# Veridise. Auditing Report

**Hardening Blockchain Security with Formal Methods**

FOR

Catalyst

Veridise Inc.
March 31, 2023

► **Prepared For:**

Catalabs
https://catalabs.org/

► **Prepared By:**

Nicholas Brown
Benjamin Sepanski
Bryan Tan

► **Contact Us:** contact@veridise.com

► **Version History:**

| | |
|---|---|
| 03 21, 2023 | V1 |
| 03 28, 2023 | Initial Draft |

# Contents



 

# 1 Executive Summary

From February 27, 2023 to March 24, 2023, Catalabs engaged Veridise to review the security of their project: Catalyst. The review covered the on-chain contracts responsible for implementing the protocol logic and handling Inter-Blockchain Communication (IBC) packets. Veridise conducted the assessment over 12 person-weeks, with 3 engineers reviewing code over 4 weeks from commits `0x53c5518`- `0x6589331`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

**Code assessment.** The code provided by the Catalabs developers implements an Automated Market Maker (AMM), as described in the Catalyst documentation, using Solidity smart contracts. This exchange is designed to allow for cross-chain swaps performed asynchronously. The Catalabs team has developed a robust mathematical framework for pricing these asynchronous swaps, and they implemented it in the Catalyst code base. To handle asynchronous transactions, they introduce a spread between the buy and sell prices which takes pending transactions into account. The major contracts consist of two types of Liquidity Pools and an IBC handler. The code also includes several modules necessary for fixed point integer computations of some analytic functions.

Our auditors feel that the code quality is very high. Contracts, data structures, and methods are extensively and accurately documented by the developers. This documentation extends to the concrete implementations of methods, where the developers have written comments explaining the rationale and assumptions behind each line of code. The auditors found the documentation extremely helpful during the auditing process. The developers also demonstrate knowledge of most security best practices. For example, they rely on several well-known and audited contracts from OpenZeppelin and have made efforts to mitigate or avoid issues such as: slippage issues, denial-of-service problems caused by arithmetic overflow, reentrancy attacks, etc.

Although the Veridise auditors read, actively engaged with, and analyzed the underlying mathematical framework while looking for bugs, the primary focus of the audit was the code base. The complex mathematics makes some functions very dense and complex, requiring a non-trivial effort to match code to the corresponding mathematical formulation. However, once Veridise auditors understood the relationships between mathematical functions and the code base, they were able to further stress test the mathematical framework by simulating and interacting with the protocol. Veridise auditors also noted that the current implementation uses a mocked IBC light client implementation. Consequently, the Veridise team has not audited the IBC setup and permissions, such as restricting port access via dynamic capabilities (see V-CAT-VUL-019).

During the audit, Catalabs developers implemented several fixes, had them reviewed by Veridise auditors, and added the fixes to the code base. Otherwise, the code remained frozen.

**Summary of issues detected.** The audit uncovered 25 issues, 3 of which are assessed to be of high or critical severity by the Veridise team. These issues could lead to theft of funds via a read-only reentrancy (V-CAT-VUL-001), a sign error which allowed profitable arbitrage via certain sequences of deposits and withdrawals (V-CAT-VUL-002), and a front-running issue during liquidity swaps (V-CAT-VUL-003).

The Veridise auditors also identified several medium-severity issues, including possible pool token burning (V-CAT-VUL-004), a missing check preventing massive weight changes by a factory owner (V-CAT-VUL-005), and an issue in the computation of exponents which prevents deposits into amplified pools when an asset has a zero balance (V-CAT-VUL-006). Additionally, the auditors found and reported several other minor issues.

The Catalabs developers fixed or acknowledged 25 of these issues. The Veridise team reviewed and approved each fix.

**Suggestions** After auditing the protocol, the auditors recommended fixes to each of the issues and reviewed each fix supplied by the Catalabs team. In addition to these recommendations, the auditors suggested the use of some standard EIPs (see V-CAT-VUL-025). These recommendations are listed in detail in Section 4.

Although the code is very well-documented, it assumes a thorough knowledge of the underlying mathematics. This requires additional effort from the reader to identify the correct part (and often, multiple parts) of the mathematical documentation corresponding to a given piece of code. Whenever a complex formula is used in the code, we recommend adding a direct reference to the associated formulae in the mathematical documentation. This allows readers to easily check that the code implements the desired mathematics, quickly understand the intent of a function by examining its mathematical representation (or vice versa), and separate concerns between the correctness of mathematics and its implementation.

We also recommend that the documentation make explicitly clear when a function parameter must be set properly for security reasons. For instance, as described in V-CAT-VUL-008, the `minOut` variable must be set to prevent front-running. However, readers may assume this variable's only usage is to prevent slippage, and not realize the importance of the parameter.

**Disclaimer.** We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

# 2 Project Dashboard

**Table 2.1:** Application Summary.

| Name | Version | Type | Platform |
|---|---|---|---|
| CatalabsCatalyst | `0x53c5518`- `0x6589331` | Solidity | Ethereum |

**Table 2.2:** Engagement Summary.

| Dates | Method | Consultants Engaged | Level of Effort |
|---|---|---|---|
| February 27- March 24, 20 | Manual & Tools | 3 | 12 person-weeks |

**Table 2.3:** Vulnerability Summary.

| Name | Number | Resolved |
|---|---|---|
| Critical-Severity Issues | 2 | 2 |
| High-Severity Issues | 1 | 1 |
| Medium-Severity Issues | 3 | 3 |
| Low-Severity Issues | 7 | 7 |
| Warning-Severity Issues | 7 | 7 |
| Informational-Severity Issues | 5 | 5 |
| TOTAL | 25 | 25 |

**Table 2.4:** Category Breakdown.

| Name | Number |
|---|---|
| Access Control | 0 |
| Data Validation | 3 |
| Denial of Service | 3 |
| Frontrunning | 2 |
| Gas Optimization | 1 |
| Locked Funds | 0 |
| Logic Error | 7 |
| Maintainability | 7 |
| Missing/Incorrect Events | 0 |
| Reentrancy | 1 |

# Audit Goals and Scope 3

## 3.1 Audit Goals

The engagement was scoped to provide a security assessment of Catalabs's smart contracts. In our audit, we sought to answer the following questions:

- Can a malicious user use deposits and withdrawals to induce a profitable arbitrage?
- Can a malicious user use the knowledge that a cross-chain swap will fail or succeed to perform trades which cause the pool to lose money?
- Can deposits, withdrawals, and local swaps lead to incorrect pricing during cross-chain swaps?
- Can multiple, concurrent cross-chain swaps interfere with each other?
- Can pools be sure the cross-chain messages they receive are from authenticated pools?
- Can a malicious user use liquidity swaps to make profitable withdraws or swaps?
- Can front-runners (or adversaries with more efficient communication) profit by racing asynchronous transactions?
- Are failed cross-chain transactions properly handled, i.e. are users guaranteed to be refunded, and is the receiving pool sure not to pay out?
- Are all unchecked blocks guaranteed to be free of integer overflow vulnerabilities?
- Are there any sequences of actions which can break the Catalyst pool invariants?

