Arcadia core contracts
Security Review

Cantina Managed review by:

R0bert, Lead Security Researcher
Windhustler, Security Researcher

April 21, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	**Description**
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Khalani is the infrastructure platform to build intent-driven solver networks that evolve with your users' dynamic needs.

From Feb 19th to Mar 8th the Cantina team conducted a review of arcadia-core-contracts on commit hash 1acd2718. The team identified a total of **32** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	8	8	0
High Risk	3	3	0
Medium Risk	6	6	0
Low Risk	10	6	4
Gas Optimizations	0	0	0
Informational	5	2	3
Total	**32**	**25**	**7**

3 Findings

3.1 Critical Risk

3.1.1 Empty `fillGraph` Allows Malicious Solvers to Steal Assets

Severity: Critical Risk

Context: SolutionLib.sol#L221

Description: In the `IntentBook.solve` function, a malicious solver could process a solution that extracts assets from a user's intent without meeting the intent's outcome requirements. This happens because the `fillGraph`, which should connect receipts or child intents to fulfill the outcome, can remain empty, bypassing essential verification.

Consider a `PctFilled` intent where Alice offers 1000 `mToken` to receive 1000 `mToken2`. A solver, Bob, can exploit this by submitting a solution that:.

- Consumes all 1000 `mToken` from Alice's intent.
- Issues a receipt for 1000 `mToken` to himself.
- Provides an empty `fillGraph`, delivering no `mToken2` to Alice.

The `checkIntentSatisfaction` function in `SolutionLib` fails to enforce a non-empty `fillGraph`, allowing the solution to pass. Specifically, in `_checkFilledPAS`, an empty `fillGraph` results in:.

- `_validateOutputIntent` setting `finalIntentSrcTokenBalance` to 0.
- `_validateReceipts` computing `receiptTotal` as 0.
- `_validateBalancesAndAmounts` calculating `tokensTakenFromInput` as 1000 without requiring a non-zero `receiptTotal`.

Consequently, the `spendGraph` transfers 1000 `mToken` to Bob's receipt and the intent is marked as "*Solved*", leaving Alice with no tokens or outcome.

Recommendation: Consider updating the `_checkFilledPAS` function to mandate a non-empty `fillGraph` for `PctFilled` intents. Implement a validation check that reverts the transaction if no `FillRecord` is provided to fulfill the intent's outcome.

Khalani: Fixed in commit c7da976.

Cantina Managed: Fix verified.

3.1.2 Lack of Sender Validation in `EventVerifier.handle` Function

Severity: Critical Risk

Context: EventVerifier.sol#L33

Description: The `handle` function in the `EventVerifier` contract processes messages from another chain using Hyperlane, a cross-chain messaging system. This function accepts parameters such as `_origin`, `_sender` and `_message`. However, it does not verify whether the `_sender` is an authorized entity. This means any account or contract on the origin chain can send messages to the hub chain, and the `EventVerifier` will process them without checking their legitimacy. In a secure cross-chain system, ensuring that only trusted senders, like the `SpokePublisher` contract, can send messages is essential. The absence of this validation creates a major security gap, as the system cannot differentiate between genuine and malicious messages.

Without this validation, unauthorized parties can send messages to the hub chain, causing fraudulent changes or triggering unintended actions within the `EventVerifier`. A malicious actor could mimic a trusted sender, such as the `SpokePublisher` and manipulate the system.

Recommendation: To address this issue, the `handle` function should include a check to confirm that the `_sender` is an authorized address before processing the message. One effective way to do this is by maintaining a list of trusted senders, such as the `SpokePublisher` contract on the origin chain, and validating against it. For instance, the contract could use a mapping to track authorized senders and enforce this restriction. On the other hand, consider allowing the processing of messages only from whitelisted chains.

Khalani: Fixed in commit 818f111.

Cantina Managed: Fix verified.

3.1.3 Missing Validation Checks In `redeem` Function

Severity: Critical Risk

Context: ReceiptManager.sol#L40

Description: The `redeem` function implemented in the `ReceiptManager` contract allows anyone to claim a `receiptId` on behalf of the `receipt.owner`, transferring the `MTokens` directly to the `receipt.owner`. However, receipts can be redeemed before they are even created or initialized. Once redeemed, `s_lockedReceipts[receiptId]` mapping is set to `true` reflecting that the receipt was redeemed even if the redemption was totally empty.

This lack of access control and validation creates an opportunity for a front-running attack. An attacker can anticipate future `receiptId` values and call `redeem` before a receipt is created. Since there's no check to confirm the receipt's existence, the function sets `s_lockedReceipts[receiptId]` to `true`, locking a nonexistent receipt. When the legitimate owner later creates and attempts to redeem the receipt, the transaction fails because the receipt is already marked as "locked." This prevents the owner from accessing their funds, resulting in a denial of service scenario and loss of funds.

