Zellic

Catalyst

Smart Contract Security Assessment

June 19, 2023

Prepared for:

Jim Chang

Cata Labs, Inc

Prepared by:

Syed Faraz Abrar and Filippo Cremonese

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for Cata Labs, Inc from May 30th to June 9th, 2023. During this engagement, Zellic reviewed Catalyst's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could an on-chain attacker circumvent the slippage logic to front-run other swaps?
- Is the math behind the liquidity management and swap equations sound?
- Could an on-chain attacker steal vault funds in any way?

1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

The math on which the vault contracts are designed is substantially complex; the relatively short audit period prevented us from being fully confident with our understanding of the math-related code.

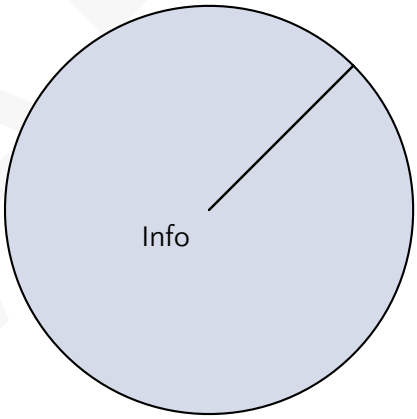
1.3 Results

During our assessment on the scoped Catalyst contracts, we discovered two findings, all of which were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for Cata Labs, Inc's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	2



2 Introduction

2.1 About Catalyst

Cata Labs, Inc are contributors to Catalyst, an open-source liquidity protocol for modular blockchains. Any new chain can use Catalyst to permissionlessly connect liquidity and swap with hubs like Ethereum and Cosmos.

2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

Catalyst Contracts

Repository	https://github.com/catalystdao/catalyst/tree/main/evm
Version	catalyst: `504b15b4924e9891af46ff96edaaf6be147bf11a`
Programs	• CatalystVaultCommon • CatalystVaultVolatile • CatalystVaultAmplified • CatalystIBCInterface • CatalystFactory
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of four person-weeks. The assessment was conducted over the course of two

calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Syed Faraz Abrar, Engineer
faith@zellic.io

Filippo Cremonese, Engineer
fcremo@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

May 30, 2023	Kick-off call
May 30, 2023	Start of primary review period
June 9, 2023	End of primary review period
TBD	Closing call

3 Detailed Findings

3.1 The IBC interface on Ethereum is not finalized

- **Target**: CatalystIBCInterface
- **Category**: Code Maturity
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The CatalystIBCInterface contract is used to achieve inter-blockchain communication (IBC) between vaults across chains. This IBC protocol is still a work in progress and thus not yet standardized. Due to this reason, there is also an IBCEmulator contract that contains more low-level IBC-related functionality that is currently being used to emulate sending IBC packets in the testing environment.

Impact

Since the IBC protocol is not standardized, we have had to make a few assumptions that we believe should be revisited after the protocol standard is finalized. We have assumed the following:

1. Only IBC interface contracts (such as CatalystIBCInterface) are able to send messages to other IBC interface contracts, meaning that crafted IBC packets from random addresses would not be accepted.
2. There are no ways to spoof the sender of the IBC messages. This is important because the current CatalystIBCInterface code does not validate that the sender is another CatalystIBCInterface contract.

Recommendations

Consider revisiting the assumptions made around IBC after the standard has been finalized. If the core principles behind the standard change at that point in time, it is likely that the code in CatalystIBCInterface (as well as any other IBC-related code) will need to be updated to ensure there are any security issues.

Remediation

The development team is aware of the potential issues, and has acknowledged this finding:

> 3.1 is something we are very aware of. In the future, we want Catalyst to support multiple messaging routers. In these situations, it is important we update the implementation to handle the security assumptions specific to each messaging router.

3.2 localSwap allows to use any input asset

- **Target**: CatalystVaultVolatile
- **Category**: Business Logic
- **Likelihood**: Low
- **Severity**: Informational
- **Impact**: Informational

Description

The `CatalystVaultVolatile::localSwap` function allows to use any asset as input, even if it is not part of the vault.

The amount of output tokens is computed by `calcLocalSwap`, which does not check that the input and output assets appear in the `_weight` map. Therefore, the weight of an incorrect asset will be treated as zero, causing zero output tokens instead of a revert.

Impact

This issue does not damage the vault contract or its existing users; it can only cause a loss for users that invoke the vault contracts with an erroneous input token address. We also note that approval must be granted to the vault to enable it to call `transferFrom`. We consider this improbable but not impossible and thus classify this issue as informational.

Recommendations

Consider reverting if the input or output assets are not part of the vault assets. This can be done by ensuring the values read from the `_weight` map are not zero.

Remediation

The development team has evaluated the tradeoff between increased gas cost and potential impact for the contract, and decided to accept the current behavior.

> While the change would only have a very modest impact on the gas cost: 1000 gas / $0,04, we don't believe the risk is significant enough to warrant this change. First of all, this would have to be done manually. Any automated system would detect the wrong output after the transaction simulation and not execute the transaction. Furthermore, by setting the minimum output to not 0, the swap should revert rather than execute. Any routing system can also set a minimum output separately from Catalyst's logic by checking its balance after the swap. This would also completly migrate [sic] the problem.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Adjustments of weight and amplification over time

Currently, Catalyst has two vault types — Volatile and Amplified. The main difference between the two types is in how the vault assets are priced.

- Volatile vaults use an asset-specific pricing curve with the equation `W/w`, where `W` is an asset-specific weight, and `w` is that same asset's vault balance.
- Amplified vaults use an asset-specific pricing curve with the equation `(1 / pow (w, theta)) * (1 - theta)`, where `theta` is the vault's amplification value, and `w` is a specific asset's balance.

Due to the way the assets are priced, it is extremely important that any adjustments to asset-specific weights and vault amplification values are performed gradually over a period of time. This is because any adjustments made will immediately introduce an arbitrage opportunity, which can be taken advantage through MEV. Minimizing the amount of arbitrage possible is crucial in this case.

The Catalyst team is already aware of this issue. They have implemented the following safety boundaries to protect liquidity providers:

1. Asset-specific weights and vault amplification values have a minimum adjustment time of seven days and a maximum adjustment time of 365 days.
2. For amplified vaults, the new amplification value cannot be greater than or less than the current value by more than a factor of 2.
3. For volatile vaults, new asset-specific weights values cannot be greater than or less than the current values by more than a factor of 10.

We believe that the above safety boundaries are adequate in this case, but we deem it necessary to mention it here as it is very important that these boundaries do not get removed or modified without prior consideration and planning.