## 3.2 Audit Methodology & Scope

To address the questions above, our audit involved an extensive manual audit. This included auditors reading documentation, reviewing code and tests, and writing proof-of-concept exploits using the Brownie development environment provided by the Catalabs developers, among other tasks.

Scope. The scope of this audit is limited to the `evm/` folder of the source code provided by the Catalabs developers, which contains the smart contract implementation of Catalyst. In particular, the audit reviewed the following files of Catalyst, as well as their associated interfaces:

- `contracts/CatalystIBCInterface.sol`
- `contracts/CatalystIBCPayload.sol`
- `contracts/SwapPoolAmplified.sol`
- `contracts/SwapPoolCommon.sol`
- `contracts/SwapPoolFactory.sol`
- `contracts/SwapPoolVolatile.sol`

## 3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

**Table 3.1:** Severity Breakdown.

| | Somewhat Bad | Bad | Very Bad | Protocol Breaking |
|---|---|---|---|---|
| Not Likely | Info | Warning | Low | Medium |
| Likely | Warning | Low | Medium | High |
| Very Likely | Low | Medium | High | Critical |

In this case, we judge the likelihood of a vulnerability as follows:

| | |
|---|---|
| Not Likely | A small set of users must make a specific mistake |
| Likely | Requires a complex series of steps by almost any user(s)<br>- OR -<br>Requires a small set of users to perform an action |
| Very Likely | Can be easily performed by almost anyone |

In addition, we judge the impact of a vulnerability as follows:

| | |
|---|---|
| Somewhat Bad | Inconveniences a small number of users and can be fixed by the user |
| Bad | Affects a large number of people and can be fixed by the user<br>- OR -<br>Affects a very small number of people and requires aid to fix |
| Very Bad | Affects a large number of people and requires aid to fix<br>- OR -<br>Disrupts the intended behavior of the protocol for a small group of users through no fault of their own |
| Protocol Breaking | Disrupts the intended behavior of the protocol for a large group of users through no fault of their own |

# Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

**Table 4.1:** Summary of Discovered Vulnerabilities.

| ID | Description | Severity | Status |
|---|---|---|---|
| V-CAT-VUL-001 | Read-only reentrancy on cross-chain pool functions | Critical | Fixed |
| V-CAT-VUL-002 | Profitable arbitrage due to sign error | Critical | Fixed |
| V-CAT-VUL-003 | Using minOut does not prevent frontrunning | High | Fixed |
| V-CAT-VUL-004 | Incorrect withdrawRatios can burn user funds | Medium | Fixed |
| V-CAT-VUL-005 | No size limit on weights in setWeights | Medium | Fixed |
| V-CAT-VUL-006 | Cannot deposit into asset with zero balance | Medium | Fixed |
| V-CAT-VUL-007 | IBCDispatcher address not validated | Low | Fixed |
| V-CAT-VUL-008 | Frontrunners can profit on liquidity swaps | Low | Intended Behavior |
| V-CAT-VUL-009 | Fee collection occurs before funds transfer | Low | Fixed |
| V-CAT-VUL-010 | _usedUnitCapacity may be larger than max | Low | Fixed |
| V-CAT-VUL-011 | Unescrowed balances can exceed _unitTracker | Low | Fixed |
| V-CAT-VUL-012 | Unchecked block around pool tokens escrow increase | Low | Fixed |
| V-CAT-VUL-013 | Unchecked addition of weighted escrow amount | Low | Fixed |
| V-CAT-VUL-014 | Use of hard-coded value for decimals | Warning | Intended Behavior |
| V-CAT-VUL-015 | Require statement re-implements modifier | Warning | Fixed |
| V-CAT-VUL-016 | Confusing naming of packet handling functions | Warning | Intended Behavior |
| V-CAT-VUL-017 | Swap pool methods missing override specifiers | Warning | Fixed |
| V-CAT-VUL-018 | Move parameter checks into factory | Warning | Fixed |
| V-CAT-VUL-019 | IBC packet sender needs stricter validation | Warning | Acknowledged |
| V-CAT-VUL-020 | Amplified calcSendAsset reverts on 0 | Warning | Fixed |
| V-CAT-VUL-021 | Gas savings by using calldata | Info | Acknowledged |
| V-CAT-VUL-022 | Reimplementation of FixedPointMathLib.mulWadDown | Info | Fixed |
| V-CAT-VUL-023 | Duplicated functionality in initializeSwapCurves | Info | Acknowledged |
| V-CAT-VUL-024 | Out-of-date documentation: variable naming | Info | Fixed |
| V-CAT-VUL-025 | Code Structure Suggestion: Use Introspection | Info | Acknowledged |

 

## 4.1 Detailed Description of Bugs

### 4.1.1 V-CAT-VUL-001: Read-only reentrancy on cross-chain pool functions

| | | | |
|---|---|---|---|
| **Severity** | Critical | **Commit** | 6f066be |
| **Type** | Reentrancy | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol`, `SwapPoolAmplified.sol`, `SwapPoolCommon.sol` | | |
| **Functions** | See description | | |

The following functions are not marked with the `nonReentrant` modifier:

- receiveAsset
- sendLiquidity
- receiveLiquidity
- sendAssetAck/Timeout
- sendLiquidityAck/Timeout

This means that other functions which are marked as `nonReentrant` (such as the swap and withdrawal functions) may reenter into these functions. For instance, if one of the tokens implements ERC777, this could occur during a `safeTransfer` or `safeTransferFrom`.

Each of the listed functions also uses `ERC20.balanceOf` to determine the current asset balances of the pool. However, these balances can become out of sync with the pool token balance by reentering during a withdrawal or deposit.

For example, a user may

1. Deposit a large amount of each asset.
2. Immediately withdraw the entire amount
3. Reenter during the transferFrom inside of `withdrawAll`. If the first asset happens to be an ERC777, this will leave the pool in a state in which the pool tokens have been burnt, but only some of the corresponding assets have been withdrawn.

A similar pattern may occur if the last token is an ERC777 and the attacker reenters the contract during the last transferFrom of a `depositMixed`. In this case, all assets have been deposited, but no pool tokens have been minted yet.

**Impact** Any function which is dependent on the current balance of pool assets may be using an out-of-date value if it is not marked as `nonReentrant`. We outline several attacks below, but this is not a comprehensive list.

**SwapPoolAmplified.sendLiquidity** Suppose Alice is sending pool tokens from `P1` to `P2`. Before she calls `sendLiquidity`, Alice obtains a flashloan of each asset in `P1`, deposits the assets, and then immediately withdraws all of them. The first asset happens to be an ERC777, allowing Alice to call `sendLiquidity` during the first `transferFrom`. Then, when `weightAssetBalance` is computed, all but the first asset have an inflated balance.

```
uint256 weightAssetBalance = weight * (ERC20(token).balanceOf(address(this)) -
    _escrowedTokens[token]);
```

This leads to a larger `walpha_0_ampped`, which is directly related to the amount of units `U` sent to `P2`.

When execution returns to `withdrawAll`, the only value that has changed is `_escrowedPoolTokens`, so Alice does receive slightly less than she deposited. However, this can still result in profit for Alice.

