// Security Assessment

09.02.2024 - 09.20.2024

HUB v1
Blueprint Finance

HALBORN

HUB v1 - Blueprint Finance

Prepared by: HALBORN

Last Updated Unknown date

Date of Engagement: September 2nd, 2024 - September 20th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
31	3	1	5	5	17

TABLE OF CONTENTS

1. Introduction

`Concrete` engaged our security analysis team to conduct a comprehensive security audit of their smart contract ecosystem. The primary aim was to meticulously assess the security architecture of the smart contracts to pinpoint vulnerabilities, evaluate existing security protocols, and offer actionable insights to bolster security and operational efficacy of their smart contract framework. Our assessment was strictly confined to the smart contracts provided, ensuring a focused and exhaustive analysis of their security features.

2. Assessment Summary

Our engagement with `Blueprint` spanned a 3-week period, during which we dedicated one full-time security engineer equipped with extensive experience in blockchain security, advanced penetration testing capabilities, and profound knowledge of various blockchain protocols. The objectives of this assessment were to:

- Verify the correct functionality of smart contract operations.

- Identify potential security vulnerabilities within the smart contracts.

- Provide recommendations to enhance the security and efficiency of the smart contracts.

In summary, `Halborn` identified several security concerns that were mostly addressed by the `Concrete team.`

3. Test Approach And Methodology

Our testing strategy employed a blend of manual and automated techniques to ensure a thorough evaluation. While manual testing was pivotal for uncovering logical and implementation flaws, automated testing offered broad code coverage and rapid identification of common vulnerabilities. The testing process included:

- A detailed examination of the smart contracts' architecture and intended functionality.

- Comprehensive manual code reviews and walkthroughs.

- Functional and connectivity analysis utilizing tools like Solgraph.

- Customized script-based manual testing and testnet deployment using Foundry.

This executive summary encapsulates the pivotal findings and recommendations from our security assessment of `Blueprint` smart contract ecosystem. By addressing the identified issues and implementing the recommended fixes, `Blueprint` can significantly boost the security, reliability, and trustworthiness of its smart contract platform.

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: sc_hub-v1

(b) Assessed Commit ID: 5ff8b67

(c) Items in scope:

- src/registry/RegistryManager.sol
- src/blueprints/implementations/LenderBlueprint.sol
- src/libraries/StorageHandlerLib.sol
- src/blueprints/implementations/ProtectionBlueprint.sol
- src/token/ERC721LogicContract.sol
- src/storage/ConcreteStorage.sol
- src/primitives/BasePongHandler.sol
- src/blueprints/implementations/PolicyTerminationBlueprint.sol
- src/primitives/ERC721_Internals.sol
- src/multiSigWallet/MultiSigWallet.sol
- src/blueprints/BaseBlueprint.sol
- src/config/ConfigManager.sol
- src/errors/Errors.sol
- src/primitives/CreateUserBlueprint.sol
- src/protocol/Protocol.sol
- src/primitives/EnableConcreteLite.sol
- src/constants/Tables.sol
- src/primitives/PacketIdHandler.sol
- src/storage/interfaces/IStorageOperations.sol
- src/primitives/EVMAddressValidation.sol
- src/primitives/GetConcreteLiteEncoding.sol
- src/accessControl/AccessControlManager.sol
- src/primitives/ValidateProtection.sol
- src/registry/RegistryManagerEvents.sol
- src/types/Enums.sol
- src/primitives/RemoteChainHandler.sol
- src/blueprints/BlueprintResolver.sol
- src/storage/ConcreteStorageConnector.sol
- src/blueprints/implementations/ProtectionBlueprintEvents.sol
- src/protocol/PauseStatus.sol
- src/registry/interfaces/IRegistryManager.sol
- src/primitives/LoanToken_OwnerOf.sol
- src/primitives/GetChainEndpoint.sol
- src/pongHandler/PongHandlerImplementation.sol
- src/constants/Namespaces.sol
- src/primitives/GetLoanTokens.sol

- src/primitives/GetEndpoint.sol
- src/primitives/SetBorrowToken.sol
- src/accessControl/OnlyRole.sol
- src/constants/ProtocolConstants.sol
- src/blueprints/implementations/PolicyTerminationBlueprintEvents.sol
- src/primitives/ERC721_Constructor.sol
- src/types/Structs.sol
- src/multiSigWallet/MultiSigWalletEvents.sol
- src/blueprints/implementations/LenderBlueprintEvents.sol
- src/blueprints/interfaces/IRegistry.sol
- src/storage/interfaces/IConcreteStorage.sol
- src/config/ConfigManagerEvents.sol
- src/storage/ConcreteStorageEvents.sol
- src/protocol/interfaces/IProtocol.sol
- src/blueprints/interfaces/IPongHandler.sol
- src/primitives/ERC721_Events.sol
- src/constants/Roles.sol
- src/accessControl/AccessControlManagerEvents.sol
- src/primitives/BasePongHandlerEvents.sol
- src/config/interfaces/IConfigManager.sol

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
3	1	5	5

INFORMATIONAL
17

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT BALANCE UPDATES IN ERC721LOGIC AND INTERNALS	CRITICAL	SOLVED - 09/26/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF ACCESS CONTROL IN PONG HANDLERS	CRITICAL	SOLVED - 09/19/2024
MISSING ACCESS CONTROL IN POLICY TERMINATION BLUEPRINT	CRITICAL	SOLVED - 09/19/2024
INCORRECT NAMESPACE USED ON BOOLEAN COMMIT	HIGH	SOLVED - 09/19/2024
MISSING VALIDATION FOR LOAN OWNER	MEDIUM	RISK ACCEPTED
LACK OF VALIDATION FOR ACCESSCONTROLMANAGER CONTRACT IN CONCRETESTORAGE	MEDIUM	SOLVED - 09/19/2024
MISSING CHECK FOR RESPONSE HANDLER ADDRESS	MEDIUM	SOLVED - 09/19/2024
MISSING HANDLING OF DELETE AND INCREMENT	MEDIUM	RISK ACCEPTED
MISSING OPERATIONS IN CONFIG AND REGISTRY PONG HANDLERS.	MEDIUM	SOLVED - 09/19/2024
MISSING NAME INITIALIZATION IN ERC721LOGIC CONSTRUCTOR	LOW	SOLVED - 09/19/2024
NON-ATOMIC PACKET ID MAY RESULT IN COLLISIONS	LOW	NOT APPLICABLE