4.2 Dependency on bots to minimize impact of arbitrage opportunities

Currently, Catalyst is working off of two key assumptions that prevent exploitation of the system as a whole:

1. Each vault will have a sufficiently large amount of liquidity of each asset.
2. Automated bots (deployed by the Catalyst team, or by any other actors) are going to rebalance vault asset balances automatically whenever there is a small arbitrage opportunity.

Both of these assumptions together ensure that any two vaults that hold the same assets will have very similar amounts of liquidity of each asset within them. It is extremely important that the team ensures that these assumptions hold, particularly the second one.

To demonstrate why this is important, let us assume that there are two vaults with very low liquidity and that there are no automated bots that are taking advantage of arbitrage opportunities to rebalance the vaults.

Due to normal usage of the vaults, and due to the fact that automated bots are not rebalancing the vaults, it is very possible for the vaults to end up in the following state:

1. Vault A — $100 worth of assets
2. Vault B — $10,000 worth of assets

When this state is reached, it is possible for a user to deposit $100 into Vault A and gain vault tokens that represent 50% of a vault's assets.

They would then be able to transfer these tokens over to Vault B (using `sendLiquidity()`) and swap them out of Vault B. This would give them 50% of Vault B's assets, in other words $5,000. That is a 5,000% profit.

Because of the above scenario, we deem it necessary to mention that the Catalyst team should always monitor their vaults to ensure that they not only have high amounts of liquidity but are also being rebalanced when needed by automated arbitrage bots.

5 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1 Module: CatalystFactory.sol

Function: `deployVault(address vaultTemplate, address[] assets, uint256[] init_balances, uint256[] weights, uint256 amp, uint256 vaultFee, string name, string symbol, address chainInterface)`

Deploys and initializes a new vault contract by creating an immutable proxy that invokes the provided implementation address (`vaultTemplate`).

The new contract is also initialized using the specified parameters.

Inputs

- `vaultTemplate`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the implementation address invoked by the proxy.
- `assets`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the addresses of the assets of the vault.
- `init_balances`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from length matching `assets`).
  - **Impact**: Specifies the initial balances of every asset of the vault.
- `weights`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from length matching `assets`).
  - **Impact**: Specifies the weights of the vault assets.

- `amp`
  - **Control**: Arbitrary.
  - **Constraints**: None directly, vault implementation might restrict this.
  - **Impact**: Specifies the amplification parameter of the vault.
- `vaultFee`
  - **Control**: Arbitrary.
  - **Constraints**: None directly, intended vault implementation limits it to 100%.
  - **Impact**: Specifies the vault fees.
- `name`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the name of the vault.
- `symbol`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the vault short name.
- `chainInterface`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the address of the CatalystIBCInterface contract.

Branches and code coverage (including function calls)

Intended branches

- Deploys the minimal vault proxy and performs vault setup.
  ☑ Test coverage

Negative behavior

- Reverts if no assets are specified.
  ☑ Negative test
- Reverts if length of `assets`, `init_balances`, and `weights` do not match.
  ☑ Negative test

Function call analysis

- `rootFunction → ERC20(assets[it]).safeTransferFrom(msg.sender, vault, init_balances[it])`
  - **What is controllable?** Amount parameter and asset address.
  - **If return value controllable, how is it used and how can it go wrong?** Not

used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is not a concern.
- `rootFunction → ICatalystV1Vault(vault).setup( ... )`
  - **What is controllable?** `vault`, `name`, `symbol`, `chainInterface`, and `vaultFee`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is not a concern.
- `rootFunction → ICatalystV1Vault(vault).initializeSwapCurves(assets, weights, amp, msg.sender)`
  - **What is controllable?** `vault`, `assets`, `weights`, and `amp`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is not a concern.

5.2 Module: CatalystIBCInterface.sol

Function: `sendCrossChainAsset(byte[32] channelId, byte[] toVault, byte[] toAccount, uint8 toAssetIndex, uint256 U, uint256 minOut, uint256 fromAmount, address fromAsset, byte[] calldata_)`

This function packs the input cross-chain vault swap–related data and calls the IbcDispatcher's `sendIbcPacket()` function with it.

Inputs

- `channelId`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toVault`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This is used to determine the target vault.
- `toAccount`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.

- `toAssetIndex`
  - **Control**: Fully controlled.
  - **Constraints**: Must be a valid asset index in the target vault.
  - **Impact**: This is used to determine the token that the caller is swapping to in the target vault.
- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used to calculate the actual number of tokens the caller is purchasing in the target vault.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.
- `fromAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: If the cross-chain swap fails, this is the amount returned back to the user.
- `fromAsset`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be a token in the vault.
  - **Impact**: This is the token being swapped.
- `calldata_`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Optional, used to make a call on the target chain if applicable.

Branches and code coverage (including function calls)

Intended branches

- Successfully calls `sendIbcPacket()` when given valid arguments (specifically `toVault` and `toAccount`).
  - ☐ Test coverage

Negative behavior

- Invalid `toVault` and `toAccount` triggers revert.
  - ☐ Negative test

Function call analysis

- `sendCrossChainAsset → IbcDispatcher(IBC_DISPATCHER).sendIbcPacket(channelId, payload, timeoutBlockHeight)`
  - **What is controllable?** `channelId` and `payload` are both controlled.
  - **If return value controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Cannot revert or reenter.

Function: `sendCrossChainLiquidity(byte[32] channelId, byte[] toVault, byte[] toAccount, uint256 U, uint256[Literal(value=2, unit=None)] minOut, uint256 fromAmount, byte[] calldata_)`

This function packs the input cross-chain liquidity swap–related data and calls the IbcDispatcher's `sendIbcPacket()` function with it.

Inputs

- `channelId`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toVault`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This is used to determine the target vault.
- `toAccount`
  - **Control**: Fully controlled by user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the target vault tokens being purchased.
- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used to calculate the actual number of tokens the caller is purchasing in the target vault.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.

- `fromAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: If the cross-chain swap fails, this is the amount of escrowed funds returned back to the user.
- `calldata_`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Optional, used to make a call on the target chain if applicable.

Branches and code coverage (including function calls)

Intended branches

- Successfully calls `sendIbcPacket()` when given valid arguments (specifically `toVault` and `toAccount`)
  - ☐ Test coverage
- Unchecked
  - ☐ Test coverage

Negative behavior

- Invalid `toVault` and `toAccount` triggers revert
  - ☐ Negative test

Function call analysis

- `sendCrossChainLiquidity` → `IbcDispatcher(IBC_DISPATCHER).sendIbcPacket(channelId, payload, timeoutBlockHeight)`
  - **What is controllable?** `channelId` and `payload` are both controlled.
  - **If return value controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Cannot revert or reenter.