We have verified the above attack with a brownie script, producing the following sequence, in which an attacker extracts funds directly from a pool.

```
[catalyst] ~/Veridise/audits/catalyst/evm % brownie run veridise/reenter_liq_send.py
     --network development --silent
Brownie v1.19.3 - Python development framework for Ethereum

EvmProject is the active project.

Launching 'ganache-cli --chain.vmErrorsOnRPCResponse true --server.port 8545 --miner.
    blockGasLimit 12000000 --wallet.totalAccounts 10 --hardfork istanbul --wallet.
    mnemonic brownie'...

Running 'scripts/veridise/reenter_liq_send.py::main'...
Initial State:
Pool[0] balances: [10000, 10000, 10000]
Pool[0] has 1.0000 pts, attacker owns 0.0000
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.0000 pts, attacker owns 0.0000

        Attacker deposits [2500, 2500, 2500] into pool, receiving 0.2500 tokens

Pool[0] balances: [12500, 12500, 12500]
Pool[0] has 1.2500 pts, attacker owns 0.2500
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.0000 pts, attacker owns 0.0000

        Attacker now deposits [6250, 6250, 6250] to receive 0.6250
        additional tokens

Pool[0] balances: [18750, 18750, 18750]
Pool[0] has 1.8750 pts, attacker owns 0.8750
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.0000 pts, attacker owns 0.0000

        The attacker now withdraws 0.6250 tokens, reentering
        on the first transfer to send the remaining 0.2500 with
        a call to sendLiquidity


        The attacker received (6249, 6249, 6249) from their
        withdrawal of 0.6250 tokens, for a profit of [-1, -1, -1]

Pool[0] balances: [12501, 12501, 12501]
Pool[0] has 1.0000 pts, attacker owns 0.0000
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.0000 pts, attacker owns 0.0000
```

 

```

        The liquidity send finalizes and is ack'ed!
        The attacker receives 0.2882 pool tokens

Pool[0] balances: [12501, 12501, 12501]
Pool[0] has 1.0000 pts, attacker owns 0.0000
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.2882 pts, attacker owns 0.2882

(2686, 2686, 2686)

        The attacker can now withdraw, receiving (2686, 2686, 2686). This is a total
    profit of
        (2686, 2686, 2686) + (6249, 6249, 6249) - [6250, 6250, 6250] - [2500, 2500,
    2500]
        = [185, 185, 185]

Final state:
Pool[0] balances: [12501, 12501, 12501]
Pool[0] has 1.0000 pts, attacker owns 0.0000
Pool[1] balances: [7314, 7314, 7314]
Pool[1] has 1.0000 pts, attacker owns 0.0000

        Reverting chain to initial state to see what intended send liquidity receipt
    was...

Reverted State:
Pool[0] balances: [12500, 12500, 12500]
Pool[0] has 1.2500 pts, attacker owns 0.2500
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.0000 pts, attacker owns 0.0000

        A normal user would just send the 0.2500 directly...
        receiving 0.2349 pool tokens

Pool[0] balances: [12500, 12500, 12500]
Pool[0] has 1.0000 pts, attacker owns 0.0000
Pool[1] balances: [10000, 10000, 10000]
Pool[1] has 1.2349 pts, attacker owns 0.2349

Terminating local RPC client...
```

**IBC-Trusted Methods (receive/ack)** Note that the `receiveAsset`, `sendAssetAck`, `sendLiquidityAck`, and `receiveLiquidity` functions are only callable by the `_chainInterface`. However, this does not fully protect them against a malicious adversary.

Suppose Bob owns an account which can call a function `f` which may eventually cause `_chainInterface` to call any of the above functions. Bob can then reenter through `deposit` or `withdrawal`'s transfers by invoking `f` during the receiver callback. In particular, whoever is in charge of the IBC router can exploit these functions.

More concretely, for the amplified pool

- Reentering a `receiveAsset` method from a `withdrawAll` allows all token balances to be read into memory before the `receiveAsset` occurs, while allowing the `receiveAsset` to take place after only one of the assets has been withdrawn from.
- Depositing, then reentering a `receiveLiquidity` method from a `withdrawAll` can allow the IBC controller to reduce the amount of liquidity others receive.
- While we are not aware of exploits inside the “ack” and “timeout” methods, the possibility to change the escrow amounts during the execution of the withdraw and deposit methods may lead to unexpected results.

**SwapPoolVolatile** At this point in time we are unable to identify an explicit exploit for these functions in the volatile pool. Many of the cross-chain functions do not read from the balances of any assets, and the withdraw/deposit functions call `balanceOf` immediately before the corresponding transfer. However, we still strongly advise adding the `nonReentrant` modifier for maintainability purposes, and due to the subtle nature of these types of attacks.

**Recommendation** Mark the functions as `nonReentrant`.

**Developer Response** The recommendation has been applied.

 

### 4.1.2 V-CAT-VUL-002: Depositing uneven balances in amplified pools leads to unexpected results

| Severity | Critical | Commit | 6f066be |
|---|---|---|---|
| Type | Logic Error | Status | Fixed |
| Files | SwapPoolAmplified.sol | | |
| Functions | N/A | | |

Certain sequences of large, adversarial deposits and withdrawals can lead to theft of funds from the pool.

Assume a pool starts in equilibrium. An attacker performs the following steps:

1. Deposit a large, but unevenly distributed amount of funds into the pool, receiving `pt1` pool tokens.
2. Withdraw all `pt1` pool tokens to receive an even distribution of funds.
3. Deposit all of those funds to receive `pt2` pool tokens.
4. Withdraw all `pt2` pool tokens to receive an even distribution of funds.
5. Arbitrage the pool back to equilibrium.

While this series of events seems totally innocuous, for certain values two unexpected effects manifest:

- The second deposit receives more pool tokens than the first, i.e. `pt2 > pt1`. Likewise, the second withdrawal receives more funds than the first.
- The total profit obtained from arbitraging in step (5.) **exceeds** the total cost of the deposits/withdrawals incurred by steps (1.)-(4.).

While the first effect signals something is awry in the implementation, the second allows for direct extraction of funds from the pool. This effect grows super-linearly as the attacker's funds approach or exceed that of the pool. It eventually becomes impossible once the withdrawal in step (4.) completely drains one asset due to issue V-CAT-VUL-006.

**Impact** This attack can directly extract funds from the pool. Further, since the attack may be performed synchronously, attackers may receive funding from flashloans.

We ran a concrete version of this attack against a pool with the following configuration (assumed to be in equilibrium):

- Amplification: 0.3
- Weights: [1, 1, 1]
- Asset Balances: [100000, 100000, 100000].

A summary of the attack results is shown below:

```
Flashloan of size 1.00% of pool extracts -0.00% of pool
Flashloan of size 10.00% of pool extracts -0.00% of pool
Flashloan of size 50.00% of pool extracts 0.41% of pool
Flashloan of size 75.00% of pool extracts 1.71% of pool
Flashloan of size 100.00% of pool extracts 5.09% of pool
Flashloan of size 120.00% of pool extracts 13.20% of pool
```

Note that the attacker requires a large number of funds for this attack to be profitable, but the attack rewards grow rapidly as the attacker's funds increase relative to the pool.