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING UNDERFLOW HANDLING	LOW	RISK ACCEPTED
SINGLE STEP OWNERSHIP TRANSFER PROCESS	LOW	RISK ACCEPTED
MISSING VALIDATION FOR CONSISTENT CHAINID AND EID	LOW	RISK ACCEPTED
LACK OF CONFIGURABILITY IN MULTISIGWALLET	INFORMATIONAL	ACKNOWLEDGED
MISSING USE OF INTERNAL ERC721 FUNCTIONS	INFORMATIONAL	SOLVED - 09/19/2024
UNUSED CONFIG PONG HANDLER	INFORMATIONAL	SOLVED - 09/19/2024
USE OF HARDCODED VALUES INSTEAD OF ENUMS	INFORMATIONAL	SOLVED - 09/26/2024
INEFFICIENT ROLE CHECKING	INFORMATIONAL	SOLVED - 09/19/2024
UNNECESSARY IMMUTABLE NAMESPACE VARIABLE	INFORMATIONAL	SOLVED - 09/26/2024
HARDCODED VALUE INSTEAD OF ENUM	INFORMATIONAL	SOLVED - 09/19/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF DISTINCTION BETWEEN DELETE AND SETTING VALUE TO 0	INFORMATIONAL	ACKNOWLEDGED
ENTROPY REDUCTION MAY LEAD TO COLLISIONS	INFORMATIONAL	ACKNOWLEDGED
POTENTIAL HASH COLLISIONS IN NAMESPACE CONSTANTS DUE TO 4-BYTE LIMITATION	INFORMATIONAL	ACKNOWLEDGED
UNUSED FUNCTION IN CONFIGMANAGER	INFORMATIONAL	SOLVED - 09/19/2024
UNUSED FUNCTIONS IN REGISTRYMANAGER	INFORMATIONAL	SOLVED - 09/19/2024
EMPTY PACKET GAP	INFORMATIONAL	NOT APPLICABLE
REDUNDANT ONLYROLE MODIFIER	INFORMATIONAL	SOLVED - 09/19/2024
INEFFICIENT PLACEMENT OF AMOUNTSUPPLY CHECK	INFORMATIONAL	SOLVED - 09/19/2024
LACK OF EVENTS FOR STATE CHANGES	INFORMATIONAL	ACKNOWLEDGED
OWNERSHIP ASSUMPTIONS	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 INCORRECT BALANCE UPDATES IN ERC721LOGIC AND INTERNALS

// CRITICAL

Description

The `update` function in both `ERC721LogicContract` and `ERC721_Internals` contracts has a **critical issue** in the token balance update logic. Specifically, the balance is not properly updated in storage, as it only modifies in-memory variables without committing changes to storage using `_storage.setUint`. Additionally, the code incorrectly uses the balance key for the `from` address when attempting to update the `to` address balance, causing an inconsistency between ownership and token balances.
This issue results in token transfers that fail to properly decrement the `from` balance and increment the `to` balance, leading to a critical discrepancy in the contract's accounting of token ownership and balances. This discrepancy could easily lead to incorrect states where users own tokens, but the balances remain inaccurate, potentially causing significant issues in the protocol's token accounting and transfer mechanisms.

Proof of Concept

```
    function test_invalid_balance_erc721() external {
        bytes32 tokenId = keccak256(abi.encodePacked("test"));
        ERC721Token erc721 = new ERC721Token(tokenId, address(concreteStorage));

        vm.prank(ADMIN);
        accessControlManager.grantRole(bytes4(keccak256("COMMON")), address(erc721));

        erc721.mint(USER1, 100);

        assertEq(erc721.ownerOf(100), USER1);
        // ERROR: This will revert as the balance is not updated
        // Reason 1 is due to using a memory variable instead of storage
        // Reason 2 is due to using `from` instead of `to` for the second storage.
        assertEq(erc721.balanceOf(USER1), 1);
    }
```

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:M/D:C/Y:C/R:N/S:C (10.0)

Recommendation

To address this critical issue, the following changes should be made:
1. **Ensure storage updates**: Modify the `update` logic to call `_storage.setUint` to properly update balances in storage, not just in memory.
2. **Correct balance key usage**: When updating balances, ensure that the `from` address balance is decremented, and the `to` address balance is incremented by using the correct balance keys for both addresses. The current implementation mistakenly uses the `from` address balance key for both, which is incorrect.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by removing the `ERC721Logic` contract.

7.2 LACK OF ACCESS CONTROL IN PONG HANDLERS

// CRITICAL

Description

The `BasePongHandler` and `PongHandlerImplementation` contracts lack proper access control mechanisms on critical functions like `pongHandler`, `registryPongHandler`, and `configPongHandler`. These functions can be called by any external address, allowing arbitrary users to commit data into important namespaces (`REGISTRY` and `COMMON`) using any known `packetId`.
This means an attacker could: - Commit data with a success status when it should not be committed. - Prevent valid commits by passing an invalid `success` value. - Clear the `ACTIVE_PACKETS_HASHES` entry, causing the `packetId` to be processed incorrectly, leading to protocol state inconsistencies.
This vulnerability could result in unauthorized modifications of protocol-critical data, impacting the reliability and security of cross-chain communication or other inter-contract processes.