5.3 Module: CatalystVaultAmplified.sol

Function: `depositMixed(uint256[] tokenAmounts, uint256 minOut)`

This function is used by users to deposit tokens to the vault.

Inputs

- `tokenAmounts`
  - **Control**: Fully controlled.
  - **Constraints**: User must own at least each amount of tokens.
  - **Impact**: The amounts of tokens are transferred from the user to this contract.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.

Branches and code coverage (including function calls)

Intended branches

- Correct amount of vault tokens are minted to the user.
  ☑ Test coverage

Negative behavior

- Slippage protection causes revert correctly when `vaultTokens < minOut`.
  ☐ Negative test
- Function reverts when `intU` is less than 0 after calculation of `walpha_0`.
  ☐ Negative test

Function: `localSwap(address fromAsset, address toAsset, uint256 amount, uint256 minOut)`

This function is used by users to perform a swap between two tokens in the vault.

Inputs

- `fromAsset`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be a token in the vault, otherwise `calcLocalSwap()` will revert.
  - **Impact**: This is the token being swapped from.
- `toAsset`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be a token in the vault, otherwise `calcLocalSwap()` will revert.
  - **Impact**: This is the token being swapped to.

- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: User must own at least this amount of `fromAsset` tokens.
  - **Impact**: This is the amount of `fromAsset` tokens swapped to `toAsset` tokens.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.

Branches and code coverage (including function calls)

Intended branches

- Local swap works and returns the correct amount of `toAsset` tokens to the user.
  ☑ Test coverage
- Special case codepath in `calcLocalSwap` for assets of equal weight
  ☑ Test coverage

Negative behavior

- `minOut` successfully protects the user in case of bad slippage by reverting.
  ☑ Negative test

Function: `onSendAssetFailure(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, address escrowToken, uint32 blockNumberMod)`

This function releases escrowed tokens and sends them back to the original fallback address specified by the user. It also updates the security limit accordingly.

Inputs

- `channelId`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toAccount`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.
- `U`
  - **Control**: Partially controlled.

  - **Constraints**: N/A.
  - **Impact**: This is used to calculate the actual number of tokens the caller is purchasing in the target vault.
- `escrowAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the number of tokens that are escrowed.
- `escrowToken`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be a token in the sender vault.
  - **Impact**: This is the token being swapped.
- `blockNumberMod`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the block number that the swap occurred on in the sender chain.

Branches and code coverage (including function calls)

Intended branches

- The escrowed tokens are released back to the user, and the security limit is updated correctly.
  ☑ Test coverage

Negative behavior

- Reverts if not called by the IBC interface contract.
  ☑ Negative test

Function: `onSendAssetSuccess(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, address escrowToken, uint32 blockNumberMod)`

This function releases escrowed tokens and updates the security limit of the vault.

Inputs

- `channelId`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.

- `toAccount`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.
- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used to calculate the actual number of tokens the caller is purchasing in the target vault.
- `escrowAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the number of tokens that are escrowed.
- `fromAsset`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be a token in the sender vault.
  - **Impact**: This is the token being swapped.
- `blockNumberMod`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the block number that the swap occurred on in the sender chain.

Branches and code coverage (including function calls)

Intended branches

- The escrowed tokens are released and the security limit is updated correctly.
  ☑ Test coverage

Negative behavior

- Reverts if not called by the IBC interface contract.
  ☑ Negative test

Function: `onSendLiquidityFailure(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, uint32 blockNumberMod)`

This function releases escrowed tokens and sends them back to the original fallback address specified by the user. It also updates the security limit accordingly.

Inputs

- `channelId`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toAccount`
  - **Control**: Fully controlled by cross-chain user.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.
- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used to calculate the actual number of tokens the caller is purchasing in the target vault.
- `escrowAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the number of tokens that are escrowed.
- `blockNumberMod`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the block number that the swap occurred on in the sender chain.

Branches and code coverage (including function calls)

Intended branches

- The escrowed tokens are released back to the user, and the security limit is updated correctly.
  ☑ Test coverage

Negative behavior

- Reverts if not called by the IBC interface contract.
  ☑ Negative test

Function: `receiveAsset(byte[32] channelId, byte[] fromVault, uint256 toAssetIndex, address toAccount, uint256 U, uint256 minOut, uint256 fromAmount, byte[] fromAsset, uint32 blockNumberMod)`

This function is called by the IBC interface to complete a cross-chain swap.

Inputs

- `channelId`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the channel ID of the source chain.
- `fromVault`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the address of the source vault.
- `toAssetIndex`
  - **Control**: Fully controlled by the cross-chain user.
  - **Constraints**: Must be a valid index of a token in this vault.
  - **Impact**: This is used to determine the token the user receives after the swap.
- `toAccount`
  - **Control**: Fully controlled by the cross-chain user.
  - **Constraints**: N/A.
  - **Impact**: This is the address the tokens are sent to after the swap.
- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the amount of units being swapped.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.
- `fromAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the amount of the tokens from the sender chain being swapped, minus fees.
- `fromAsset`
  - **Control**: Fully controlled.

  - **Constraints**: N/A.
  - **Impact**: This is only used to emit the event.
- `blockNumberMod`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the block number that the swap occurred on in the sender chain.

Branches and code coverage (including function calls)

Intended branches

- Cross-chain swap is completed successfully.
  ☑ Test coverage

Negative behavior

- Reverts if not called by the IBC interface contract.
  ☑ Negative test
- `minOut` successfully protects the user in case of bad slippage by reverting.
  ☑ Negative test

Function: `receiveLiquidity(byte[32] channelId, byte[] fromVault, address toAccount, uint256 U, uint256 minVaultTokens, uint256 minReferenceAsset, uint256 fromAmount, uint32 blockNumberMod)`

This function is called by the IBC interface to complete a cross-chain liquidity swap.

Inputs

- `channelId`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the channel ID of the source chain.
- `fromVault`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the address of the source vault.
- `toAccount`
  - **Control**: Fully controlled by the cross-chain user.
  - **Constraints**: N/A.
  - **Impact**: This is the address the tokens are sent to after the swap.

- `U`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the amount of units being swapped.
- `minVaultTokens`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection to ensure the user receives the correct amount of vault tokens.
- `minReferenceAsset`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is also used for slippage protection but to ensure that the value of the vault tokens is at least this amount when comparing against the reference asset being used to price the vault tokens.
- `fromAmount`
  - **Control**: Partially controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the amount of vault tokens from the sender chain being swapped, minus fees.
- `blockNumberMod`
  - **Control**: Not controlled.
  - **Constraints**: N/A.
  - **Impact**: This is the block number that the swap occurred on in the sender chain.