Recommendation: Consider updating the `redeem` function ensuring only the receipt's owner can call the function by verifying `msg.sender` against `receipt.owner`. Moreover, ensure that that the receipt has been initialized (e.g., `receipt.owner` is not the zero address) before allowing the redemption process to proceed.

Khalani: Fixed in commit 62984a3.

Cantina Managed: Fix verified.

3.1.4 Lack of Access Control In `processEvent` And `produceEvent` Functions

Severity: Critical Risk

Context: MTokenCrossChainAdapter.sol#L44

Description: The `MTokenCrossChainAdapter` contract implements the `processEvent` function which is intended to handle inbound cross-chain deposit events from spoke chains and mint the corresponding `MTokens` on the hub. Specifically, when it receives an `AssetReserveDeposit` event, it locates the correct `MToken` via `MTokenRegistry` and calls `MTokenManager.mintMToken` on behalf of the depositor.

Similarly, the `produceEvent` function in the `MTokenCrossChainAdapter` contract is used to dispatch cross-chain withdrawal events. Both functions are core to the bridging workflow, ensuring that tokens deposited on a spoke chain are minted on the hub and that withdrawals are properly propagated back to the spoke.

However, neither function imposes an access control check (e.g., an `onlyEventVerifier` or similar). Because `processEvent` can trigger `mintMToken`, any malicious actor could directly call `processEvent` with fabricated parameters, minting arbitrary `MTokens` without actual deposits. Meanwhile, a malicious caller to `produceEvent` could publish fraudulent cross-chain events. In both cases, this circumvents legitimate bridging and poses a critical risk: one function inflates `MToken` supply without corresponding collateral and the other can dispatch unauthorized withdrawals messages to the spoke.

Recommendation: Enforce proper access restrictions on the `processEvent` and `produceEvent` functions, so that only a trusted contract (e.g., the `HubHandler` or another verified source of cross-chain events) can invoke them. One common approach is to add a modifier (like `onlyEventHandler`) checking `msg.sender` against a known handler contract that relays verified cross-chain messages.

Khalani: Fixed in commit 7673695.

Cantina Managed: Fix verified.

3.1.5 Incorrect Child Intent srcAmount Check in validateSpendGraph Allows Minting Extra MTokens

Severity: Critical Risk

Context: SolutionLib.sol#L107-L110

Description: The `validateSpendGraph` function currently verifies that each newly created child Intent's `srcAmount` is not less than the `qty` allocated to it from a parent Intent, but it never checks that the child's `srcAmount` matches the exact total tokens transferred to the child:

```
if (spendingIntent.srcAmount < quantitySpent) {
    revert SolutionLib__IntentAmountMismatch();
}
```

As a result, a solver can craft a leftover child receipt whose `srcAmount` surpasses the actual tokens moved from its parent Intent, effectively generating unbacked tokens. Instead of limiting the output receipt/intent `srcAmount` to the sum of its corresponding `MoveRecords`, the function only ensures `spendingIntent.srcAmount >= qty`, which is an incomplete check that opens the door to token inflation.

Proof of Concept: Initial Intents:

- User1 posts a `PctFilled` Intent locking 1000 tokens of `mToken`, wanting some ratio in `mToken2`.
- User2 also posts a `PctFilled` Intent locking 100 tokens of `mToken2`, wanting some ratio in `mToken`.

Solution Setup:

- We combine these two Intents (`intentId` and `intentId2`) in a single solution.
- Output Intents: Only one leftover child is declared for User1 with `srcAmount = 900` of `mToken`. It is labeled "PctFilled" and inherits the same ratio.
- Receipts: Two receipts are defined:
  - A receipt requesting 500 `mToken` tokens for User2 (but the `spendGraph` only allocates 100 from User1's Intent to it).
  - A second receipt with 100 `mToken2` for User1 (allocated from User2's Intent).

Spend Graph:

- MoveRecord(0): Uses 100 `mToken` from User1's parent Intent to fill `receipts[0]`.
- MoveRecord(1): Uses 100 `mToken2` from User2's parent Intent to fill `receipts[1]`.
- MoveRecord(2): Takes 900 `mToken` from User1's parent Intent and puts them into the single leftover child `outputIntents[0]`.

Where the problem occurs:

- The leftover child's declared `srcAmount = 900`, combined with the 100 allocated to the 500-token receipt, totals 1000 `mToken` from `User1`. But the receipt's definition claims 500 `mToken` even though only 100 is moved to it in the `spendGraph`. Because `validateSpendGraph` only checks `childIntent.srcAmount >= allocatedQty`, it misses that 400 are effectively unaccounted for. An attacker can exploit this mismatch to inflate tokens out of thin air by artificially raising the leftover's `srcAmount` or the receipts' amounts.

Recommendation: Require each output Intent/receipt's `srcAmount/mTokenAmount` to match precisely the total quantity allocated to it across all relevant `MoveRecords` in the same solution. If the leftover intents/receipts are supposed to hold 300 tokens, for example, ensure their total `srcAmount/mTokenAmount` is exactly 300, rather than merely not being lower than the allocated amount.