Proof of Concept

```
function test_pong_handler() external {
    vm.prank(ADMIN);
    accessControlManager.grantRole(bytes4(keccak256("COMMON_STAGE")), address(pongHandlerImplementati

    bytes32 packetId = keccak256(abi.encodePacked("test"));
    pongHandlerImplementation.pongHandler(packetId, 0);
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:C/D:C/Y:C/R:N/S:C (10.0)

Recommendation

Introduce strict access control, ensuring that only authorized addresses (e.g., app chain relayers) are allowed to call these functions. Implement an `onlyRole` modifier for these functions, or restrict access to an account that has validated the remote transaction event status.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by adding access control. A new role, `PONG_HANDLER_CALLER`, has been created and is required to call any pong handler function.

7.3 MISSING ACCESS CONTROL IN POLICY TERMINATION BLUEPRINT

// CRITICAL

Description

The functions `forecloseLite`, `forecloseBeforeExpiration`, and `reclaimOrForecloseAfterExpiration` in the `PolicyTerminationBlueprint` contract lack any form of access control protection. Without proper access control, any entity can call these functions to trigger foreclosure or reclamation operations on the remote chain for any loan, irrespective of whether they have the authority to do so.

This vulnerability can lead to several critical issues: - Unauthorized users can trigger foreclosure actions on loans they do not own. - Malicious actors could modify loan fee values, triggering unintended consequences across the system. - The protocol could suffer financial losses or inconsistencies by allowing foreclosure operations without proper checks.

Currently, there are no restrictions that limit who can call these functions, which opens up the system to exploitation. These functions should either be protected by access control or restricted to loan owners.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:C/D:C/Y:C/R:N/S:C (10.0)

Recommendation

1. **Implement Access Control:** Use role-based access control to limit who can call these sensitive functions. For instance, restricting these functions to be called only by the `BLUEPRINT_CALLER` role.
2. **Restrict to Loan Owner:** Alternatively, the functions should only be callable by the owner of the loan to prevent unauthorized access. This could be achieved by checking the loan ownership before proceeding.
3. **Combining Both Approaches:** The protocol can implement both access control and ownership checks for added security, ensuring only specific roles (like bots) or the loan owner can trigger these actions.

By applying these restrictions, the protocol ensures that only authorized entities can perform sensitive operations, reducing the risk of malicious exploitation and preserving the integrity of the loan management system.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by adding the access control.

7.4 INCORRECT NAMESPACE USED ON BOOLEAN COMMIT

// HIGH

Description

In the `RegistryManager` contract, the implementation of the `_commitNewBoolForAddressOnRemoteRegistry` function incorrectly uses the `replaceKeyNamespace(BYTES32, key)` function when updating the namespace. Since the function deals with boolean values, it should use the `replaceKeyNamespace(BOOL, key)` function instead. Using the wrong namespace can lead to inconsistencies in how the storage is accessed and managed, potentially causing incorrect data retrieval or unintended behavior in the protocol. This misalignment between the data type and the namespace could affect how boolean values are stored and accessed, causing logical errors in the contract.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (7.5)

Recommendation

Update the implementation of `_commitNewBoolForAddressOnRemoteRegistry` to use the correct namespace replacement function.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by removing the function.

7.5 MISSING VALIDATION FOR LOAN OWNER

// MEDIUM

Description

In the `ProtectionBlueprint` contract, the function `enableConcreteLite` does not validate if the `loanId_Owner` is set before proceeding. This missing validation could result in sending unnecessary cross-chain communication (CCCM) messages, which could impact the integrity of both on-chain and off-chain states. The commented-out lines in the code seem to be intended to perform this validation, but as it stands, the function can potentially send messages even when `loanId_Owner` is not set.
Moreover, several other functions within the contract use a `loanId` parameter but do not check if the associated loan actually exists. This could lead to unwanted behavior, such as attempting to perform operations on non-existent loans, affecting the protocol's state integrity.
These missing validations could open the door to unauthorized access, manipulation of loan-related data, or inconsistencies across the system.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:L/D:N/Y:N/R:P/S:C (6.6)

Recommendation

1. **Uncomment or add validation for** `loanId_Owner` : Ensure that the `loanId_Owner` is validated before proceeding with any logic that sends CCCM messages or modifies state.
2. **Check if loan exists in other functions:** For any function that takes `loanId` as a parameter, ensure that it checks if the loan exists. This can be done by checking if the loan is set in storage or by validating other key attributes related to the loan.
3. **Avoid sending unnecessary CCCM messages:** Ensure that CCCM messages are only sent when necessary and after proper validation checks. This will avoid unnecessary communication and maintain the integrity of the app chain and any off-chain state.
By enforcing these validations, you can prevent unauthorized loan operations, improve protocol security, and maintain data consistency across the system.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding. Checking the `concreteLite` status is enough; they do not need to add extra validation. If the loan exists, it should have `concreteLiteInfo` .

7.6 LACK OF VALIDATION FOR ACCESSCONTROLMANAGER CONTRACT IN CONCRETESTORAGE

// MEDIUM

Description

In the `ConcreteStorage` contract, during the `updateAccessControl` function, the provided `accessControlManagerContract_` is assumed to be a valid `AccessControlManager` without any validation. If a non-conforming contract or an invalid address is provided, the protocol could end up without a functioning access control manager. This could result in the inability to upgrade contracts, revoke roles, or control access, leaving the protocol vulnerable to unauthorized actions.
Without proper validation, an attacker or a mistake could set an invalid address, permanently affecting the ability to manage roles and permissions, effectively locking the protocol or granting unauthorized access.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:P/S:C (6.3)

Recommendation

Ensure that the `accessControlManagerContract_` is a valid `AccessControlManager` by implementing a validation check. This can be done through `ERC-165` interface detection or by calling a function that proves the contract conforms to the expected role management behavior. For example, add a `require` check:

```
require(AccessControlManager(accessControlManagerContract_).hasRole(ROLES, accessControlManagerContra
```

Alternatively, check for the existence of an admin role:

```
require(AccessControlManager(accessControlManagerContract_).hasRole(ACCESS_CONTROL_ADMIN, accessContr
```

Either check would ensure that the provided contract has the appropriate roles and adheres to the expected `AccessControlManager` contract interface.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by adding a `require` condition: if the contract doesn't have the function, it will revert.

7.7 MISSING CHECK FOR RESPONSE HANDLER ADDRESS

// MEDIUM

Description

In the `BaseBlueprint` contract, the `_getReponseTemplate` function is responsible for fetching the response handler address from storage. This address is used when sending a cross-chain messaging (CCCM) response. However, there is no validation to ensure that the address retrieved from storage is not the zero address (`0x0`). If an invalid or uninitialized address is used, it can result in the CCCM message failing to find a valid handler, leading to unexpected behavior or failure to process the message.
Failure to check the validity of this address could result in invalid messages being sent with no recipient to handle them. This could disrupt the protocol's cross-chain operations, particularly when expecting a response from another chain.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:P/S:C (6.3)

Recommendation

Add a check in the `_getReponseTemplate` function to ensure that the response handler address is valid and not set to `0x0` . If the address is invalid, revert the transaction to prevent sending a faulty CCCM message.

Remediation Comment

SOLVED: The **Concrete team** solved the issue by adding an if condition to revert if the returned address is zero.

7.8 MISSING HANDLING OF DELETE AND INCREMENT

// MEDIUM

Description

In the `BasePongHandler` contract, the `_basePongHandler` function is responsible for handling various packet operations. However, it does not handle two critical operations: `DELETE_UINT256` and `INCREMENT_UINT256`.

- The `DELETE_UINT256` operation should result in the deletion of a `uint256` value from storage. Without handling this case, the deletion will not be performed as expected, leaving stale data in the storage.
- The `INCREMENT_UINT256` operation should increment a `uint256` value in storage. However, without handling it, the expected increment does not occur, leading to protocol logic failures where increments are expected.

Failing to handle these cases could lead to data inconsistencies and incorrect protocol behavior.

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:C (6.3)

Recommendation

Add handling for `DELETE_UINT256` and `INCREMENT_UINT256` operations in `_basePongHandler`.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding. Those instructions do not create a key in `COMMON_STAGE` to be committed later; they only create the key with the instruction inside the `packetKeys` array. This is because they do not need to “move” any value from stage, we only need to either set it to 0 or add 1.

7.9 MISSING OPERATIONS IN CONFIG AND REGISTRY PONG HANDLERS.

// MEDIUM

Description

In the `BasePongHandler`, the `_configPongHandler` and `_registryPongHandler` do not handle instructions for certain data types and operations, including `ADDRESS`, `INCREMENT_UINT256`, `AGGREGATE_UINT256`, `SUBSTRACT_UINT256`, and `DELETE_UINT256`. These instructions are fundamental for managing various types of storage updates across the protocol.
Without proper handling of these instructions, important protocol operations involving address mappings, numeric aggregations, and incremental updates are either ignored or lead to incorrect storage states.
Specifically: - The `ADDRESS` type is used to handle data from different chain formats (e.g., bytes32) and will play a critical role in future multi-chain compatibility. - For `INCREMENT_UINT256`, `AGGREGATE_UINT256`, and `SUBSTRACT_UINT256`, the protocol risks silent failures for aggregation and arithmetic operations on stored values. - `DELETE_UINT256` should either delete or mark the value as deleted, but no logic currently exists to handle such cases.
The absence of handling for these instructions can lead to security vulnerabilities, inconsistencies, and protocol failures, especially when managing cross-chain operations or multi-chain compatibility.

BVSS

AO:A/AC:L/AX:L/C:N/I:M/A:N/D:N/Y:N/R:N/S:C (6.3)

Recommendation

1. **Implement missing instruction handling:** Update `_configPongHandler` and `_registryPongHandler` to handle the instructions.
2. **Address handling**: Since `ADDRESS` is mostly used in the form of `bytes32` for future compatibility with non-EVM chains, ensure that `ADDRESS` instructions either handle the value correctly or trigger an appropriate "not implemented" error if this case is yet to be supported.
3. **Non-implemented instructions**: For any instructions that are intentionally not implemented, add a revert statement to indicate the lack of support, avoiding silent failures. This could prevent confusion or exploitation due to unhandled cases.
By implementing these changes, the protocol will ensure proper handling of all critical instructions and maintain the integrity and security of its storage operations.

Remediation Comment

SOLVED: The **Concrete team** solved this issue. `BasePongHandler` has been refactored to account for all instructions in every function

References

7.10 MISSING NAME INITIALIZATION IN ERC721LOGIC CONSTRUCTOR

// LOW

Description

In the `ERC721Logic` contract, there is no name or symbol set during the contract initialization. This is problematic, as ERC-721 tokens usually require a name and symbol to provide clear identification for users interacting with the contract. Without this, the token may be harder to identify and interact with, especially through user interfaces or external applications that expect these fields to be set.
The constructor should ideally inherit from the `ERC721_Constructor` and initialize the token name and symbol properly using the `_erc721_Constructor` function, which ensures these fields are correctly set. Failing to set a name and symbol could result in confusion, particularly for users or other contracts that rely on these fields for identifying the token.

Proof of Concept

```
address constant ADMIN = address(0x1337);

ConcreteStorage public concreteStorage;

function setUp() external {
    concreteStorage = new ConcreteStorage(ADMIN);
}

function test_no_name_erc721() external {
    bytes32 tokenId = keccak256(abi.encodePacked("test"));
    ERC721Logic erc721 = new ERC721Logic(tokenId, address(concreteStorage));

    console.log(erc721.name());
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:P/S:U (3.8)

Recommendation

Refactor the `ERC721Logic` contract to inherit from `ERC721_Constructor` and initialize the name and symbol during the constructor call. Use the `_erc721_Constructor` to ensure that these values are set properly.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing the contract.