Below we have included concrete instantiations of the attack used to create the above code listing.

```
0.01
Initial token balances: [100000, 100000, 100000]
Deposited [0, 1500.0, 1500.0] for total of 3000.0, new balances: [100000, 101500, 101500]
Received 0.0100 tokens (out of 1.0100 total: 0.99%)
Withdrew to receive (996, 1000, 1000) for total of 2996
Deposited those amounts and received 0.0100 tokens (out of 1.0100 total: 0.99%)
Withdrew (995, 1000, 1000) for total of 2995
Withdrawals - Deposits: -5.0
Swapped 996 for 997 for profit of 1, new token balances: [100001, 99503, 100500]
Swapped 498 for 498 for profit of 0, new token balances: [100001, 100001, 100002]
Total arbitrage profit: 1
Total Profit: -4.0

0.1
Initial token balances: [100000, 100000, 100000]
Deposited [0, 15000.0, 15000.0] for total of 30000.0, new balances: [100000, 115000, 115000]
Received 0.0993 tokens (out of 1.0993 total: 9.03%)
Withdrew to receive (9660, 10061, 10061) for total of 29782
Deposited those amounts and received 0.0993 tokens (out of 1.0993 total: 9.03%)
Withdrew (9660, 10061, 10061) for total of 29782
Withdrawals - Deposits: -218.0
Swapped 9732 for 9876 for profit of 144, new token balances: [100072, 95063, 104939]
Swapped 5009 for 5082 for profit of 73, new token balances: [100072, 100072, 99857]
Swapped 215 for 214 for profit of -1, new token balances: [99858, 100072, 100072]
Total arbitrage profit: 216
Total Profit: -2.0

0.5
Initial token balances: [100000, 100000, 100000]
Deposited [0, 75000.0, 75000.0] for total of 150000.0, new balances: [100000, 175000, 175000]
Received 0.4863 tokens (out of 1.4863 total: 32.72%)
Withdrew to receive (43746, 50840, 50840) for total of 145426
Deposited those amounts and received 0.4894 tokens (out of 1.4894 total: 32.86%)
Withdrew (43935, 51058, 51058) for total of 146051
Withdrawals - Deposits: -3949.0
Swapped 45251 for 48645 for profit of 3394, new token balances: [101316, 75297, 123942]
Swapped 26019 for 27804 for profit of 1785, new token balances: [101316, 101316, 96138]
Swapped 5178 for 5178 for profit of 0, new token balances: [96138, 101316, 101316]
Total arbitrage profit: 5179
Total Profit: 1230.0

0.75
Initial token balances: [100000, 100000, 100000]
```

 

```
Deposited [0, 112500.0, 112500.0] for total of 225000.0, new balances: [100000,
    212500, 212500]
Received 0.7226 tokens (out of 1.7226 total: 41.95%)
Withdrew to receive (62810, 76415, 76415) for total of 215640
Deposited those amounts and received 0.7377 tokens (out of 1.7377 total: 42.45%)
Withdrew (63577, 77326, 77326) for total of 218229
Withdrawals - Deposits: -6771.0
Swapped 65834 for 73586 for profit of 7752, new token balances: [102257, 61588,
    135174]
Swapped 40669 for 44826 for profit of 4157, new token balances: [102257, 102257,
    90348]
Swapped 11909 for 11908 for profit of -1, new token balances: [90349, 102257, 102257]
Total arbitrage profit: 11908
Total Profit: 5137.0

1.0
Initial token balances: [100000, 100000, 100000]
Deposited [0, 150000.0, 150000.0] for total of 300000.0, new balances: [100000,
    250000, 250000]
Received 0.9560 tokens (out of 1.9560 total: 48.88%)
Withdrew to receive (80814, 101958, 101958) for total of 284730
Deposited those amounts and received 1.0066 tokens (out of 2.0066 total: 50.16%)
Withdrew (82998, 104623, 104623) for total of 292244
Withdrawals - Deposits: -7756.0
Swapped 85583 for 100502 for profit of 14919, new token balances: [102585, 44875,
    145377]
Swapped 57710 for 65818 for profit of 8108, new token balances: [102585, 102585,
    79559]
Swapped 23026 for 23026 for profit of 0, new token balances: [79559, 102585, 102585]
Total arbitrage profit: 23027
Total Profit: 15271.0

1.2
Initial token balances: [100000, 100000, 100000]
Deposited [0, 180000.0, 180000.0] for total of 360000.0, new balances: [100000,
    280000, 280000]
Received 1.1412 tokens (out of 2.1412 total: 53.30%)
Withdrew to receive (94648, 122357, 122357) for total of 339362
Deposited those amounts and received 1.2706 tokens (out of 2.2706 total: 55.96%)
Withdrew (99526, 128413, 128413) for total of 356352
Withdrawals - Deposits: -3648.0
Swapped 100742 for 128982 for profit of 28240, new token balances: [101216, 22605,
    151587]
Swapped 78611 for 93623 for profit of 15012, new token balances: [101216, 101216,
    57964]
Swapped 43252 for 43251 for profit of -1, new token balances: [57965, 101216, 101216]
Total arbitrage profit: 43251
Total Profit: 39603.0
```

**Recommendation** We are still working to understand the underlying issue which leads to this vulnerability. As we continue investigating, we will update this issue with more information and further recommendations.

**Developer Response** This appears due to a sign error in withdrawal functions. When computing the fraction of pool tokens, the withdrawn number of pool tokens should be subtracted, not added.

 

### 4.1.3 V-CAT-VUL-003: Liquidity swaps rely on pool token amounts to prevent front-running

| | | | |
|---|---|---|---|
| **Severity** | High | **Commit** | b3b5a17 |
| **Type** | Frontrunning | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol`, `SwapPoolAmplified.sol` | | |
| **Functions** | receiveLiquidity(), sendLiquidity() | | |

When a liquidity swap occurs between two pools `P0` and `P1`, the sender protects against front-running on `P0` by providing a `minOut` parameter. If someone deposits into `P1` right before the swap, then the amount of pool tokens received on `P1` will be less than expected, and the swap will revert.

However, if someone is simultaneously performing liquidity swaps from `P1` with a third pool `P2`, then the value of each pool token in `P1` may change. In particular, if Alice performs a liquidity swap from `P0` to `P1` with some `minOut` required number of pool tokens to receive on pool 1, she cannot be sure of the value (in units) of those tokens.

**Impact** If an adversary is able to manipulate the value of the tokens on `P1`, they still may be able to front-run liquidity swappers. This may allow them to enact the attack described in related issue V-CAT-VUL-008.