Khalani: Fixed in commit 3179365.

Cantina Managed: Fix verified.

3.1.6 `AssetReserves::produceEvent` function allows arbitrary `mToken` minting on Hub chain

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `AssetReserves::produceEvent` function allows any caller to encode arbitrary data with the deposit event without actually depositing any tokens into the contract. This can be abused to send a cross-chain message to the Hub chain and mint `mTokens` based on the amount specified in the `eventData`. The flow through functions is:

- Spoke Chain: `AssetReserves.produceEvent` → `SpokePublisher.publishEvent` → `EventProver.registerEvent`.
- Hub Chain: `EventVerifier.handle` → `HubHandler.handleEvent` → `MTokenCrossChainAdapter.processEvent` → `MTokenManager.mintMToken`.

Recommendation: Remove the `produceEvent` function or implement further changes to the system to support sending arbitrary messages to Hub chain.

Khalani: Fixed in commit a5b178d.

Cantina Managed: Fix verified.

3.1.7 Non-unique receipt ID during intent solving leads to token loss

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `IntentBook::solve` function creates receipts by calling `ReceiptManager::createReceipt` with IDs computed as `keccak256(abi.encode(owner, mToken, amount, block.number))`. When a solver creates multiple receipts in the same block for the same owner with identical `mToken` and `amount` values, it generates duplicate receipt IDs, leading to token loss since a single receipt can only be redeemed once.

Proof of Concept:

1. User1 creates two intents to swap 250 `mToken` for 500 `mToken2` each.
2. User2 creates two intents to swap 500 `mToken2` for 250 `mToken` each.
3. Solver creates a solution with four receipts:
   - Two receipts for User1: 500 `mToken2` each.
   - Two receipts for User2: 250 `mToken` each.
4. Due to identical parameters (owner, mToken, amount, block.number):
   - User1's receipts get the same ID.
   - User2's receipts get the same ID.
5. Result:
   - 1000 `mToken2` are transferred to the receipt for User1 but only 500 `mToken2` are redeemable.
   - 500 `mToken` are transferred to the receipt for User2 but only 250 `mToken` are redeemable.

Recommendation: Include a unique nonce in receipt ID calculation:

```diff
- receiptId = keccak256(abi.encode(owner, mToken, amount, block.number));
+ receiptId = keccak256(abi.encode(owner, mToken, amount, block.number, receiptNonce++));
```

Khalani: Fixed in commit c59e635.

Cantina Managed: Fix verified.

3.1.8 Child Intents in solve Do Not Receive The Parent's `mTokens`

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `finalizeSolution` function, when new child intents are created from a parent intent, the code never transfers the parent's `mTokens` balance to these newly spawned children. The parent intent should hand over its `mTokens` to allow the child intents to operate with those tokens, yet the logic omits this step.

As there is no call to the `transferMtokens(parent, child, ...)` function, each child is created with a zero `mTokens` balance. Because of this, no `mTokens` can ever be withdrawn from those intents. As the parent intent is marked as `SOLVED` and the child intents can not be used, all the remaining `mTokens` are lost.

Recommendation: Update the `finalizeSolution` function to call the `transferMtokens` function in order to send the `mTokens` from the parent intent to the newly created child intents.

Khalani: Fixed in commit 3179365.

Cantina Managed: Fix verified.

3.2 High Risk

3.2.1 `OnlyOwner` Restriction on `lockIntents` Prevents Solvers To Access `solve` Function

Severity: High Risk

Context: IntentBook.sol#L227

Description: In the `IntentBook` contract, the `solve` function should be callable by any whitelisted solver, to submit solutions for intents. However, the function includes a call to `lockIntents(consumedIntentIds)`, which is restricted by an `onlyOwner` modifier. This restriction means that only the contract owner can actually `solve` intents as `solve` internally calls the `lockIntents` function. As a result, when a non-owner attempts to call `solve`, the transaction reverts because they lack the necessary permissions to invoke `lockIntents`. This creates a contradiction: while `solve` appears intended to enable multiple external actors (solvers) to participate in submitting solutions, the `onlyOwner` restriction on `lockIntents` effectively limits this functionality to the owner alone.

Recommendation: Consider removing the `onlyOwner` modifier from the `lockIntents` function and declaring the function as `internal` instead of `public` so it can not be accessed externally.

Khalani: Fixed in commit 8a603f4.

Cantina Managed: Fix verified.

3.2.2 Intent Id Collision Risk in Partial Fill Child Intents Due to Identical Fields

Severity: High Risk

Context: IntentBook.sol#L280

Description: The `getIntentId` function calculates an Intent's ID by `keccak256(abi.encode(intent))`, relying solely on the Intent's fields. This works for most single-intent scenarios, but there is no mechanism preventing multiple separate child intents (or a child and a new parent intent) from having identical fields, particularly if the solver reuses the same `nonce` and `srcAmount` values. As a result, two distinct partial fills from different parent intents can create child intents with identical (`author`, `ttl`, `nonce`, `srcMToken`, `srcAmount`, `outcome`), leading to the same `keccak256` hash. When the second child is stored in `s_intents[intentId]`, it overwrites the first one, causing the original leftover to vanish from on-chain state and effectively losing those assets. This collision can also occur between a newly published parent intent and a leftover child intent if both share the same fields.