Branches and code coverage (including function calls)

Intended branches

- Cross-chain liquidity swap is completed successfully.
  ☑ Test coverage

Negative behavior

- Reverts if not called by the IBC interface contract.
  ☑ Negative test
- `minVaultTokens` and `minReferenceAsset` successfully protects the user in case of bad slippage by reverting.
  ☑ Negative test

Function: `sendAsset(byte[32] channelId, byte[] toVault, byte[] toAccount, address fromAsset, uint8 toAssetIndex, uint256 amount, uint256 minOut, address fallbackUser, byte[] calldata_)`

This function is used to initiate a cross-chain swap.

Inputs

- `channelId`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toVault`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This is used to determine the target vault.
- `toAccount`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.
- `fromAsset`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be a token in the vault.
  - **Impact**: This is the token being swapped.
- `toAssetIndex`
  - **Control**: Fully controlled.
  - **Constraints**: Must be a valid asset index in the target vault.
  - **Impact**: This is used to determine the token that the caller is swapping to in the target vault.
- `amount`
  - **Control**: Fully controlled.
  - **Constraints**: Caller must own at least this amount of `fromAsset` tokens.
  - **Impact**: This amount is transferred from the caller to this vault.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.
- `fallbackUser`
  - **Control**: Fully controlled.
  - **Constraints**: `fallbackUser` $\neq$ `address(0)`.

  - **Impact**: If the cross-chain swap fails, the caller's funds are sent to this address.
- `calldata_`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Optional, used to make a call on the target chain if applicable.

Branches and code coverage (including function calls)

Intended branches

- Asset swap works correctly with correct arguments.
  ☑ Test coverage

Negative behavior

- Escrow already existing causes revert.
  ☐ Negative test
- Bad `fromAsset` causes revert.
  ☑ Negative test
- Bad return value from `calcSendAsset()` causes revert.
  ☐ Negative test

Function call analysis

- `sendAsset` → `CatalystIBCInterface(_chainInterface).sendCrossChainAsset(...)`
  - **What is controllable?** All function arguments are controllable.
  - **If return value controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, the entire call reverts. No reentrancy possible.
- `sendAsset` → `ERC20(fromAsset).safeTransferFrom(msg.sender, address(this), amount)`
  - **What is controllable?** `fromAsset` and `amount` are controllable.
  - **If return value controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, the entire call reverts. No reentrancy possible.

Function: `sendLiquidity(byte[32] channelId, byte[] toVault, byte[] toAccount, uint256 vaultTokens, uint256[Literal(value=2, unit=None)] minOut, address fallbackUser, byte[] calldata_)`

This function is used to initiate a cross-chain liquidity swap.

Inputs

- `channelId`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be a valid ID for a chain.
  - **Impact**: This is used to determine the target chain.
- `toVault`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This is used to determine the target vault.
- `toAccount`
  - **Control**: Fully controlled by caller.
  - **Constraints**: Must be 65 bytes correctly encoded.
  - **Impact**: This account receives the funds being transferred.
- `vaultTokens`
  - **Control**: Fully controlled.
  - **Constraints**: Caller must own at least this amount of vault tokens.
  - **Impact**: This amount of vault tokens is burned from the caller.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.
- `fallbackUser`
  - **Control**: Fully controlled.
  - **Constraints**: `fallbackUser ≠ address(0)`.
  - **Impact**: If the cross-chain swap fails, the escrowed funds are sent to this address.
- `calldata_`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: Optional, used to make a call on the target chain if applicable.

Branches and code coverage (including function calls)

Intended branches

- Liquidity swap works correctly with valid arguments.
  ☑ Test coverage

Negative behavior

- Overflow on calculated `U` causes revert.
  ☐ Negative test
- Escrow already existing causes revert.
  ☐ Negative test

Function call analysis

- `sendAsset` → `CatalystIBCInterface(_chainInterface).sendCrossChainLiquidity( ... )`
  - **What is controllable?** All function arguments are controllable.
  - **If return value controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters, or does other unusual control flow?** If it reverts, the entire call reverts. No reentrancy possible.

Function: `updateMaxUnitCapacity()`

This function is used to update the security limit when funds are sent to the vault.

Branches and code coverage (including function calls)

Intended branches

- New `_maxUnitCapacity` is calculated correctly for multiple ranges of escrowed token balances.
  ☐ Test coverage

Function: `withdrawAll(uint256 vaultTokens, uint256[] minOut)`

This function allows users to burn their vault tokens and get back their original tokens.

Inputs

- `vaultTokens`
  - **Control**: Fully controlled.
  - **Constraints**: User must own at least this amount of vault tokens to burn.

  - **Impact**: This amount of tokens is burned from the user.
- `minOut`
  - **Control**: Fully controlled.
  - **Constraints**: N/A.
  - **Impact**: This is used for slippage protection.

Branches and code coverage (including function calls)

Intended branches

- Correct amount of tokens are returned to the user.
  ☑ Test coverage

5.4 Module: CatalystVaultCommon.sol

Function: `onSendAssetFailure(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, address escrowToken, uint32 blockNumberMod)`

This function is invoked by the IBC chain interface when an interchain asset transfer fails (the chain interface receives a time-out message or an ACK signalling an error from the IBC dispatcher).

It validates that the details of the transfer and releases the assets held in escrow, returning them back to the sender of the transfer.

In practice, the function is not invoked directly but rather as a super method by the overriding implementations in the Amplified or Volatile vaults.

Inputs

Note: This function can only be called directly by the IBC interface contract. The IBC interface performs a call when receiving a time-out packet or an ACK signalling an error from the IBC dispatcher, also assumed to be trusted. Therefore, the threat model has been developed with respect to the control an attacker can have without compromising these two trusted contracts.

- `channelId`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: None (only part of an emitted event).
- `toAccount`
  - **Control**: Semi-arbitrary (requires a legitimate transfer of assets to that ad-

dress).
  - **Constraints**: None.
  - **Impact**: Specifies the receiver of the funds on the other chain, part of the hash that identifies the transfer.
- `U`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Specifies the amount of units transferred, part of the hash that identifies the transfer.
- `escrowAmount`
  - **Control**: Semi-arbitraty (this amount has to be actually transferred).
  - **Constraints**: None.
  - **Impact**: Specifies the amount held in escrow, part of the hash that identifies the transfer.
- `escrowToken`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the asset held in escrow, part of the hash that identifies the transfer.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Nonce used to randomize the hash that identifies the transfer.