For example, consider an attack similar to the one described in the above issue, in which Alice front-runs Bob in an attempt to profit off of his liquidity send. In this scenario,

1. Pools `P1` and `P2` each contain 1000 each of assets A, B, and C, with a total supply each of 1000 pool tokens. Neither Bob nor Alice own any stake in either pool.
2. Bob deposits 250 of each asset into pool 1. He now owns 250 pool tokens in `P1`, which has balances [1250, 1250, 1250].
3. Bob now decides to perform a liquidity swap, wanting to send 125 of his pool tokens from `P1` to `P2` via a liquidity swap.
    a) Alice notices this is about to happen and deposits [250, 250, 250] just before the swap occurs, receiving 250 pool tokens in `P1` (which now has balances [1500,1500,1500].
    b) However, Bob anticipated this situation and set the `minOut` parameter. Having just checked that there are 1000 pool tokens in `P2`, Bob expects to receive 125/(1250-125)*1000 = 111.111... pool tokens, and sets `minOut = 111`. Since Alice front-ran Bob, he now thinks that he will only receive 125/(1500-125)*1000 = 90 tokens, causing the liquidity send to revert.
4. To thwart the set `minOut`, Alice now deposits [222.3, 222.3, 222.3] into `P2`, receiving 222.3 pool tokens in `P2`.
5. Now Bob's sendLiquidity request is received. He receives 125/(1500-124)*1222.3 = 111.111... pool tokens in `P2`, as desired.
6. Alice now withdraws her 222.3 tokens from from `P2`, receiving [203.7, 203.7, 203.7] and incurring a loss of [19.6, 19.6, 19.6].
7. Bob's sendLiquidity is now ack'ed back at `P1`. Alice withdraws her 125 pool tokens from `P1` to receive 125 / (1500-125) * [1500,1500,1500] = [272, 272, 272] of assets A, B, and C.

Alice has profited 4.1 of each asset in `A`.

**Recommendation** Allow users to specify the minimum number of units (or equivalent value in units) they wish to receive on the receiving pool, in addition to the number of tokens.

There are multiple approaches here, but the user should be able to specify some extra minimum value that prevents adversaries from manipulating pool token prices so that front-running is prevented.

**Developer Response** The developers have implemented an added check, allowing the user to set a minimum reference asset amount, as well as a minimum number of pool tokens.

**Updated Recommendation** The previously-recommended fix does not prevent front-running since Alice has both inflated `P2`'s pool tokens and `P2`'s asset balances/reference amounts enough that Bob is fully satisfied with the transaction. Only the pool has received a loss.

The underlying issue is that `minOut` can be computed using stale values for the asset balances of `P1`. This can be amended by either computing `minOut` as part of the `sendLiquidity` transaction (taking the receiving pool's balances/total supply as input), or allowing Bob to specify a "maxBalance" inside of `sendLiquidity`. For instance, if Bob specifies that he used asset balances of [1250, 1250, 1250] to compute `minOut`, the `sendLiquidity` would revert before any cross-chain interactions began.

**Developer Response** After further analysis, cases in which Alice deposits on `P2` are equivalent to a sequence of swaps. In fact, the events described above maintains the invariant for a volatile pool (after all parties have withdrawn).

However, this issue does highlight a closely related front-running issue in which Alice already has a stake in `P2`, and withdraws instead of depositing. This can still cause the pool tokens to achieve the desired `minOut` value, but decreases the asset reference amount.

**Auditor Response** We appreciate the above analysis, and have verified it on our side as well. However, one continued item of concern for us with the reference asset balance fix is the possibility of third-party liquidity swaps. For instance, Alice may inflate Bob's received pool tokens on `P2` by performing her own liquidity swap from a third pool `P3` targeting `P2`. She can then swap her pool tokens back once Bob's swap is confirmed. This operation will not affect the reference asset balance amount, so allows Alice to successfully front-run.

Note that while this does not cause the pool to lose money, it does hurt Bob's stake in `P1` (while not affecting his stake in `P2`).

**Developer Response** Although the send liquidity from `P3` to `P2` will not affect the asset balances, it does affect the percentage of the pool tokens which Bob owns (decreasing the number). This decreased percentage will lead to a decrease in the reference asset balance, preventing Alice from front-running.

 

### 4.1.4 V-CAT-VUL-004: withdrawMixed ratios with all entries less than one can cause users to lose assets

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | b3b5a17 |
| **Type** | Logic Error | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol`, `SwapPoolAmplified.sol` | | |
| **Functions** | withdrawMixed() | | |

The `withdrawMixed()` method in `SwapPoolVolatile` and `SwapPoolAmplified` allows a user to burn pool tokens in exchange for assets stored in the pool token. Because a pool may contain multiple tokens, the function provides a `withdrawRatio` argument that can be used to specify the distribution of those tokens (in terms of Catalyst units). The distribution is applied in a cumulative manner; for example, a `withdrawRatio` of 33.33%, 50%, 100% means "withdraw 33.33% of the units as token A, then withdraw 50% of the remaining units (i.e., 50% * 66.67% = 33.33%) as token B, and then withdraw all of the remaining units after that as token C.

The `withdrawMixed()` method indirectly checks that the sum of the withdraw ratio entries cannot exceed 100%; however, it does not check that all of the units are actually exchanged for assets.

**Impact** It is possible for a user to call `withdrawMixed()` with all ratio values less than 100%, burning pool tokens without getting the expected amount of assets out. As an extreme example, setting all `withdrawRatio` entries to 0 will result in pool tokens being burned but no assets being transferred to the caller. Such a situation is possible if there are bugs in off-chain frontends to Catalyst or in third-party smart contract integrations with the Catalyst swap pools.

**Recommendation** The developer should check that the `withdrawRatio` entries contain an entry equal to 1 (as a fixed point number with 18 decimals), either directly or by checking another quantity. For example, they can:

- Enforce that all units are consumed by the end of the `withdrawMixed()` method. This ensures that the `withdrawRatio` entries corresponds to at least 100% of the units.
- Check that after all units are consumed, the remaining `withdrawRatio` entries are 0. This will help prevent some user errors where they may accidentally specify a nonzero `withdrawRatio` entry after an entry that corresponds to `100%`.

**Developer Response** The developers have implemented both recommendations.

### 4.1.5 V-CAT-VUL-005: Malicious factory owners could abuse setWeights for Volatile pools

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | 6f066be |
| **Type** | Logic Error | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol` | | |
| **Functions** | setWeights() | | |

The factory owner of a volatile pool can set the weights to any values they desire. In particular, the `MIN_ADJUSTMENT_TIME` does not prevent the weights from changing by many orders of magnitude in seconds.

For example, consider the following sequence:

```
Time is 1678914550
Weights are [1, 1, 1]

Malicious factory owner sets weights to ['2**200', '2**200', '2**200']
over the course of 7 days

One block is mined, 14 seconds pass. Time is now 1678914564

The malicious owner now sends 1 pool token(s)
via a sendLiquidity, causing the weights to be updated.
Note that there are 10**18.00 total pool tokens.

Weights are ['2**184.70', '2**184.70', '2**184.70']
Units sent to other pool: 2**186.28575448233389
```

By setting the target weights to be enormous values, the malicious factory owner is able produce a huge number of units from only 1 pool token after only a single block has been mined on the chain.

**Impact** Liquidity providers may be hesitant to provide liquidity without additional protection. Further, any interior key breach which leads to loss of control over the factory owner could lead to pool funds being lost.

**Recommendation** Set a delay before updating the weights, or enforce a maximum relative change per unit of time.

**Developer Response** The factory owner always needs to be a time-locked contract. Otherwise a compromised factory owner could cause significant damage in other ways as well (e.g. by connecting to a pool with assets under owner control).