In normal "publishIntent" calls, the nonce is enforced to be strictly increasing for the same author, reducing collisions among parent intents. However, leftover child intents do not enforce this. If two separate partial fills produce leftover children with matching (`nonce`, `srcAmount`, `outcome`, etc.), the final `intentId` is identical, and the second solution overwrites the first in `s_intents`, `s_intentStates`, and `s_intentVersions`. This permanent overwrite results in permanent loss of funds tied to the first overwritten leftover.

Recommendation: Enforce a unique leftover nonce per parent to avoid such collisions. A child intent should never have the same nonce as its parent. Moreover, two child intents should never have the same nonce.

Khalani: Fixed in commit b14fdf8.

Cantina Managed: Fix verified.

3.2.3 Lack of Authorized Solver Check in `solve` Function Allows Untrusted Actors to Propose Solutions

Severity: High Risk

Context: IntentBook.sol#L130-L132

Description: The IntentBook contract contains a s_solvers mapping to whitelist addresses intended to act as authorized solvers. However, there is no enforcement of this whitelist in the solve function itself. Consequently, anyone can submit partial-fill solutions, circumventing the notion of a "trusted solver". Unrestricted solution submission can lead to malicious or poorly formed solutions entering the pipeline, including partial fills that lock user funds or spawn leftover child intents in ways that disadvantage the original intent authors.

Recommendation: Ensure that the solve function enforces an authorized solver requirement, such as:

```
require(s_solvers[msg.sender], "Only authorized solver can call solve");
```

Khalani: Fixed in commit 8a603f4.

Cantina Managed: Fix verified.

3.3 Medium Risk

3.3.1 Deep Intent Chains Leads to Potential DoS

Severity: Medium Risk

Context: IntentBook.sol#L316

Description: The getIntentChainRoot function traces an intent's ancestry by repeatedly following parent references from s_intentVersions[intentId] until it finds an intent with no parent. Each iteration of this loop performs a storage read (roughly 200 gas) plus additional overhead, totaling around 250–300 gas per iteration. If a deep nested chain of parent–child Intents is created, calls to getIntentChainRoot can become prohibitively expensive. In extreme scenarios (like 10,000 nested Intents), a single call might consume millions of gas. Moreover, the solve function that invokes getIntentChainRoot multiple times, especially across multiple Intents in a single transaction, can push overall gas usage to or beyond the block gas limit, preventing otherwise valid transactions from finalizing and leading into a DoS state.

Recommendation: Impose a maximum chain depth at Intent creation or partial fill leftover generation. For instance, limit each Intent's ancestry to fewer than a fixed number (e.g., 100). If an Intents chain reaches that threshold, either disallow further partial fill or require a consolidation step that merges older child Intents.

Khalani: Fixed in commit d91c7a3.

Cantina Managed: Fix verified.

3.3.2 Missing origin chain validation in EventVerifier::handle enables unauthorized cross-chain message processing

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The EventVerifier::handle function processes cross-chain messages without validating if the origin chain is a configured/trusted chain. While the need to validate the _sender address is covered in another issue, this vulnerability focuses on the missing origin chain validation.

Given the number of chains Hyperlane might support in the future, if any untrusted chain becomes compromised and an attacker gains control of an address matching a legitimate _sender, they could send unauthorized messages that would be processed by the handle function as legitimate.

Recommendation: Validate that the _origin is a trusted origin chain in the EventVerifier::handle function. The contract can maintain a mapping of whitelisted chains.

Khalani: Fixed in commit 571ba01.

Cantina Managed: Fix verified.

3.3.3 Missing event type validation in HubPublisher::publishEvent

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `HubPublisher` contract maintains a mapping of authorized event producers for specific event types in `s_producerForEventType`, but fails to validate this authorization when publishing events. It only checks if the `msg.sender` is registered as an event producer but doesn't check if the event type is authorized for the producer.

Recommendation: Add event type authorization check inside the `publishEvent` function:

```
+ if (s_producerForEventType[eventType] != msg.sender) {
+     revert HubPublisher__EventProducerNotAuthorized(msg.sender);
+ }
```

Khalani: Fixed in commit bcc326f.

Cantina Managed: Fix verified.

3.3.4 Missing event type registration in `registerEventOnProducer` leads to incomplete event tracking

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Both `HubPublisher` and `SpokePublisher` contracts fail to add event types to the `s_eventTypeRegistrations` array when registering them through the `registerEventOnProducer` function. This prevents proper tracking of producer-specific event types and breaks the `revokeProducerAccessFull` functionality, as event types cannot be correctly removed from the `s_producerForEventType` mapping.