Branches and code coverage (including function calls)

Intended branches

- Computes the hash identifying the transfer, releases the associated assets held in escrow, and returns them to the user.
  ☑ Test coverage

Negative behavior

- Reverts if the transfer has not happened or the assets in escrow have already been released.
  ☑ Negative test

Function call analysis

- `rootFunction → ERC20(escrowToken).safeTransfer(fallbackAddress, escrowAmount)`

- **What is controllable?** `fallbackAddress` and `escrowAmount`.
- **If return value controllable, how is it used and how can it go wrong?** Not used.
- **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is not a concern.

Function: `onSendAssetSuccess(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, address escrowToken, uint32 blockNumberMod)`

This function is invoked by the IBC chain interface when an interchain asset transfer succeeds (the chain interface receives a successful ACK from the dispatcher).

It validates the details of the transfer and releases the assets held in escrow (permanently keeping them).

In practice, the function is not invoked directly but rather as a super method by the overriding implementations in the Amplified or Volatile vaults.

Inputs

Note: This function can only be called directly by the IBC interface contract. The IBC interface performs a call when receiving a successful ACK from the IBC dispatcher, also assumed to be trusted. Therefore, the threat model has been developed with respect to the control an attacker can have without compromising these two trusted contracts.

- `channelId`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: None (only part of an emitted event).
- `toAccount`
  - **Control**: Semi-arbitrary (requires a legitimate transfer of assets to that address).
  - **Constraints**: None.
  - **Impact**: Specifies the receiver of the funds on the other chain, part of the hash that identifies the transfer.
- `U`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Specifies the amount of units transferred, part of the hash that identifies the transfer.

- `escrowAmount`
  - **Control**: Semi-arbitraty (this amount has to be actually transferred).
  - **Constraints**: None.
  - **Impact**: Specifies the amount held in escrow, part of the hash that identifies the transfer.
- `escrowToken`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the asset held in escrow, part of the hash that identifies the transfer.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Nonce used to randomize the hash that identifies the transfer.

Branches and code coverage (including function calls)

Intended branches

- Computes the hash identifying the transfer, releases the associated assets held in escrow.
  ☑ Test coverage

Negative behavior

- Reverts if the transfer has not happened or the assets in escrow have already been released.
  ☑ Negative test

Function: `onSendLiquidityFailure(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, uint32 blockNumberMod)`

This function is invoked by the IBC chain interface when an interchain liquidity transfer fails (the chain interface receives a time-out message or an ACK signalling an error from the IBC dispatcher).

It validates the details of the transfer and releases the liquidity held in escrow, returning it back to the sender of the transfer.

In practice, the function is not invoked directly but rather as a super method by the overriding implementations in the Amplified or Volatile vaults.

Inputs

Note: This function can only be called directly by the IBC interface contract. The IBC interface performs a call when receiving a time-out packet or an ACK signalling an error from the IBC dispatcher, also assumed to be trusted. Therefore, the threat model has been developed with respect to the control an attacker can have without compromising these two trusted contracts.

- `channelId`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: None (only part of an emitted event).
- `toAccount`
  - **Control**: Semi-arbitrary (requires a legitimate transfer of assets to that address).
  - **Constraints**: None.
  - **Impact**: Specifies the receiver of the funds on the other chain, part of the hash that identifies the transfer.
- `U`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Specifies the amount of units transferred, part of the hash that identifies the transfer.
- `escrowAmount`
  - **Control**: Semi-arbitraty (this amount has to be actually transferred).
  - **Constraints**: None.
  - **Impact**: Specifies the amount held in escrow, part of the hash that identifies the transfer.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Nonce used to randomize the hash that identifies the transfer.

Branches and code coverage (including function calls)

Intended branches

- Computes the hash identifying the transfer and releases the liquidity held as escrow, minting the tokens back to the user that initiated the transfer.
  ☑ Test coverage

Negative behavior

- Reverts if the transfer has not happened or the escrow was already released.
  ☑ Negative test

Function: `onSendLiquiditySuccess(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, uint32 blockNumberMod)`

This function is invoked by the IBC chain interface when an interchain liquidity transfer succeeds (the chain interface receives a successful ACK from the dispatcher).

It validates the details of the transfer and releases the liquidity held in escrow (permanently keeping it).

In practice, the function is not invoked directly but rather as a super method by the overriding implementations in the Amplified or Volatile vaults.

Inputs

Note: This function can only be called directly by the IBC interface contract. The IBC interface performs a call when receiving a successful ACK from the IBC dispatcher, also assumed to be trusted. Therefore, the threat model has been developed with respect to the control an attacker can have without compromising these two trusted contracts.

- `channelId`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: None (only part of an emitted event).
- `toAccount`
  - **Control**: Semi-arbitrary (requires a legitimate transfer of assets to that address).
  - **Constraints**: None.
  - **Impact**: Specifies the receiver of the funds on the other chain, part of the hash that identifies the transfer.
- `U`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Specifies the amount of units transferred, part of the hash that identifies the transfer.
- `escrowAmount`
  - **Control**: Semi-arbitraty (this amount has to be actually transferred).
  - **Constraints**: None.
  - **Impact**: Specifies the amount held in escrow, part of the hash that identifies

the transfer.

- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Nonce used to randomize the hash that identifies the transfer.

Branches and code coverage (including function calls)

Intended branches

- Computes the hash identifying the transfer and releases the liquidity held in escrow.
  ☑ Test coverage

Negative behavior

- Reverts if the transfer has not happened or the escrow was already released.
  ☑ Negative test

5.5 Module: CatalystVaultVolatile.sol

Function: `depositMixed(uint256[] tokenAmounts, uint256 minOut)`

This function can be used to deposit assets into the vault, obtaining vault tokens representing the position in exchange.

Inputs

- `tokenAmounts`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the amounts of the tokens to be deposited.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the minimum amount of vault tokens to be minted (slippage protection).

Branches and code coverage (including function calls)

Intended branches

- Transfers to the vault the assets to be deposited, mints the corresponding amount

of vault tokens to the user, and performs slippage checks.
  ☑ Test coverage

Negative behavior

- Reverts if slippage is too high.
  ☑ Negative test
- Reverts if asset transfer from the user fails.
  ☑ Negative test

Function call analysis

- `rootFunction → ERC20(token).balanceOf(address(this))`
  - **What is controllable?** None.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable, used as part of the calculation of the amount of tokens to be minted.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy prevented via `nonReentrant` modifier.
- `rootFunction → ERC20(token).safeTransferFrom(msg.sender, address(this), tokenAmounts[it])`
  - **What is controllable?** Amount.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy prevented via `nonReentrant` modifier.