7.11 NON-ATOMIC PACKET ID MAY RESULT IN COLLISIONS

// LOW

Description

In the `PacketIdHandler` contract, the `endpoint.getPacketId` function is used to generate packet IDs, but it requires manual nonce increment via `endpoint.incrementNonce()`. If the developer forgets to call `incrementNonce`, or if the function fails or is bypassed, it is possible that the same packet ID could be generated more than once. This would lead to potential packet collisions, where two different transactions could end up with the same packet ID, resulting in incorrect data being handled by the contract.
A collision in packet IDs can lead to issues such as overwriting state, mismatched packets, and even replay attacks, depending on the implementation of the packet handling system.

BVSS

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

Recommendation

The `endpoint.getPacketId` function should atomically increment the nonce internally, ensuring that each call generates a unique packet ID without requiring the developer to manually call `incrementNonce()`. This can be achieved by updating the underlying `getPacketId` function within the endpoint contract to automatically manage nonce increment.

Remediation Comment

NOT APPLICABLE: The **Concrete team** marked this as not applicable. They manually increase the nonce only in case of a hash collision, as we are using only 4 bytes. Either way, the packet is created when the message is sent in the CCCM. Before that, it's only being fetched. We increase the nonce only in cases of a `bytes4(hash(packetId))` collision.

7.12 MISSING UNDERFLOW HANDLING

// LOW

Description

In the `BasePongHandler` contract, the `_basePongHandler` function handles the `SUBSTRACT_UINT256` operation when the transaction is successful. However, the current implementation does not check if the value to be subtracted (`uintValue`) is greater than the value stored in `_storage.getUint(key)` . Since `uint256` does not allow negative numbers, this can cause an underflow, which would lead to incorrect values being stored (due to wrapping in Solidity), potentially causing severe inconsistencies in the protocol.
If the stored value is less than `uintValue` , the function should either: 1. Revert with a custom error to prevent the underflow. 2. Set the value to 0, but this can lead to further inconsistencies as the result does not reflect the actual intent of the operation.
Alternatively, the protocol could consider using signed integers (`int256`) to handle these cases if negative values are a valid outcome of the operation.

BVSS

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

Recommendation

Add an underflow check to ensure that the stored value is greater than or equal to `uintValue` before performing the subtraction. If not, revert the transaction with a custom error.
If the possibility of a negative value is acceptable, consider using `int256` instead of `uint256` to prevent such issues.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding. There should not be a case where it fails because the value is checked before being sent. However, if it's not, it will result in a failure on the remote chain, so the success flag will be 0. If that's not the case, the operation will simply revert in the event of an underflow.

7.13 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

It was identified that the `ConcreteStorage` contract inherited from OpenZeppelin's `Ownable` library. Ownership of the contracts that are inherited from the `Ownable` module can be lost, as the ownership is transferred in a single-step process. The address that the ownership is changed to should be verified to be active or willing to act as the `owner`. `Ownable2Step` is safer than `Ownable` for smart contracts because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer only completes when the new owner accepts ownership.

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}

function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

BVSS

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

Recommendation

Consider using the `Ownable2Step` https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol library over the `Ownable` library.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding.

7.14 MISSING VALIDATION FOR CONSISTENT CHAINID AND EID

// LOW

Description

In the `RegistryManager` contract, the function `addEndpointIdToChainId` is not enforcing validation to check if `chainId_` and `eid_` (Endpoint ID) are equal, which is implied as a requirement by other contract logic. This can cause issues, especially in the context of the function `_getRemoteChainEid` in `RemoteChainHandler`. If a mismatch or incorrect mapping occurs, it could lead to returning invalid or duplicate addresses for different chain IDs through the `_getRemoteChainEidAddress` function. Specifically, if the same `eid_` is assigned to multiple chain IDs or if the `REMOTE_ENDPOINT_TO_ADDRESS` mapping isn't properly verified before setting values, the protocol could retrieve wrong data, potentially affecting the functionality across different chains.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:F/S:C (2.3)

Recommendation

To prevent invalid mappings and ensure that each chain ID corresponds to its own endpoint ID, you should:

- Add a check in `addEndpointIdToChainId` to ensure that `chainId_` and `eid_` are equal, or at least enforce a validation logic that guarantees no duplicate endpoint IDs are assigned to multiple chains.
- Before setting or updating the endpoint or chain mappings, ensure that `REMOTE_ENDPOINT_TO_ADDRESS` or `REMOTE_CHAIN_TO_REGISTRY` are properly validated to avoid un-synced or duplicate mappings.

Remediation Comment

RISK ACCEPTED: The **Concrete team** accepted the risk of this finding. Names that could cause confusion were changed. There is a 1-to-n relationship between chainId and eid. Usually, there will be only one eid per chainId, but the distinction is made in case we want to have more than one deployment per chain. So we will only have one active eid per chain.

7.15 LACK OF CONFIGURABILITY IN MULTISIGWALLET

// INFORMATIONAL

Description

The `MultiSigWallet` contract has a few significant limitations in its current implementation:

1. **No way to change** `_numConfirmationsRequired`: The number of confirmations required to execute a transaction is set during contract construction but cannot be updated afterward. This is problematic because the initial setting could be too low or too high for the wallet's evolving needs. Additionally, the contract allows `_numConfirmationsRequired` to be set to 1 by default, which effectively bypasses the multisig functionality, allowing a single owner to execute transactions unilaterally.
2. **No quorum enforcement:** There is no enforced quorum that requires a minimum percentage of owners (such as 50%) to confirm a transaction. This can result in decisions being made by a tiny subset of the owners, undermining the purpose of a multisig wallet.
3. **No option to add or remove owners:** Once the contract is deployed, the set of owners is immutable. There is no functionality to add new owners, remove current ones, or handle owner changes dynamically. This is a critical flaw, as it limits the contract's ability to adapt to changes, such as a current owner becoming inactive or new stakeholders joining.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:H/Y:N/R:N/S:U (1.9)

Recommendation

1. **Introduce functionality to change** `_numConfirmationsRequired`: Add a function that allows the owners to modify the required number of confirmations. This function should have appropriate safeguards, such as requiring approval from a majority of the current owners.
2. **Enforce a quorum:** Consider implementing a quorum mechanism that requires at least 50% of the owners to confirm a transaction, regardless of the current `_numConfirmationsRequired`.
3. **Add functions for owner management:** Implement functions to allow adding and removing owners, with appropriate confirmation from the existing owners.