Recommendation: Add the event type to the `s_eventTypeRegistrations` array for the producer in both contracts `registerEventOnProducer` functions.

Khalani: Fixed in commit ebd8b60.

Cantina Managed: Fix verified.

3.3.5 Missing producer removal functionality in `SpokePublisher` and `HubPublisher` prevents access revocation

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Both publisher contracts allow adding producers via `addProducer` but lack functionality to remove them from `s_eventProducers` mapping. This prevents the contract owner from revoking access from compromised or malicious producers.

Recommendation: Add producer removal functionality to both contracts.

Khalani: Fixed in commit c8236c5.

Cantina Managed: Fix verified.

3.3.6 Excess Native Assets Stuck in `EventProver` When `msg.value` Exceeds Actual Costs

Severity: Medium Risk

Context: EventProver.sol#L75-L78

Description: The `EventProver.registerEvent` function calls `mailbox.quoteDispatch` to determine the base bridging fee, pays that amount via `mailbox.dispatch{value: fee}`, then calculates a second gas payment via `interchainGasPaymaster.quoteGasPayment` and calls `interchainGasPaymaster.payForGas{value: gasPayment}()`. These calculations rely on `msg.value` forwarded from the users from the initial `AssetReserves.deposit` call, but there is no mechanism to refund any excess of native assets. If the user sends exactly `fee + gasPayment`, the system works cleanly. However, if the user overestimates costs and sends more that what they should, everything above `fee + gasPayment` remains in the `EventProver` contract's balance indefinitely.

There is no code to return the surplus to `msg.sender` or to withdraw native assets from the contract.

Recommendation: Add a final step in `registerEvent` function to refund any leftover `msg.value` after both `mailbox.dispatch` and `interchainGasPaymaster.payForGas` calls. For example, track the sum of `fee + gasPayment` and return `address(this).balance` minus that total to `msg.sender`.

Khalani: Fixed in commit 416d491.

Cantina Managed: Fix verified.

3.4 Low Risk

3.4.1 Exclusion of Smart Contract Signers Due to ECDSA Verification

Severity: Low Risk

Context: SignatureLib.sol#L16

Description: The code recovers the signer of an intent by calling `ECDSA.recover(intentHash, signedIntent.signature)`, which only supports externally owned accounts (EOAs). Smart contracts cannot generate valid ECDSA signatures, restricting any contract or multisignature wallet from interacting with the protocol. This effectively excludes a large segment of DeFi, as contract wallets and DAO-managed addresses cannot directly sign intents to automate or participate in intent-based workflows. The result is severely reduced flexibility and potential adoption, since many advanced use cases involve smart contract wallets or contract-based token managers.

Recommendation: Consider using the `SignatureChecker` library instead of `ECDSA` for signature verification. `OpenZeppelin's SignatureChecker.isValidSignatureNow` supports both EOAs and contract wallets. This approach allows both types of accounts to produce valid signatures, enabling a broader range of automated and contract-based use cases without excluding any part of the user base.

Khalani: Fixed in commit 7973625.

Cantina Managed: Fix verified.

3.4.2 Improper Declaration of `SolutionLib` as a Contract Instead of a Library

Severity: Low Risk

Context: SolutionLib.sol#L11

Description: `SolutionLib` is currently declared as a contract rather than a library, yet its intended purpose is providing utility functions for the `IntentBook` contract. Moreover, its name reflects the initial intention of creating it as a library.

Recommendation: Transform `SolutionLib` into an actual Solidity library. This eliminates external calls, lowers gas usage by inlining or link-time binding of functions, and makes it clear at the language level that the code does not hold or mutate contract storage.

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.4.3 Unvalidated Outcome Array Length in Exactly Intents

Severity: Low Risk

Context: SolutionLib.sol#L186-L192

Description: In the `SolutionLib` contract, the `_checkFilledEAS` function is designed to validate intents with the `FillStructure.Exactly`, which should involve precisely one outcome token. However, the function fails to enforce this requirement explicitly. It does not check whether the outcome arrays (`mTokens` and `mAmounts` in the `Outcome` struct) contain exactly one element. Instead, it only validates the first element of these arrays against the provided receipt, disregarding any additional elements.