Function: `localSwap(address fromAsset, address toAsset, uint256 amount, uint256 minOut)`

This function can be used to perform a local swap between assets held by the vault. The caller must have given approval to the vault to pull the input asset.

Inputs

- `fromAsset`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the asset used as input for the swap.
- `toAsset`
  - **Control**: Arbitrary.
  - **Constraints**: None.

  - **Impact**: Determines the asset used as output for the swap.
- `amount`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from the user balance being sufficient).
  - **Impact**: Specifies the amount of input asset.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Specifies the minimum amount of output token for slippage checks.

Branches and code coverage (including function calls)

Intended branches

- Computes the swap fees and return value, performs slippage checks, and performs the transfers of the input and output tokens as well as of the governance fee.
  ☑ Test coverage

Negative behavior

- Reverts if the output amount is not enough (slippage check fail).
  ☑ Negative test
- Reverts if the caller's balance of the input asset is not enough.
  ☑ Negative test

Function call analysis

- `rootFunction → calcLocalSwap(fromAsset, toAsset, amount - fee)`
  - **What is controllable?** `fromAsset`, `toAsset`, and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not arbitrarily controllable in a meaningful way; used as the amount of output asset.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `calcLocalSwap → ERC20(fromAsset).balanceOf(address(this))`
  - **What is controllable?** `fromAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not arbitrarily controllable in a meaningful way; if `fromAsset` is not a vault asset the associated weight is zero.
  - **What happens if it reverts, reenters, or does other unusual control flow?**

Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.

- `calcLocalSwap → ERC20(toAsset).balanceOf(address(this))`
  - **What is controllable?** `toAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not arbitrarily controllable in a meaningful way; if `toAsset` is not a vault asset, the associated weight is zero.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(fromAsset).safeTransferFrom(msg.sender, address(this), amount)`
  - **What is controllable?** `fromAsset` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(toAsset).safeTransfer(msg.sender, out)`
  - **What is controllable?** `toAsset` (but not in a meaningful way).
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → _collectGovernanceFee(fromAsset, fee)`
  - **What is controllable?** `fromAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `_collectGovernanceFee → ERC20(asset).safeTransfer(factoryOwner(), governanceFeeAmount)`
  - **What is controllable?** `asset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.

Function: `onSendAssetSuccess(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, address escrowToken, uint32 blockNumberMod)`

This function handles IBC responses from cross-chain transfers. It can be invoked exclusively by the IBC chain interface contract.

It delegates almost all logic (including security checks) to `CatalystVaultCommon::onSendAssetSuccess`, only handling the update to the variable that tracks the unit capacity of the vault.

Inputs

- `channelId`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Used for event data.
- `toAccount`
  - **Control**: Arbitrary (matches the recipient of the transfer).
  - **Constraints**: None.
  - **Impact**: Recipient of the transfer (used to compute the hash identifying the transfer).
- `U`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Amount of units transferred.
- `escrowAmount`
  - **Control**: Matches the input asset amount.
  - **Constraints**: None.
  - **Impact**: Amount of assets held in escrow (used to compute the `sendAssetHash`).
- `escrowToken`
  - **Control**: Matches the input asset.
  - **Constraints**: None.
  - **Impact**: Asset held in escrow.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Used to compute `sendAssetHash`.

Branches and code coverage (including function calls)

Intended branches

- Invokes `CatalystVaultCommon::onSendAssetSuccess` and recomputes the `_usedUnitCapacity` to account for the transfer.
  ☑ Test coverage

Negative behavior

- Logic is shared with `CatalystVaultCommon::onSendAssetSuccess`.

Function call analysis

- `rootFunction → super.onSendAssetSuccess( ... )`
  - **What is controllable?** All parameters (callee validates them).
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers (indirectly, `CatalystVaultVolatile::onSendAssetSuccess` is not marked `nonReentrant`, but it invokes `CatalystVaultCommon::onSendAssetSuccess` which is).

Function: `onSendLiquiditySuccess(byte[32] channelId, byte[] toAccount, uint256 U, uint256 escrowAmount, uint32 blockNumberMod)`

This function handles IBC responses from cross-chain liquidity transfers. It can be invoked exclusively by the IBC chain interface contract.

It delegates almost all logic (including security checks) to `CatalystVaultCommon::onSendLiquiditySuccess`, only handling the update to the variable that tracks the unit capacity of the vault.

Inputs

- `channelId`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Used for event data.
- `toAccount`
  - **Control**: Arbitrary (matches the recipient of the transfer).
  - **Constraints**: None.
  - **Impact**: Recipient of the transfer (used to compute the hash identifying the

transfer).

- `U`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Amount of units transferred.
- `escrowAmount`
  - **Control**: Matches the input vault tokens amount.
  - **Constraints**: None.
  - **Impact**: Amount of vault tokens held in escrow (used to compute the `sendAssetHash`).
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Used to compute `sendAssetHash`.

Branches and code coverage (including function calls)

Intended branches

- Invokes `CatalystVaultCommon::onSendLiquiditySuccess` and recomputes the `_usedUnitCapacity` to account for the transfer.
  ☑ Test coverage

Negative behavior

- Logic is shared with `CatalystVaultCommon::onSendLiquiditySuccess`.

Function call analysis

- `rootFunction → super.onSendLiquiditySuccess(...)`
  - **What is controllable?** All parameters (callee validates them).
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers (indirectly, `CatalystVaultVolatile::onSendLiquiditySuccess` is not marked `nonReentrant`, but it invokes `CatalystVaultCommon::onSendLiquiditySuccess` which is).

Function: `receiveAsset(byte[32] channelId, byte[] fromVault, uint256 toAssetIndex, address toAccount, uint256 U, uint256 minOut, uint256 fromAmount, byte[] fromAsset, uint32 blockNumberMod)`

This function cannot be invoked directly but only by the IBC interface contract. It is invoked to handle an incoming swap.

A simplified version of this function exists, which does not take the `dataTarget` and `data` arguments. The business logic is shared between the two, with the function taking additional arguments invoking the function taking less arguments. The function call analysis section omits this call indirection for simplicity.

Inputs

- `channelId`
  - **Control**: None.
  - **Constraints**: Must correspond to a connected vault (together with `fromVault`).
  - **Impact**: Identifies the IBC channel used to connect to the other vault.
- `fromVault`
  - **Control**: None.
  - **Constraints**: Must correspond to a connected vault (together with `channelId`).
  - **Impact**: Identifies the vault on the source chain.
- `toAssetIndex`
  - **Control**: Arbitrary.
  - **Constraints**: Must correspond to an asset in the `_tokenIndexing` array.
  - **Impact**: Identifies the output asset for the swap.
- `toAccount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Idenfifies the recipient of the output assets.
- `U`
  - **Control**: Indirect partial control (computed by the sender vault based on the transferred amount).
  - **Constraints**: Limited to `_maxUnitCapacity`.
  - **Impact**: Determines the amount of units being transferred.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Minimum output amount (used for slippage checks).