These changes will improve the flexibility, security, and long-term utility of the `MultiSigWallet` contract.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding. The current multisig wallet will not be used, should be out of scope.

7.16 MISSING USE OF INTERNAL ERC721 FUNCTIONS

// INFORMATIONAL

Description

The `ERC721Logic` contract does not utilize the internal ERC-721 functions available in the `ERC721_Internals` contract, such as `_erc721_mint`, `_erc721_update`, `_erc721_ownerOf`, and others. These functions provide crucial functionality for managing token ownership, approvals, and other standard ERC-721 behaviors. By not using these internal functions, the contract risks inconsistent or incomplete token management, leading to potential errors or vulnerabilities in how the token operates. The contract should be leveraging these internal functions to properly handle minting, updating, ownership verification, approvals, and more, as these are well-tested and designed for reuse in ERC-721 implementations.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:F/S:U (1.9)

Recommendation

Refactor the `ERC721Logic` contract to utilize the internal functions from `ERC721_Internals` to handle token minting, updating, and other operations. This ensures that the contract adheres to the ERC-721 standard and avoids duplicating logic. Implement the following changes:

- Use `_erc721_mint` for minting functionality.
- Use `_erc721_update` for token transfers or updates.
- Use `_erc721_ownerOf` for retrieving token ownership.
- Use `_erc721_isApprovedForAll` for checking operator approvals.
- Use `_erc721_isAuthorized` for checking whether a user is authorized for specific token operations.
- Use `_erc721_getApproved` for retrieving token-specific approvals.
- Use `_erc721_checkAuthorized` to verify authorization for token operations.
- Use `_erc721_approve` for approving token transfers.
- Use `_erc721_requireOwned` to check token ownership internally.

Additionally, expose an internal `_mint` function that utilizes `_erc721_mint` to enable proper token minting.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing the use of this contract, keeping only the `ownerOf` function.

7.17 UNUSED CONFIG PONG HANDLER

// INFORMATIONAL

Description

The `configPongHandler` function is currently implemented but not used anywhere in the code. The `ConfigManager` contract is using the `registryPongHandler` selector to handle configuration updates instead of the `configPongHandler`. As a result, there is unnecessary duplication of logic, and the presence of an unused function increases the contract size and complexity without adding value. Additionally, both `configPongHandler` and `registryPongHandler` have identical logic. Maintaining unused and redundant code increases the risk of maintenance errors and makes the codebase harder to audit and manage.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:C (1.9)

Recommendation

Either:

1. **Use the** `configPongHandler` in the `ConfigManager`: If `configPongHandler` was intended to handle configuration updates specifically, modify the `ConfigManager` to use this selector rather than `registryPongHandler`. This will ensure that the contract logic is clearly separated and adheres to its intended design.
2. **Remove the** `configPongHandler` entirely: If no distinction is necessary between configuration and registry updates, you can simplify the code by removing `configPongHandler` from the contracts. This will reduce contract size and improve maintainability.

Remediation Comment

SOLVED: The **Concrete team** solved this issue. The `configPongHandler` is being used on the `ConfigManager` and the `registryPongHandler` function on the `RegistryManager`.

7.18 USE OF HARDCODED VALUES INSTEAD OF ENUMS

// INFORMATIONAL

Description

In the `EVMAddressValidation` contract, the `_evmAddressValidation` function checks the `tokenConfig_` by comparing values using `uint` numbers, specifically `_storage.getUint(key) < 3` to ensure that a token is not blacklisted. However, this approach relies on magic numbers, which makes the code less readable and prone to errors when interpreting the various token configurations.
The system is already using an enum called `TokenConfig` with different states such as `AvailableWithNoProtection`, `AvailableForConcreteLite`, `AvailableForProtection`, and `Blacklisted`. Using the enum directly instead of magic numbers improves code clarity, maintainability, and reduces the chance of misinterpretation when developers work with the code or modify it.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (1.5)

Recommendation

Refactor the comparison of the `tokenConfig_` to use the `TokenConfig` enum instead of comparing with raw integer values. This will provide better readability and ensure that future updates are more manageable.

```
if (_storage.getUint(key) != uint256(TokenConfig.Blacklisted)) {
    // logic
}
```

By referencing the enum, the code becomes self-explanatory, reducing the chance of misinterpreting what each value represents and ensuring that the logic remains clear across the protocol. Ensure all instances where `tokenConfig_` values are checked across the protocol use the enum rather than magic numbers.

Remediation Comment

SOLVED: The **Concrete team** solved this issue. All instances of the `TokenConfig` are now using the enum values.

7.19 INEFFICIENT ROLE CHECKING

// INFORMATIONAL

Description

The `hasRole` function in the `AccessControlManager` contract currently relies on fetching the `rolesVersion` externally from the storage and then checking if the role is set using `_storage.getBool`. This results in two external calls to the storage: one to retrieve the `rolesVersion` and another to check if the role exists. This introduces unnecessary gas costs due to the repeated calls to the storage layer.
By using `_storage.hasRole` directly, the function can efficiently check for the role in a single call, as this method internally fetches the `rolesVersion` and checks the storage, optimizing gas consumption.

BVSS

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

Recommendation

Refactor the `hasRole` function to call `_storage.hasRole` directly instead of making two separate external storage calls. This will reduce gas costs and simplify the logic.

Remediation Comment

SOLVED: The **Concrete team** solved this issue. Refactored the `hasRole` function to call `_storage.hasRole` directly instead of making two separate external storage calls.