Recommendation: Consider updating the `_checkFilledEAS` function in the `SolutionLib` contract to explicitly verify that the outcome arrays (`mTokens` and `mAmounts`) each contain exactly one element for `Exactly` intents. This ensures that the intent adheres to its intended strict fulfillment logic. Below is the adjusted code:

```
function _checkFilledEAS(
    Intent memory inputIntent,
    FillRecord[] memory fillersForCurrInput,
    Intent[] memory, /*createdIntents*/
    Receipt[] memory createdReceipts
) internal {
    if (inputIntent.outcome.mTokens.length != 1) {
        revert("Exactly intents must have exactly one outcome token");
    }
    if (inputIntent.outcome.mAmounts.length != 1) {
        revert("Exactly intents must have exactly one outcome amount");
    }
    if (fillersForCurrInput.length != 1) {
        revert SolutionLib__InvalidFillGraphForEASIntent();
    }
    FillRecord memory currFiller = fillersForCurrInput[0];
    if (currFiller.outType == OutType.Intent) {
        revert SolutionLib__ExactAnySingleMustBeFulfilledWithReceipts(currFiller);
    }
    Receipt memory fillerReceipt = createdReceipts[currFiller.outIdx];
    if (inputIntent.outcome.mTokens[0] != fillerReceipt.mToken) {
        revert SolutionLib__InputOutputTokenTypeMismatch(inputIntent.outcome.mTokens[0], fillerReceipt.mToken);
    }
    if (inputIntent.outcome.mAmounts[0] != fillerReceipt.mTokenAmount) {
        revert SolutionLib__IntentAmountMismatch();
    }
    if (inputIntent.author != fillerReceipt.owner) {
        revert SolutionLib__MismatchBetweenInputAndOutputOwners();
    }
}
```

Khalani: Fixed in 3ab05a5.

Cantina Managed: Fix verified.

3.4.4 Impractical Matching for `PctFilled` Intents with Non-Integer Divisible `mAmounts`

Severity: Low Risk

Context: SolutionLib.sol#L297-L300

Description: In the `SolutionLib` contract, the `_checkFilledPAS` function validates `PctFilled` intents by calculating the expected receipt amount based on the proportion of the source tokens consumed, using the `mAmounts` field as a percentage scaled by `10^18`. While this works well for integer-based ratios like `100e18 (1:1)` or `50e18 (1:2)`, it becomes problematic when users specify a finely granular ratio, such as 1_123456789012345678. For an intent offering 100 `mToken` to receive `mToken2` at this rate, the expected outcome is roughly 112.345678901234567800 `mToken2` for a full fill, or 11.234567890123456780 `mToken2` for a 10 `mToken` partial fill.

However, this granularity introduces challenges in Solidity, which relies on integer arithmetic and does not natively support fractional numbers. The precision required to compute these amounts can lead to rounding errors or fractional outcomes that can not be accurately represented. Additionally, for this intent to be matched, an opposing intent must offer `mToken2` and expect `mToken` at the precise inverse rate (approximately `0.887654321098765432`). Finding such a perfectly aligned opposing intent becomes impractical due to the fine-grained nature of the ratio, potentially leaving the intent unmatchable.

Recommendation: To mitigate these issues, it's advisable to impose constraints on the `mAmounts` field for `PctFilled` intents. By limiting the granularity of the ratios, the system can ensure that calculated token amounts remain practical and compatible with the token's precision. One effective approach is to require that `mAmounts` be a multiple of a coarser base unit, such as `10^13` or `10^14`. For instance, adding a check like this in the `publishIntent` function of the `IntentBook` contract could help:

```
require(mAmounts % 10^13 == 0, "mAmounts must yield manageable outcomes");
```

Khalani: Fixed in commit e3cb212.

Cantina Managed: Fix verified.

3.4.5 Hardcoded `msg.sender` as `mToken` recipient lacks support for smart contract wallets

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `AssetReserves::deposit` function sends a cross-chain message to mint mTokens to `msg.sender` on the destination chain without allowing specification of a different recipient address. This creates issues for smart contract wallets that may not have the same address across different chains, potentially leading to permanently inaccessible tokens.

```
function deposit(
    address token,
    uint256 amount,
    uint32 destChain,
    uint256 permitNonce,
    uint256 deadline,
    bytes calldata signature
) external payable {
    // ... other code ...
    AssetReserveDeposit memory depositEvent = AssetReserveDeposit(token, scaledAmount, msg.sender);
    // msg.sender is hardcoded as recipient
}
```

Recommendation: Add recipient parameter to deposit function:

```
  function deposit(
      address token,
      uint256 amount,
      uint32 destChain,
+     address recipient,
      uint256 permitNonce,
      uint256 deadline,
      bytes calldata signature
  ) external payable {
-     AssetReserveDeposit memory depositEvent = AssetReserveDeposit(token, scaledAmount, msg.sender);
+     AssetReserveDeposit memory depositEvent = AssetReserveDeposit(token, scaledAmount, recipient);
  }
```

Khalani: Fixed in commit a635c78.

Cantina Managed: Fix verified.

3.4.6 Dispatching the message through Hyperlane pays for the default gas on top of configured gas

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `EventProver` contract first uses the `Mailbox::dispatch()` function with default hooks to send the message through Hyperlane and then pays for additional gas through the `InterchainGasPaymaster`. The dispatch function uses default hooks and it will call the `InterchainGasPaymaster` and pay for the default gas which is 50k. Additional gas provided in the `.payForGas` call is added on top of the default gas.

Recommendation: Consider using the `Mailbox::dispatch()` function overload where you can specify the gas amount.