A maximum allowed relative change has been applied (factor of 10). Similarly, a relative maximum change of the amplification value has also been set (factor of 2).

### 4.1.6 V-CAT-VUL-006: Cannot deposit or swap into asset with zero balance

| | | | |
|---|---|---|---|
| **Severity** | Medium | **Commit** | 6f066be |
| **Type** | Denial of Service | **Status** | Fixed |
| **Files** | `SwapPoolAmplified.sol` | | |
| **Functions** | depositMixed(), calcLocalSwap(), calcSendAsset() | | |

As described in issue V-CAT-VUL-020, the `_calcPriceCurveArea` function reverts when the base of the exponent is zero. When depositing or swapping into an asset which has a balance of zero, this causes the contract to revert (note that the `depositMixed` does not use `_calcPriceCurveArea`, but does compute a power with a (weighted) asset balance as the base).

**Impact** Once an asset is dropped to a zero balance, the asset can no longer be brought above zero except by transferring directly to the pool.

**Recommendation** When computing a power `x**y` in which `x` might be zero (e.g. the power of any weighted balance), handle the zero case separately. This could be easily handled by changing the implementation of `powWad`.

**Developer Response** When computing balance0, cases where the balance is 0 is also handled.

### 4.1.7 V-CAT-VUL-007: IBCDispatcher address not validated

| **Severity** | Low | **Commit** | 53c5518 |
|---|---|---|---|
| **Type** | Data Validation | **Status** | Fixed |
| **Files** | `CatalystIBCInterface.sol` | | |
| **Functions** | constructor() | | |

```
constructor(address IBCDispatcher_) {
    IBC_DISPATCHER = IBCDispatcher_;
}
```

**Snippet 4.1:** The implementation of the `CatalystIBCInterface` constructor.

The `IBC_DISPATCHER` variable in the `CatalystIBCInterface` contract is invoked in order to send packets over IBC. However, it is possible for an admin to deploy a `CatalystIBCInterface` with an `IBC_DISPATCHER` set to address 0, which will cause all IBC communications to revert.

**Impact** If a pool is deployed with a `CatalystIBCInterface` that has an `IBC_DISPATCHER` set to the zero address, then any cross-chain swaps performed through the pool will revert. Since there is no functionality to change the `_chainInterface` of a pool, cross-chain swaps for such a pool will always revert.

**Recommendation** Insert `require(IBCDispatcher_ != address(0))` at the top of the constructor.

**Developer Response** Recommendation implemented.

 

### 4.1.8 V-CAT-VUL-008: Frontrunners can use deposit/withdrawal to extract funds after large liquidity swaps

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 53c5518 |
| **Type** | Frontrunning | **Status** | Intended Behavior |
| **Files** | `SwapPoolCommon.sol` | | |
| **Functions** | sendLiquidityAck(), _releaseLiquidityEscrow() | | |

When a sendLiquidity is ack-ed, `_escrowedPoolTokens` is decreased by the `escrowAmount`. For large `escrowAmounts`, this can cause a noticeable and predictable change in the value of pool tokens (ignoring other intermediate liquidity swaps between the send and the ack for simplicity).

A front-running adversary may use this information to deposit right before a liquidity swap, and withdraw immediately after it is acknowledged. Since the value of the held pool tokens will likely have increased in value, the adversary may withdraw immediately after the ack for a profit.

**Impact** If the liquidity swap changes the value of the pool tokens by a factor of more than `_poolFee`, a front-running adversary can take funds from the pool by depositing before the swap, and withdrawing once it is completed.

For example, consider two pools each with 1000 each of token A, B, C and 1000 pool tokens (the following numbers come from running a `brownie` script).

1. Bob deposits 250 each of tokens A, B, and C into pool 1 and pool 2, receiving 250 pool tokens. Pool 1 now has 1250 each of token A, B, and C, and 1250 pool tokens in total.
2. Alice is front-running, and notices that Bob is about to perform a liquidity swap to pool 2.
3. Alice also deposits 250 token As, Bs, and Cs into pool 1, receiving 250 pool tokens. Pool 1 now has 1500 each of tokens A, B, and C, and 1500 pool tokens in total.
   a) Bob performs a cross-chain liquidity swap with 125 pool tokens, and it is acknowledged. Bob now has 125 pool tokens (of 1375 total) in pool 1, and 363 pool tokens in pool 2.
4. Alice now owns 250 of the 1375 remaining pool tokens in pool 1, or 18% of the 1500 token As in the pool. She withdraws them to receive 272 each of tokens A, B, and C, for a profit of 22 of each token (ignoring pool fees).

Before:

| | Pool 1 Tokens | Value | Pool 2 Tokens | Value |
|---|---|---|---|---|
| Bob | 250 | 250 (each of A, B, C) | 250 | 250 (each of A, B, C) |
| Alice | 250 | 250 (each of A, B, C) | 0 | 0 |

After:

| | Pool 1 Tokens | Value | Pool 2 Tokens | Value |
|---|---|---|---|---|
| Bob | 125 | 136 (each of A, B, C) | 363 | 333 (each of A, B, C) |
| Alice | 250 | 272 (each of A, B, C) | 0 | 0 |

When Alice withdraws all of her money, she will have a profit of 22 of each token in A, B, C.

**Recommendation** Require the sender to provide funds on the receiving chain accounting for the transferred units. Release an equivalent amount of units on the receiving chain once this is complete.

**Developer Response** The developers indicated that this is expected behavior. To guard against arbitragers, Bob can set the `minOut` parameter in the `sendLiquidity()` call to ensure that he gets the intended amount of pool tokens on the receiving chain. If Bob does not get the `minOut` amount of pool tokens, then the cross-chain swap will revert. Alice will then have gained nothing in exchange for paying the governance and gas fees for depositing and withdrawing.

See related issue V-CAT-VUL-003.

 

### 4.1.9 V-CAT-VUL-009: Fee collection occurs before funds transfer

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 53c5518 |
| **Type** | Denial of Service | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol SwapPoolAmplified.sol` | | |
| **Functions** | sendAsset() | | |

In `sendAsset`, the governance fee is collected before funds are transferred to the pool. This involves sending the governance fee from the pool to the pool's factory owner. In the case of a large swap from an asset which the pool has little of, the pool may not have enough of the input asset to cover the governance fee, leading to a revert. This could prevent the pool from processing the transaction.

```
// Governance Fee
_collectGovernanceFee(fromAsset, fee);

// Collect the tokens from the user.
ERC20(fromAsset).safeTransferFrom(msg.sender, address(this), amount);
```

**Snippet 4.2:** The affected lines in `sendAsset()`

```
function _collectGovernanceFee(address asset, uint256 poolFeeAmount) internal {

    uint256 governanceFeeShare = _governanceFeeShare;

    if (governanceFeeShare != 0) {
        uint256 governanceFeeAmount = FixedPointMathLib.mulWadDown(poolFeeAmount,
    governanceFeeShare);
        ERC20(asset).safeTransfer(factoryOwner(), governanceFeeAmount);
    }
}
```

**Snippet 4.3:** The implementation of `_collectGovernanceFee()`, as defined in `SwapPoolCommon.sol`

**Impact** Large swaps on assets which the pool has little of are very beneficial to the pool. In this extreme case, the opportunity is lost because the transaction will revert. The user can work around this issue by issuing two separate transactions, but this is annoying for the user.