7.20 UNNECESSARY IMMUTABLE NAMESPACE VARIABLE

// INFORMATIONAL

Description

The `ERC721Logic` contract currently uses the `StorageConnectorWithNamespace` contract, which includes an immutable `NAMESPACE` variable. However, `ERC721Logic` does not leverage the immutable `NAMESPACE` variable efficiently, leading to unnecessary gas costs due to the inclusion of additional logic and storage that may not be required for this particular implementation.
Since `ERC721Logic` does not utilize multiple namespaces and staging logic, the usage of `StorageConnectorPrimitive`, which omits the namespace logic, would result in reduced deployment costs by simplifying the contract structure and eliminating unnecessary storage.

BVSS

AO:S/AC:L/AX:L/C:N/I:M/A:L/D:N/Y:N/R:N/S:U (1.1)

Recommendation

Refactor `ERC721Logic` to inherit from `StorageConnectorPrimitive` instead of `StorageConnectorWithNamespace`. This will reduce deployment costs by avoiding the additional logic related to the immutable `NAMESPACE` variable, which is not necessary for this use case.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing the `ERC721Logic` contract.

7.21 HARDCODED VALUE INSTEAD OF ENUM

// INFORMATIONAL

Description

In the `LenderBlueprint` contract, the `_initialSupplyPrimitive` function, which internally calls `_evmAddressValidation`, uses a hardcoded value of `1` to represent a token configuration. This is meant to indicate the `AvailableForConcreteLite` configuration but lacks clarity and could lead to misinterpretations in future code changes.
Using hardcoded values decreases code readability and maintainability, making it difficult for developers to understand what the `1` value represents. This can also introduce potential errors if the enum values change in the future or if a developer misinterprets the hardcoded value.

BVSS

AO:S/AC:L/AX:L/C:N/I:M/A:N/D:L/Y:N/R:N/S:U (1.1)

Recommendation

Replace the hardcoded value `1` with the `TokenConfig.AvailableForConcreteLite` enum to enhance readability and maintainability. This will also make the code less error-prone, as the enum provides a more explicit representation of the token configuration.

Remediation Comment

SOLVED: The **Concrete team** solved this issue. Magic numbers have been replaced by the corresponding enums.

7.22 LACK OF DISTINCTION BETWEEN DELETE AND SETTING VALUE TO 0

// INFORMATIONAL

Description

The `DELETE_UINT256` operation in the `_basePongHandler` function simply deletes a `uint256` value from storage, which effectively has the same outcome as setting the value to 0. This can lead to ambiguity in the protocol, as there is no way to distinguish between a deleted value and a value that was intentionally set to 0. Without a clear distinction, it could cause issues in other parts of the protocol that expect a deleted value to behave differently from a zeroed value.
For example, in certain financial applications, a value of 0 could mean "no debt" or "no funds," while a deleted value might indicate that the entity no longer exists or that the record is invalid. Without a mechanism to indicate the value was deleted, it could lead to incorrect assumptions or logic failures.

BVSS

AO:S/AC:L/AX:L/C:N/I:M/A:L/D:N/Y:N/R:N/S:U (1.1)

Recommendation

Consider introducing a flag to indicate that the value was deleted and is no longer valid. This flag could be stored in a separate boolean variable in storage to mark the deletion status.
This way, the protocol can differentiate between values that are intentionally set to 0 and values that have been deleted. This ensures more robust logic and avoids unintended behavior when handling deleted entries.
If a 0 value is preferred, the protocol should always check for it during its logic.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding. There is a distinction: DELETE is an optimization because it does not require reading a key from a stage to determine which value to set. It will always set the value to zero, so it saves us from reading from storage.

7.23 ENTROPY REDUCTION MAY LEAD TO COLLISIONS

// INFORMATIONAL

Description

The `StorageHandlerLib` contract uses the `_createKey` function to generate unique keys by combining a `namespace` and a truncated hash, `pointerPlusIdHash`. The `pointerPlusIdHash` is reduced from 256 bits to 192 bits by shifting 64 bits to leave space for a potential `packetId` in the `_createKeyWithPacketId` function. While this approach helps accommodate packet IDs, it reduces the entropy of `pointerPlusIdHash`, which could increase the chance of hash collisions.
Despite the reduced entropy, collisions are still improbable because the `pointerPlusIdHash` is always derived from hashed data. However, the reduced entropy may still introduce risk in high-collision probability scenarios, such as when dealing with a large number of unique entries in the storage.

BVSS

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

Recommendation

Consider using the full 256-bit hash for `pointerPlusIdHash` to preserve the full entropy and mitigate any potential risks of hash collisions. If a reduction is necessary to support packet IDs, implement additional mechanisms to detect or mitigate collisions, such as pre-checking for existing keys in the storage system before assigning a new one.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this issue. Addresses are 20 bytes, and there is not a concern about collisions. To construct the keys, it is also used the first 4 bytes of the NAMESPACE, which further reduces the chance of collisions.

7.24 POTENTIAL HASH COLLISIONS IN NAMESPACE CONSTANTS DUE TO 4-BYTE LIMITATION

// INFORMATIONAL

Description

In the `Namespaces` contract, only the first 4 bytes of the hash are being used to define namespaces (e.g., `bytes4 constant ROLES = bytes4(keccak256("ROLES"));`). This truncation significantly increases the likelihood of hash collisions since only 4 bytes (32 bits) are used to distinguish between different namespaces. As more namespaces or data types are added in the future, the probability of collisions increases, potentially leading to namespace overlap, which may cause unexpected behavior or vulnerabilities in the system.

BVSS

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

Recommendation

It is essential to check for potential hash collisions before adding new namespaces. This can be done by manually verifying the first 4 bytes of the hash value when defining new namespaces. Including the calculated hex values in the natspec or as comments for each namespace will help ensure that no accidental collisions occur. For example:

```
/// @dev ROLES namespace (first 4 bytes: 0x5b5298e6)
bytes4 constant ROLES = bytes4(keccak256("ROLES"));

/// @dev COMMON namespace (first 4 bytes: 0xaabbccdd)
bytes4 constant COMMON = bytes4(keccak256("COMMON"));
```

This practice will provide a simple way to verify and avoid future hash collisions when adding new namespaces.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this issue. Function signatures use the first 4 bytes, and there are likely more chances of having multiple function signatures in a single contract than there are namespaces in the entire system.