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.4.7 Missing `chainId` in `DOMAIN_SEPARATOR` leads to cross-chain signature replay

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `DOMAIN_SEPARATOR` calculation in IntentLib does not include the `chainId` parameter as required by EIP-712 specification. This omission enables signature replay attacks across different chains where the contract is deployed, allowing to reuse signatures on other chains.

```
bytes32 DOMAIN_SEPARATOR = keccak256(
    abi.encode(
        keccak256("EIP712Domain(string name,string version,address verifyingContract)"),
        keccak256(bytes(name)),
        keccak256(bytes(version)),
        address(this)
    )
);
```

Recommendation:

1. Add `chainId` to DOMAIN_SEPARATOR as per EIP-712:

```
  bytes32 DOMAIN_SEPARATOR = keccak256(
      abi.encode(
-         keccak256("EIP712Domain(string name,string version,address verifyingContract)"),
+         keccak256("EIP712Domain(string name,string version,uint256 chainId,address
↪   verifyingContract)"),
          keccak256(bytes(name)),
          keccak256(bytes(version)),
          block.chainid,
          address(this)
      )
  );
```

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.4.8 Private functions that are not used anywhere

Severity: Low Risk

Context: *(See the cases below)*

Description: These are private functions that are not used:

- SpokePublisher.sol#L139.
- SpokePublisher.sol#L132.
- HubPublisher.sol#L107.
- HubPublisher.sol#L124.
- HubPublisher.sol#L114.

Recommendation: Consider the usage of these functions within the `SpokePublisher` and `HubPublisher` contracts and use them accordingly. Otherwise, if they aren't intended to be used, remove them.

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.4.9 General Code Improvements

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description:

- spokeHandler storage variable is set in the constructor but can't be changed. Add a function to change the address of the `spokeHandler`.
- `EventVerifier` has a function `verifyEvent` that returns the boolean value indicating if a specific event hash has been verified. Rename the function to `isEventVerified` and declare it as view.

- The `IntentBook::validateSolutionInputs` function that gets called during the `IntentBook::solve` function execution doesn't check if passed intents have duplicates. There is a revert in the `lockIntents` function that gets called a few lines below so this prevents double spending the same intent. Regardless, it would be good to have a sorted intent array passed and sorting checked in the `validateSolutionInputs` function to prevent any duplicates.
- Rename `AssetRegistry__InavlidAssetId()` to `AssetRegistry__InvalidAssetId()`.
- `registerEventProver` does not verify that `eventProver` is a valid contract or implements an expected interface (e.g., IEventProver). Consider verifying this.
- In the `emit AssetWithdrawn(token, to, amount)` event present in the `AssetReserves._withdraw` function, the `amount` logged is the pre-escaled amount, not the actual amount of tokens that will be received by the user. Consider updating the event to log the amount transferred instead.
- `checkIntentValidToSpend` calls `verifyIntentSignature`. This is not necessary as this verification was already done in the `IntentBook.validateSolutionInputs` function.
- `IntentLib.hashIntent` and `IntentLib.hashIntentWithEip712` functions can be declared as `pure` instead of `view`.
- The `IntentLib.hashStructOfIntent`, `IntentLib.alternativeEncode`, `IntentLib.eip712AbiEncodedData`, `IntentLib.hashOutcomeAlternate`, `IntentLib.outcomeTypeHash` and `IntentLib.encodedOutcome` functions can be removed as they are not used anywhere in the codebase.
- `s_receiptsByOwner[owner]` owner array will keep growing indefinitely. Consider removing the `receiptId` from the `s_receiptsByOwner[owner]` array every time `redeem` is called.
- In the `EventVerifier` contract, `i_eventHandler` was declared as immutable and therefore it can not be updated after deployment, restricting the protocol's ability to adapt to future upgrades or replacements of the `HubHandler` contract. If the HubHandler requires any update, such as fixing security vulnerabilities, optimizing event processing logic or integrating new features, the `EventVerifier` cannot redirect events to an updated handler without a complete redeployment. Consider adding a setter to the `i_eventHandler` state variable removing its immutability.

Recommendation: Implement the recommendations outlined above.

Khalani: Fixed in commit 1e887b7.

Cantina Managed: Fix verified.

3.4.10 Malicious Partial Fills Causing Dust Intent Fragmentation

Severity: Low Risk

Context: IntentBook.sol#L236

Description: In the `IntentBook` contract, the `solve` function calls the `SolutionLib.checkIntentSatisfaction` to process solutions, allowing solvers to perform partial fills on intents. This design allows a malicious solver to repeatedly execute partial fills in small increments on a single intent, generating a series of leftover child intents, each containing only a negligible "dust" amount of tokens. Since each partial fill produces exactly one new child intent, an attacker willing to pay the gas costs can progressively fragment the original intent into many tiny, dust-sized pieces. These resulting child intents hold such small token amounts that they become economically unviable for other solvers to fill, or for the publisher to claim, given the transaction fees involved, and. As a result, the user is left with a collection of nearly unfillable intents.