**Recommendation** We believe this ordering was chosen to protect against reentrancy attacks. We recommend marking `sendAsset` (as well as the other off-chain transfer functions) as `nonReentrant` and collecting the fee after the transfer.

**Developer Response** The developers indeed said that the ordering was explicitly chosen to prevent reentrancy attacks. They have implemented the recommendation.

### 4.1.10 V-CAT-VUL-010: _usedUnitCapacity can be larger than _maxUnitCapacity in SwapPoolAmplified

| Severity | Low | Commit | 6f066be |
|---|---|---|---|
| Type | Logic Error | Status | Fixed |
| Files | `SwapPoolAmplified.sol` | | |
| Functions | N/A | | |

`SwapPoolAmplified` assumes that the property `_usedUnitCapacity <= _maxUnitCapacity` holds in order to safely perform some computations in `unchecked` blocks. However, there are sequences of transactions which result in `_usedUnitCapacity > _maxUnitCapacity`.

For example, the following sequence of transactions leaves `_usedUnitCapacity` at 15000 and `_maxUnitCapacity` at 14909.

```
Initial State:
Pool[0] balances: [10000, 10000, 10000]
Pool[1] balances: [10000, 10000, 10000]
Pool[0]._maxUnitCapacity = 30000
Pool[0]._usedUnitCapacity = 0
Pool[1]._maxUnitCapacity = 30000
Pool[1]._usedUnitCapacity = 0

        Attacker sends [8000, 4967, 4967] into pools[0] to be
        swapped for each token of pools[1].
        The sendAsset is then ack'ed.

Pool[0] balances: [18000, 14967, 14967]
Pool[1] balances: [3620, 5690, 5690]
Pool[0]._maxUnitCapacity = 47934
Pool[0]._usedUnitCapacity = 0
Pool[1]._maxUnitCapacity = 15000
Pool[1]._usedUnitCapacity = 15000

        Now some arbitraging happens

Swapping 690.0 of token 0 for token 1
Swapping 690.0 of token 0 for token 2
Pool[0] balances: [18000, 14967, 14967]
Pool[1] balances: [5000, 4937, 4972]
Pool[0]._maxUnitCapacity = 47934
Pool[0]._usedUnitCapacity = 0
Pool[1]._maxUnitCapacity = 14909
Pool[1]._usedUnitCapacity = 15000

Terminating local RPC client...
```

**Impact** Since `_usedUnitCapacity` is not always less than the current amount of (weighted) assets, it is possible for `_usedUnitCapacity` to be greater than `_maxUnitCapacity`.

This means that an unchecked block in `depositMixed()` may not actually be safe:

```
_maxUnitCapacity += assetDepositSum;
// Short term decrease the security limit by the amount deposited.
unchecked {
  // _usedUnitCapacity < _maxUnitCapacity => _usedUnitCapacity + assetDepositSum <
    _maxUnitCapacity + assetDepositSum
  _usedUnitCapacity += assetDepositSum;
}
```

**Snippet 4.4:** Snippet from `depositMixed()`

**Recommendation** Remove the unchecked block.

**Developer Response** The recommendation has been applied.

### 4.1.11 V-CAT-VUL-011: _unitTracker can be greater than un-escrowed weighted balance sum

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 6f066be |
| **Type** | Denial of Service | **Status** | Fixed |
| **Files** | `SwapPoolAmplified.sol` | | |
| **Functions** | withdrawAll(), withdrawMixed(), sendLiquidity() | | |

Although it is the case that the sum of the (amplified) weighted asset balances is greater than than the `_unitTracker`, this is not always the case for un-escrowed balances.

```
Pool weights: [1, 1, 1]
Amplification: 0.01

Pools[0] balances: [1100, 1100, 1100]
Pools[0] has 1.1000 total pool tokens
Pools[0]._unitTracker = 0.0000
Pools[0] sum of amplified unescrowed bals: 3076.805286045988

        The attacker puts down a large bogus sendAsset, sending
        [1155, 1155, 1155] funds on each token which they know will fail
        (e.g. by setting minOut too high, or providing
        an ICatalystReceiver which always reverts)

Pools[0] balances: [2255, 2255, 2255]
Pools[0] has 1.1000 total pool tokens
Pools[0]._unitTracker = 3185.5303
Pools[0] sum of amplified unescrowed bals: 3076.805286045988
```

**Impact** The below unchecked block may underflow when the `weightedAssetBalanceSum` is computed using un-escrowed balances (i.e. in `withdrawAll()`, `withdrawMixed()`, and `sendLiquidity()`).

```
unchecked {
    // weightedAssetBalanceSum - _unitTracker can overflow for negative _unitTracker. The result will
    // be correct once it is casted to uint256.
    walpha_0_ampped = uint256(weightedAssetBalanceSum - _unitTracker) / it;   // By design, weightedAssetBalanceSum > _unitTracker
}
```

**Snippet 4.5:** Common snippet used to compute `walpha_0_ampped`

This may lead to denial of service for withdrawers, since the unexpectedly large reference balance leads to issues in later computations. This is exhibited in the following extending example:

```
[catalyst] ~/Veridise/audits/catalyst/evm % brownie run veridise/amp_unit_tracker_unchecked.py main --silent --network development
Brownie v1.19.3 - Python development framework for Ethereum
```

 

```

Initial State:
Pool weights: [1, 1, 1]
Amplification: 0.01
Pools[0] balances: [1000, 1000, 1000]
Pools[0] has 1.0000 total pool tokens
Pools[0]._unitTracker = 0.0000
Pools[0] sum of amplified unescrowed bals: 2799.762902390973

        An innocent bystander deposits [100, 100, 100] receiving 0.1000 tokens

Pools[0] balances: [1100, 1100, 1100]
Pools[0] has 1.1000 total pool tokens
Pools[0]._unitTracker = 0.0000
Pools[0] sum of amplified unescrowed bals: 3076.805286045988

        The attacker puts down a large bogus sendAsset, sending
        [1155, 1155, 1155] funds on each token which they know will fail
        (e.g. by setting minOut too high, or providing
        an ICatalystReceiver which always reverts)

Pools[0] balances: [2255, 2255, 2255]
Pools[0] has 1.1000 total pool tokens
Pools[0]._unitTracker = 3185.5303
Pools[0] sum of amplified unescrowed bals: 3076.805286045988

        Now the initial depositer tries to withdraw a 1e-18 pool token

Transaction sent: 0x8a5cae6a6dc9a18e67c0364562fa122ff1a4ec14fb9b72a9c80b2d7052fcaf5d
revert: Integer overflow
Terminating local RPC client...
```

**Recommendation** The balance0 computation should be performed with a separate `_unitTracker` value to account for cross-chain sends which have not yet been ack'ed.

**Developer Response** The balance0 computation scheme has been modified to use the total balance, not just the un-escrowed balance. This also maintains its relation with `_unitTracker` more correctly.