- `fromAmount`
  - **Control**: Arbitrary (matches the input asset amount).
  - **Constraints**: None.
  - **Impact**: Only used as event data.
- `fromAsset`
  - **Control**: Arbitrary (matches the input asset).
  - **Constraints**: None.
  - **Impact**: Only used as event data.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: None.
  - **Impact**: Only used as event data.
- `dataTarget`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the contract handling the `ICatalystReceiver::onCatalystCall` callback.
- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Data supplied as second argument to the `onCatalystCall` callback.

Branches and code coverage (including function calls)

Intended branches

- Computes the output amount, checks for slippage, sends the output asset to the recipient, and invokes the callback.
  ☑ Test coverage

Negative behavior

- Reverts if the input units exceed the maximum unit capacity.
  ☑ Negative test
- Reverts if slippage checks fail.
  ☑ Negative test

Function call analysis

- `rootFunction` → `calcReceiveAsset(toAsset, U)`
  - **What is controllable?** `U` matches the units that were actually sent, `toAsset` depends on the `toAssetIndex` argument of `receiveAsset`

  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(toAsset).balanceOf(address(this))`
  - **What is controllable?** `toAsset`, depending on the `toAssetIndex` argument of `receiveAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(toAsset).safeTransfer(toAccount, purchasedTokens)`
  - **What is controllable?** `toAccount`, `toAsset` depends on the `toAssetIndex` argument of `receiveAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.

Function: `receiveLiquidity(byte[32] channelId, byte[] fromVault, address who, uint256 U, uint256 minVaultTokens, uint256 minReferenceAsset, uint256 fromAmount, uint32 blockNumberMod, address dataTarget, byte[] data)`

This function handles incoming cross-chain liquidity transfers from a connected vault. It can only be invoked by the IBC interface contract, which is assumed to be trusted. The threat model of the input arguments is evaluated with respect to the capabilities an attacker can have by sending a cross-chain transfer, without compromising the IBC dispatcher or IBC interface contracts.

Inputs

- `channelId`
  - **Control**: None.
  - **Constraints**: Must correspond to a connected vault (together with `fromVault`).
  - **Impact**: Identifies the IBC channel used to connect to the other vault.
- `fromVault`

  - **Control**: None.
  - **Constraints**: Must correspond to a connected vault (together with `channelId`).
  - **Impact**: Identifies the vault on the source chain.
- `who`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Identifies the address that receives the liquidity.
- `U`
  - **Control**: None (matches the units sent by the source vault).
  - **Constraints**: N/A.
  - **Impact**: Amount of units of liquidity being transferred.
- `minVaultTokens`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Minimum amount of output vault tokens (for slippage checks).
- `minReferenceAsset`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Minimum amount of output assets (for slippage checks).
- `fromAmount`
  - **Control**: None (matches the amount from the sender vault).
  - **Constraints**: N/A.
  - **Impact**: Used as event data.
- `blockNumberMod`
  - **Control**: None.
  - **Constraints**: N/A.
  - **Impact**: Used as event data.
- `dataTarget`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address used for `onCatalystCall` callback.
- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Data supplied to the `onCatalystCall` callback.

Branches and code coverage (including function calls)

Intended branches

- Checks the transfer does not exceed the maximum unit capacity, computes the amount of vault tokens to be minted, checks for slippage, mints vault tokens to the user, and invokes the callback handler.
  ☑ Test coverage

Negative behavior

- Reverts if the callback receiver reverts.
  ☑ Negative test
- Reverts if `(channelId, fromVault)` do not identify a connected vault.
  ☑ Negative test

Function call analysis

- `rootFunction → receiveLiquidity`
  - **What is controllable?** All arguments except.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `receiveLiquidity → ERC20(token).balanceOf(address(this))`
  - **What is controllable?** None.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable, used for slippage checks.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ICatalystReceiver(dataTarget).onCatalystCall(purchasedVaultTokens, data)`
  - **What is controllable?** `dataTarget` and `data`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up (intended); reentrancy is prevented via `nonReentrant` modifiers.

Function: `sendAsset(byte[32] channelId, byte[] toVault, byte[] toAccount, address fromAsset, uint8 toAssetIndex, uint256 amount, uint256 minOut, address fallbackUser, byte[] calldata_)`

This function can be used to perform a cross-chain swap.

Inputs

- `channelId`
  - **Control**: Arbitrary.
  - **Constraints**: Must correspond to a connected vault (together with `toVault`).
  - **Impact**: Identifies the IBC channel used to connect to the other vault.
- `toVault`
  - **Control**: Arbitrary.
  - **Constraints**: Must correspond to a connected vault (together with `channelId`).
  - **Impact**: Identifies the vault on the destination chain.
- `toAccount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Identifies the recipient of the swapped asset on the destination chain.
- `fromAsset`
  - **Control**: Arbitrary.
  - **Constraints**: No constraints.
  - **Impact**: Determines the address of the asset used as input for the swap.
- `toAssetIndex`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the output asset for the swap (by index).
- `amount`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from the caller balance being sufficient).
  - **Impact**: Determines the amount of the input asset.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Used to perform slippage checks.
- `fallbackUser`

  - **Control**: Arbitrary.
  - **Constraints**: `fallbackUser` $\neq$ `address(0)`.
  - **Impact**: Address that receives the input funds from escrow if the swap fails.
- `calldata_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Calldata provided to the user callback.

Branches and code coverage (including function calls)

Intended branches

- Computes vault fees and the units corresponding to the input amount, initiates a cross-chain asset swap by invoking the IBC interface, transfers the input assets from the sender in escrow, and collects governance fees.
  ☑ Test coverage

Negative behavior

- Reverts if `fallbackUser` is unspecified.
  ☑ Negative test
- Reverts if the hash identifying the transfer already exists.
  ☑ Negative test
- Reverts if sender balance is insufficient.
  ☑ Negative test

Function call analysis

- `rootFunction` → `calcSendAsset(fromAsset, amount - fee)`
  - **What is controllable?** `fromAsset` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable in a meaningful way; if `fromAsset` is not a vault asset (including a potentially rogue contract address), its weight is zero.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented by `nonReentrant` modifier.
- `calcSendAsset` → `ERC20(fromAsset).balanceOf(address(this))`
  - **What is controllable?** `fromAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable in a meaningful way; if `fromAsset` is not legitimate, its weight is zero.
  - **What happens if it reverts, reenters, or does other unusual control flow?**

Reverts are bubbled up; reentrancy is prevented by `nonReentrant` modifier.