Recommendation: Consider updating the `solve` function or the `SolutionLib.checkIntentSatisfaction` logic to impose restrictions on partial fills. One approach is to enforce a minimum threshold for the `srcAmount` of any resulting child intent, rejecting solutions that would create a leftover intent below this limit, such as 1% of the original intent's `srcAmount`.

Khalani: Fixed in commit 1852399.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Unused imports, errors and variables

Severity: Informational

Context: *(See the cases below)*

Description: Here are the unused imports:

- HubHandler.sol#L4.
- AssetReserves.sol#L8.
- AssetReserves.sol#L7.
- SpokeHandler.sol#L4.
- EventProver.sol#L9.
- MTokenCrossChainAdapter.sol#L9.
- SolutionLib.sol#L9.

And here are the unused errors:

- MTokenManager.sol#L32.
- MTokenManager.sol#L30.
- EthAipRedeemer.sol#L13.
- EthAipRedeemer.sol#L14.

Unused storage variables:

- SpokePublisher.sol#L37.
- HubPublisher.sol#L38.

Recommendation: Remove unused imports, errors and storage variables.

Khalani: Fixed in commit da0fdc9.

Cantina Managed: Fix verified.

3.5.2 Lack of a Double-Step Transfer Ownership Pattern

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The standard OpenZeppelin's Ownable contract allows transferring the ownership of the contract in a single step:

```
/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public virtual onlyOwner {
    if (newOwner == address(0)) {
        revert OwnableInvalidOwner(address(0));
    }
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

If the nominated EOA account is not a valid account, it is entirely possible that the owner may accidentally transfer ownership to an uncontrolled account, losing the access to all functions with the `onlyOwner` modifier.

Recommendation: It is recommended to implement a two-step transfer process in all the contracts in the codebase where the owner nominates an account and the nominated account needs to call an `acceptOwnership()` function for the transfer of the ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. A good code example could be OpenZeppelin's Ownable2Step contract:

```
/**
 * @dev Starts the ownership transfer of the contract to a new account. Replaces the pending transfer if there
 ↪   is one.
 * Can only be called by the current owner.
 *
 * Setting `newOwner` to the zero address is allowed; this can be used to cancel an initiated ownership
 ↪   transfer.
 */
function transferOwnership(address newOwner) public virtual override onlyOwner {
    _pendingOwner = newOwner;
    emit OwnershipTransferStarted(owner(), newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`) and deletes any pending owner.
 * Internal function without access restriction.
 */
function _transferOwnership(address newOwner) internal virtual override {
    delete _pendingOwner;
    super._transferOwnership(newOwner);
}

/**
 * @dev The new owner accepts the ownership transfer.
 */
function acceptOwnership() public virtual {
    address sender = _msgSender();
    if (pendingOwner() != sender) {
        revert OwnableUnauthorizedAccount(sender);
    }
    _transferOwnership(sender);
}
```

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.5.3 Lack Of Support for Fee-on-Transfer Tokens

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The smart contract's current design always assumes that the entire specified token amounts are transferred and received without any reductions. However, fee-on-transfer tokens automatically deduct a percentage (such as 2% or 5%) during the transfer process, resulting in the recipient receiving less than the intended amount. For example, transferring 100 tokens with a 2% fee means the recipient gets only 98 tokens, with the remaining 2 tokens redirected elsewhere. This fee mechanism, built into the token's `transfer` or `transferFrom` functions, causes a mismatch between the expected and actual amounts received by the contract. Since the contract does not adjust for these deductions, using them would lead to a total broken accounting.

Recommendation: Avoid whitelisting any fee-on-transfer token.

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.5.4 Limited Support for Fill Structures and Outcome Asset Structures in Intent System

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The intent system, as defined in the `Intent.sol` file, includes an enumeration called `FillStructure` with options `Exactly`, `Minimum`, `PctFilled` and `ConcreteRange`. However, only `Exactly` and `PctFilled` are currently supported in the current implementation. Similarly, for outcome asset structures, only `AnySingle` is functional, while other potential options remain unimplemented.

Recommendation: Merely informative. Consider updating the documentation within the codebase to explicitly state that only `Exactly` and `PctFilled` are supported for fill structures and `AnySingle` for outcome asset structures.

Khalani: Acknowledged.

Cantina Managed: Acknowledged.

3.5.5 Insufficient Validation in `mTokenApprove` Function

Severity: Informational

Context: MTokenManager.sol#L256

Description: The `mTokenApprove` function is intended to facilitate the approval of a specific amount of tokens, allowing a designated spender to utilize them on behalf of an owner for a given `mToken`. However, this function does not call the `_validateMToken` function as its done in the rest of the contract's function.

Recommendation: Update the `mTokenApprove` function by adding the following line to validate that `mToken` passed as parameter is supported, not paused and not destroyed:

```
_validateMToken(mToken);
```

Khalani: Fixed in commit c60427c.

Cantina Managed: Fix verified.