7.25 UNUSED FUNCTION IN CONFIGMANAGER

// INFORMATIONAL

Description

The `ConfigManager` contract contains a private function named `_commitNewUintForRemoteRegistry`, which is currently not being used anywhere in the contract. Unused functions not only increase deployment costs but can also confuse developers and auditors, making it harder to maintain or audit the codebase.
Unused code, especially functions, can lead to unnecessary complexity and potential attack vectors if not properly managed.

BVSS

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

Recommendation

Remove the `_commitNewUintForRemoteRegistry` function if it is not required in the contract. If the function is intended for future use, consider commenting it out or explaining its purpose in the code comments to avoid confusion.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing unused functions.

7.26 UNUSED FUNCTIONS IN REGISTRYMANAGER

// INFORMATIONAL

Description

In the `RegistryManager` contract, the following functions are declared but remain unused throughout the contract:

- `_commitNewUintForRemoteRegistry`
- `_commitNewBoolForAddressOnRemoteRegistry`
- `addBoolRegistry`
- `removeBoolRegistry`

Unused functions introduce unnecessary complexity, increase the contract size, and raise the potential for confusion or future security issues. Additionally, these functions may consume unnecessary gas costs during contract deployment.

BVSS

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

Recommendation

If these functions are not intended to be used in the future or have been deprecated, it is recommended to remove them from the contract to reduce contract size and avoid potential confusion.
However, if these functions are planned for future use, consider adding comments explaining their purpose and ensuring they are properly tested and implemented before use.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing unused functions.

7.27 EMPTY PACKET GAP

// INFORMATIONAL

Description

In the `LenderBlueprint` contract, the `_initialBorrow` function introduces an issue when `borrowToken == address(0)`. In this case, the `packetsArrays.packetsIdx` is set to a fixed value. However, subsequent packet filling increments `packetsArrays.packetsIdx` without properly handling the case when `borrowToken` is set to the zero address. This results in an empty gap in the `packetsArrays.packets` array, which can lead to inconsistencies in the data being sent between contracts.
This could cause issues with cross-chain communications or the handling of messages, as the missing packet could lead to unexpected behavior or misinterpretation of packet data.

BVSS

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

Recommendation

Introduce logic to skip the unnecessary increment of `packetsArrays.packetsIdx` when `borrowToken == address(0)` to ensure that no empty packet gaps are created.

Remediation Comment

NOT APPLICABLE: The **Concrete team** marked this issue as not applicable. During `_storeInTemp` and `_sendMultiMessage` functions, the construction of the array of keys has been optimized to eliminate gaps.

References

7.28 REDUNDANT ONLYROLE MODIFIER

// INFORMATIONAL

Description

In the `RegistryManager` contract, the internal functions `_removeRemoteTokenToRegistry` and `_addRemoteTokenToRegistry` have the `onlyRole(REGISTRY_ADMIN)` modifier applied. This is redundant because these internal functions are typically called by external or public functions that already enforce role-based access control. Having the `onlyRole` modifier on both the external caller and the internal function results in unnecessary code duplication and adds extra gas costs.
It is more efficient to apply the `onlyRole` modifier either at the external/public level or at the internal function level, but not both. Since these functions are internal, the access control should only be applied to the public or external-facing functions, ensuring consistency and reducing unnecessary computation.

BVSS

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

Recommendation

Remove the `onlyRole(REGISTRY_ADMIN)` modifier from the internal functions `_removeRemoteTokenToRegistry` and `_addRemoteTokenToRegistry`. Ensure that the external functions that call these internal functions are properly protected with access control checks.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by removing the modifier.

7.29 INEFFICIENT PLACEMENT OF AMOUNTSUPPLY CHECK

// INFORMATIONAL

Description

In the `LenderBlueprint` contract, the `_initialSupplyPrimitive` function performs a validation on `amountSupply_` only after several other checks and logic have been executed. The `amountSupply_` check verifies that the supplied amount is greater than zero, which is a fundamental requirement for continuing the process. If this check fails, all prior logic would have been unnecessarily executed. This leads to inefficiency in the contract's execution, as checks on fundamental parameters like `amountSupply_` should be performed at the earliest possible point to save gas and prevent wasted computations.

BVSS

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

Recommendation

Move the `require` check for `amountSupply_ > 0` to the beginning of the function, along with other error checks. This will improve the efficiency of the contract by ensuring that the function halts early if the supplied amount is invalid, thereby saving unnecessary computational costs.

Remediation Comment

SOLVED: The **Concrete team** solved this issue by checking for `amountSupply_ > 0` moved to the beginning of the function.

7.30 LACK OF EVENTS FOR STATE CHANGES

// INFORMATIONAL

Description

Important state-changing functions such as `setAddress`, `setUint`, `setString`, etc., do not emit events. This can make it challenging to track changes and debug issues.

BVSS

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

Recommendation

Consider emitting events in all state-changing functions.

```
event AddressSet(bytes32 indexed key, address value);
event UintSet(bytes32 indexed key, uint256 value);
event StringSet(bytes32 indexed key, string value);
// Emit these events in respective functions
```

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

7.31 OWNERSHIP ASSUMPTIONS

// INFORMATIONAL

Description

The contract uses `Ownable`, and assumes that the `multisig_` provided in the constructor will always be secure and correctly managed. If this address is compromised, the whole storage system can be at risk.

BVSS

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

Recommendation

Consider the multisig address is always managed securely. Implement additional checks if necessary.

Remediation Comment

ACKNOWLEDGED: The **Concrete team** acknowledged this finding.

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.