- `rootFunction → CatalystIBCInterface(_chainInterface).sendCrossChainAsset(...)`
  - **What is controllable?** All parameters — some (like `channelId`, `toVault`) with restrictions.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented by `nonReentrant` modifier.
- `rootFunction → ERC20(fromAsset).safeTransferFrom(msg.sender, address(this), amount)`
  - **What is controllable?** `fromAsset` and `amount`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented by `nonReentrant` modifier.
- `rootFunction → _collectGovernanceFee(fromAsset, fee)`
  - **What is controllable?** `fromAsset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.
- `_collectGovernanceFee → ERC20(asset).safeTransfer(factoryOwner(), governanceFeeAmount)`
  - **What is controllable?** `asset`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up. Reentrancy is prevented via `nonReentrant` modifiers.

Function: `sendLiquidity(byte[32] channelId, byte[] toVault, byte[] toAccount, uint256 vaultTokens, uint256[2] minOut, address fallbackUser, byte[] calldata_)`

This function can be used to initiate a cross-chain liquidity transfer between two connected vaults.

Inputs

- `channelId`
  - **Control**: Arbitrary.
  - **Constraints**: Must correspond to a connected vault (together with `fromVault`).
  - **Impact**: Identifies the IBC channel used to connect to the other vault.
- `toVault`
  - **Control**: Arbitrary.
  - **Constraints**: Must correspond to a connected vault (together with `channelId`).
  - **Impact**: Identifies the vault on the destination chain.
- `toAccount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Idenfifies the recipient of the output assets.
- `vaultTokens`
  - **Control**: Arbitrary.
  - **Constraints**: Sender balance must be sufficient.
  - **Impact**: Determines the amount of vault tokens to be transferred.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Minimum output amount (used for slippage checks by the receiver vault).
- `fallbackUser`
  - **Control**: Arbitrary.
  - **Constraints**: Must not be the null address.
  - **Impact**: Address that receives the escrow if the transfer fails.
- `calldata_`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the `dataTarget` and `dataArguments` parameters used to invoke callbacks.

Branches and code coverage (including function calls)

Intended branches

- Burns the vault tokens to be transferred, computes the corresponding amount

of units, invokes the IBC interface to initiate the transfer, and stores information about the transfer for escrow.
  ☑ Test coverage

Negative behavior

- Reverts if the fallback user is the zero address.
  ☑ Negative test
- Reverts if the caller's vault tokens balance is insufficient.
  ☑ Negative test
- Reverts if there is a collision in the hash identifying the transfer.
  ☑ Negative test

Function call analysis

- `rootFunction` → `CatalystIBCInterface(_chainInterface).sendCrossChainLiquidity( ... )`
  - **What is controllable?** `channelId`, `toVault`, `toAccount`, `minOut`, `vaultTokens`, and `calldata_`.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction` → `_computeSendLiquidityHash( ... )`
  - **What is controllable?** `toAccount` and `vaultTokens`.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable in a meaningful way (hash of the arguments identifying the transfer).
  - **What happens if it reverts, reenters, or does other unusual control flow?** Cannot revert nor reenter.

Function: `withdrawAll(uint256 vaultTokens, uint256[] minOut)`

This function can be used to redeem vault tokens for the corresponding share of the assets contained in the vault.

The amounts of assets redeemed are balanced so that their value (using the current price) is equal.

Inputs

- `vaultTokens`

  - **Control**: Arbitrary.
  - **Constraints**: None (apart from the sender balance being sufficient).
  - **Impact**: Determines the amount of vault tokens to be redeemed.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the minimum amount of assets to be returned for slippage protection.

Branches and code coverage (including function calls)

Intended branches

- Burns the specified amount of vault tokens, computes and sends to the user the corresponding amount of the underlying assets, and performs slippage checks.
  ☑ Test coverage

Negative behavior

- Reverts if one (or more) output amount is not sufficient.
  ☑ Negative test
- Reverts if the user vault token balance is insufficient.
  ☑ Negative test

Function call analysis

- `rootFunction → ERC20(token).balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts (unlikely) are bubbled up; reentrancy (also unlikely) is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(token).safeTransfer(msg.sender, tokenAmount)`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.

Function: `withdrawMixed(uint256 vaultTokens, uint256[] withdrawRatio, uint256[] minOut)`

This function can be used to redeem vault tokens for the corresponding share of the assets contained in the vault.

The total value of the assets redeemed depends on the amount of vault tokens; unlike `withdrawAll`, the specific amounts of the redeemed assets are specified by the caller using the `withdrawRatio` parameter.

Inputs

- `vaultTokens`
  - **Control**: Arbitrary.
  - **Constraints**: None (apart from the sender balance being sufficient).
  - **Impact**: Determines the amount of vault tokens to be redeemed.
- `withdrawRatio`
  - **Control**: Arbitrary.
  - **Constraints**: Must consume all units awarded by burning the input vault tokens; implicitly, every item must be <= 100%; all items after one representing 100% must be zero.
  - **Impact**: Determines the repartition of output assets.
- `minOut`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Determines the minimum amount of assets to be returned for slippage protection.

Branches and code coverage (including function calls)

Intended branches

- Burns the specified amount of vault tokens, computes and sends to the user the corresponding amount of the underlying assets, and performs slippage checks.
  ☑ Test coverage

Negative behavior

- Reverts if one (or more) output amount is not sufficient.
  ☑ Negative test
- Reverts if the user vault token balance is insufficient.
  ☑ Negative test
- Reverts if a `withdrawRatio` element is > 1.

☑ Negative test

- Reverts if a `withdrawRatio` element != 0 follows an element representing 100%.
  ☑ Negative test
- Reverts if the specified `withdrawRatio` does not use all the units.
  ☑ Negative test

Function call analysis

- `rootFunction → ERC20(token).balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts (unlikely) are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.
- `rootFunction → ERC20(token).safeTransfer(msg.sender, tokenAmount)`
  - **What is controllable?** Nothing.
  - **If return value controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters, or does other unusual control flow?** Reverts are bubbled up; reentrancy is prevented via `nonReentrant` modifiers.

6 Audit Results

At the time of our audit, the audited code was not deployed to mainnet evm.

During our assessment on the scoped Catalyst contracts, we discovered two findings, all of which were informational in nature. Cata Labs, Inc acknowledged all findings and implemented fixes.

6.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.