrategy` contract is protected by both the `onlyAdminOrOperator` modifier and an explicit check that `msg.sender == merchant` . However, this redundancy means that the merchant contract must be explicitly granted the operator or admin role on the strategy in order to call this function, even though only the merchant should ever be allowed to call it.

This complicates deployment and increases the risk of misconfiguration, as the merchant contract must be assigned roles that are otherwise unnecessary for its operation.

BVSS

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

Recommendation

Remove the `onlyAdminOrOperator` modifier from `finalizeMerchantYieldAccrual()` and rely solely on the explicit merchant address check to simplify access control and reduce the risk of accidental privilege escalation or misconfiguration.

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/ASSETct/commit/871595c22f39e5a44dc7634ea1db5d9e5137ed28

7.6 VAULT CONFIG UPDATE CAN BREAK YIELD FLOW // LOW

Description

The `ConcreteMerchant.finalizeYieldAccrual()` function finalizes yield for a given nonce using the current vault yield configuration. However, if `setVaultYieldConfig()` is called after yield initiation but before finalization, the strategy contract may change.

This causes finalization to revert with `UnknownYieldNonce`, as the new strategy does not recognize the pending nonce. Yield for that nonce becomes permanently stuck until the configuration is manually rolled back.

BVSS

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

Recommendation

Store the strategy and multisig addresses for each yield nonce at initiation. Use the stored configuration during finalization to ensure consistency. Alternatively, prevent configuration changes while any nonces are pending for a vault.

Remediation Comment

SOLVED: The **Blueprint Finance** team solved this finding in the specified commit by removing the `setVault()` function, restricting `setUserAccount()` and `setUserVault()` to only allow changes when the user has no active deposits, and adding an emergency migration function for user accounts with strict checks. This prevents vault configuration changes while nonces are pending and eliminates the risk of breaking the yield flow.

Remediation Hash

https://github.com/Blueprint-Finance/ASSETct/commit/e257b137696a432b66cf1d723d58de75cf203973

7.7 MISSING INPUT VALIDATION IN VAULT YIELD CONFIGURATION

// INFORMATIONAL

Description

Throughout the `ConcreteMerchant` contract, several administrative functions lack proper input validation when configuring vaults and user mappings. Failing to validate inputs can lead to silent misconfiguration, fund loss, or denial of service.

Instances of this issue include:

- In `ConcreteMerchant.setVaultYieldConfig()` , the `yieldMultisig_` parameter is not validated against `strategy.getMultiSig()` . If mismatched, yield tokens will be sent to an unintended address during `initiateYieldAccrual()` , potentially causing fund loss or requiring manual recovery.
- In `ConcreteMerchant.setVaultYieldConfig()` , the function does not verify that `strategy.getYieldAsset()` matches the merchant's global `yieldAsset` . A mismatch can cause denial of service on all yield accruals for that vault, as the merchant approves one asset while the strategy attempts to transfer another.
- In `ConcreteMerchant.initialize()` , `assetCTVault.asset()` is not validated against `address(assetCT)` . While `setAssetCTVault()` includes this validation, the same check is missing during initialization, allowing a mismatched vault to be set during deployment.
- In `ConcreteMerchant.setUserVault()` , the `userId` is not validated against `userAccounts[userId]` to ensure the user exists. This allows vaults to be assigned to non-existent users, causing `mintAndDeposit()` to revert with `UnknownUser` and making errors difficult to diagnose.

BVSS

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

Recommendation

Add validation checks:

- In `setVaultYieldConfig()` to ensure `yieldMultisig_` matches `strategy.getMultiSig()` and `strategy.getYieldAsset()` matches the merchant's global `yieldAsset` .
- In `initialize()` to ensure `assetCTVault.asset() == address(assetCT)` .
- In `setUserVault()` to ensure `userId` is mapped to a valid user account before allowing vault assignment.

Remediation Comment

SOLVED: The **Blueprint Finance team** solved this finding in the specified commit by adding extra validations and specifying the exception of:

> - **yieldMultisig_ is an operational wallet and can be by design different than the strategy multisig.**
> - **strategy vault address is checked and vault asset type is checked. During deployment strategy vault address must match the vault address and asset type.**

Remediation Hash

https://github.com/Blueprint-Finance/ASSETct/commit/a16ef4797d8fd6bc4c99fb5751cbc79517cb3909

7.8 GLOBAL QCTXTOAMOUNT MAPPING ENABLES CROSS FLOW DOS

// INFORMATIONAL

Description

The `ConcreteMerchant` contract uses a single `qcTxToAmount` mapping to track transaction hashes across four distinct operational flows: `mintAndDeposit()` , `finalizeQCRedemption()` , `finalizeYieldAccrual()` , and `finalizeAssetCtRedemption()` . This design enforces global uniqueness of transaction hashes across all processes.

However, once a transaction hash is consumed in one flow, it cannot be reused in any other flow, even though the operations are semantically unrelated.

For example:

1. An operator calls `mintAndDeposit()` with transaction hash `H` .
2. The hash `H` is stored in `qcTxToAmount[H]` .
3. Later, the operator attempts to call `finalizeYieldAccrual()` with the same hash `H` for an unrelated yield accrual event.
4. The transaction reverts with `TxHashAlreadyUsed(H)` , blocking the yield finalization.

This cross-flow coupling can lead to accidental denial-of-service if off-chain systems reuse hashes across different event types, or intentional griefing where an operator preemptively registers a hash in one flow to block its use in another.

BVSS

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

Recommendation

Consider a domain separation of the transaction hash keys by flow type using a namespace prefix. For example, compute the storage key as `keccak256(abi.encode(flowDomain, txHash))` where `flowDomain` is a unique identifier for each flow (e.g., `"MINT"` , `"YIELD"` , `"QC_REDEEM"` , `"ASSET_REDEEM"`).

This preserves replay protection within each flow while preventing cross-context collisions, using a single mapping without additional storage overhead.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** acknowledged this finding, stating that:

> This is a purposeful behaviour. The qc txHash should represent a qualified custodian transaction and must be 100% unique. It should never be reused. Each of the listed Merchant actions should be backed by a QC transaction as the proof of validity for a clean chain of accounting.

> Denial of service due to hash preemption is an extremely unlikely scenario and if someone would gain such unauthorised privilege it would be the least of our concerns. In case of unintentional reusing of a hash, however unlikely in our multi-signature independent entity cross validation setup, it would be our preference to block further execution requiring admin intervention and emergency response.

7.9 MISSING EVENTS

// INFORMATIONAL

Description

Throughout the contracts in scope, there are several instances where administrative functions change contract state by modifying core state variables without them being reflected in event emissions.

The absence of events may hamper effective state tracking in off-chain monitoring systems.

Instances of this issue can be found in:

- `ConcreteMerchant.setVaultYieldConfig()`
- `ConcreteMerchant.setAssetCTVault()`
- `ConcreteMerchant.setUserAccount()`
- `ConcreteMerchant.setUserVault()`
- `ConcreteMerchant.removeUserVault()`

BVSS

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

Recommendation

Emit events for all state changes that occur as a result of administrative functions to facilitate off-chain monitoring of the system.

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/ASSETct/commit/07d825c0d3d4911e0576afe68b8bd45d2e81f608

7.10 FLOATING PRAGMA

// INFORMATIONAL

Description

The contracts in scope currently use floating pragma version `^0.8.24` , which means that the code can be compiled by any compiler version that is greater than `0.8.0` , and less than `0.9.0` . 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` .

If deploying to networks other than Ethereum mainnet, the selected EVM version should be supported. Otherwise, unsupported opcodes may cause deployment failures or transaction issues.

BVSS

AO:A/AC:L/AX:L/R:N/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.

Remediation Comment

ACKNOWLEDGED: The **Blueprint Finance team** acknowledged this finding, stating that:

> We choose keep ^0.8.24 pragma allowing versions between ^0.8.24 and 9.0.0. We select the compiler version inside our foundry.toml file. We will take target evm version under considerations during deployments on new chains.

7.11 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 `ConcreteAssetCtVault`:

```
import {StateInitLib} from "@blueprint-finance/earn-v2-core/src/lib/StateInitLib.sol";
import {ConcreteV2RolesLib as RolesLib} from "@blueprint-finance/earn-v2-core/src/lib/Roles.sol";
```

- In `AssetCtMultisigStrategy`:

```
import {PositionAccountingStorageLib} from "@blueprint-finance/earn-v2-core/src/periphery/lib/Posit
import {PositionAccountingLib} from "@blueprint-finance/earn-v2-core/src/periphery/lib/PositionAcco
import {PeripheryRolesLib} from "@blueprint-finance/earn-v2-core/src/periphery/lib/PeripheryRolesLi
```

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/ASSETct/commit/9547dc0e3b87886e1a71a8c818abcd232da89aac

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.