### 4.1.12 V-CAT-VUL-012: Unchecked block around pool tokens escrow increase

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 6f066be |
| **Type** | Logic Error | **Status** | Fixed |
| **Files** | `SwapPoolAmplified.sol` | | |
| **Functions** | sendLiquidity() | | |

The addition in the following unchecked block may silently overflow.

```
// Escrow the pool tokens
require(_escrowedPoolTokensFor[sendLiquidityHash] == address(0));
_escrowedPoolTokensFor[sendLiquidityHash] = fallbackUser;
unchecked {
    _escrowedPoolTokens += poolTokens;
}
```

**Snippet 4.6:** Snippet from `sendLiquidity()`

**Impact** The escrowed pool tokens could change from a very large number to near zero in the case of an overflow.

**Recommendation** Remove the `unchecked` block.

**Developer Response** Recommendation applied.

 

### 4.1.13 V-CAT-VUL-013: Unchecked addition of weighted escrow amount

| | | | |
|---|---|---|---|
| **Severity** | Low | **Commit** | 55ac118 |
| **Type** | Logic Error | **Status** | Fixed |
| **Files** | SwapPoolAmplified.sol | | |
| **Functions** | sendAssetAck() | | |

In `sendAssetAck()`, the (weighted) escrow amount is added to `_maxUnitCapacity` inside of an unchecked block.

```
function sendAssetAck(bytes32 toAccount, uint256 U, uint256 escrowAmount, address
    escrowToken, uint32 blockNumberMod) public override {
    // Execute common escrow logic.
    super.sendAssetAck(toAccount, U, escrowAmount, escrowToken, blockNumberMod);

    // ...
    unchecked {
        // ...
        _maxUnitCapacity += escrowAmount * _weight[escrowToken];  // Does not
    overflow, since weight times balance of the pool doesn't overflow.
    }
}
```

**Snippet 4.7:** Snippet from `sendAssetAck`

However, at this point it is possible that we have not computed the weighted sum of the pool asset balances. In particular, this sum is not computed in the `sendAsset()` function.

**Impact** This computation may silently overflow, leading to denial of service (since `_maxUnitCapacity` would be very small).

**Recommendation** Perform the computation outside of the unchecked block.

**Developer Response** If `sendAssetAck` fails, then funds can be locked, so reverting on an overflow would cause more harm than the overflow itself.

**Updated Recommendation** To both avoid overflow and avoid this call from failing, check whether the addition will overflow, and set `_maxUnitCapacity` to `uint256::MAX` in the overflow case. Note that this will also require removing several unchecked blocks of the form

```
unchecked {
    _maxUnitCapacity -= ...
}
```

since it may no longer be the case that `_maxUnitCapacity` is greater than the sum of the weighted asset balances.

**Updated Developer Response** Recommendation implemented.

 

### 4.1.14 V-CAT-VUL-014: Use of hard-coded value for decimals

| Severity | Warning | Commit | 53c5518 |
|---|---|---|---|
| Type | Maintainability | Status | Intended Behavior |
| Files | SwapPoolCommon.sol | | |
| Functions | _setPoolFee() | | |

`_setPoolFee()` checks that the fee is less than 100% by checking against `1e18`,

```
require(fee <= 1e18);  // dev: PoolFee is maximum 100%.
```

instead of using the `DECIMALS` field.

**Impact** If `DECIMALS` is modified, then this require statement will no longer check that the fee is at most 100%.

**Recommendation** Replace `1e18` with `10**DECIMALS`.

**Developer Response** All internal computations and units use a fixed value of 18 for DECIMALS due to use of the Solmate library, which assumes 18 decimals of precision.

### 4.1.15 V-CAT-VUL-015: Require statement re-implements modifier

| Severity | Warning | Commit | 53c5518 |
|---|---|---|---|
| Type | Maintainability | Status | Fixed |
| Files | SwapPoolCommon.sol | | |
| Functions | setFeeAdministrator() | | |

`setFeeAdministrator()` uses a require statement to ensure that its caller is the factory owner, rather than using the equivalent modifier `onlyFactoryOwner` . Second, `onlyFactoryOwner` directly calls `CatalystSwapPoolFactory` to retrieve the factory owner, rather than using the helper method `factoryOwner()` which abstracts over that functionality.

```
function setFeeAdministrator(address administrator) public override {
    require(msg.sender == factoryOwner());    // dev: Only factory owner
```

**Snippet 4.8:** Location in `setFeeAdministrator` with the `require` statement.

```
modifier onlyFactoryOwner() {
  require(msg.sender == CatalystSwapPoolFactory(FACTORY).owner());
  _;
}
```

**Snippet 4.9:** Definition of `onlyFactoryOwner()`

```
function factoryOwner() public view override returns (address) {
    return CatalystSwapPoolFactory(FACTORY).owner();
}
```

**Snippet 4.10:** Definition of `factoryOwner()`

#### Impact

- If the definition of `onlyFactoryOwner` is updated, the check in `setFeeAdministrator` will not be updated. A developer may forget to update `setFeeAdministrator`, which could introduce an access control issue.
- Similarly, if the definition of `factoryOwner()` is updated, then `onlyFactoryOwner` will become inconsistent with the check in `setFeeAdministrator`.

**Recommendation** Use the `onlyFactoryOwner` modifier in place of the `require` in `setFeeAdministrator()`, and use `factoryOwner()` in `onlyFactoryOwner()` instead of directly retrieving the `owner()` from `CatalystSwapPoolFactory`.

**Developer Response** Recommendation implemented.

 

### 4.1.16 V-CAT-VUL-016: Confusing naming of packet handling functions

| Severity | Warning | Commit | 53c5518 |
|---|---|---|---|
| Type | Maintainability | Status | Intended Behavior |
| Files | interfaces/ICatalystV1PoolAckTimeout.sol | | |
| Functions | See description | | |

The `SwapPoolCommon` contract implements the `ICatalystV1PoolAckTimeout` interface, which specifies methods for handling ack and timeout packets over IBC. These methods are:

- `sendAssetAck`
- `sendAssetTimeout`
- `sendLiquidityAck`
- `sendLiquidityTimeout`

However, the above methods are called when a packet is **received**, which is confusing.

**Recommendation** The developer should rename the above methods to be prefixed with `receive` to avoid confusion and improve maintainability.

**Developer Response** The methods are named with the convention “message name” “ack/-timeout”. For example, “sendAsset Ack”.

### 4.1.17 V-CAT-VUL-017: Swap pool methods missing override specifiers

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | b3b5a17 |
| **Type** | Maintainability | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol`, `SwapPoolAmplified.sol` | | |
| **Functions** | See description | | |

The `CatalystSwapPoolVolatile` and `CatalystSwapPoolAmplified` contracts implement the following interface methods:

- `ICatalystV1PoolPermissionless.localSwap`
- `ICatalystV1PoolPermissionless.sendAsset`
- `ICatalystV1PoolPermissionless.receiveAsset`
- `ICatalystV1PoolPermissionless.sendLiquidity`
- `ICatalystV1PoolPermissionless.receiveLiquidity`
- `ICatalystV1PoolPermissionless.depositMixed`
- `ICatalystV1PoolPermissionless.withdrawMixed`
- `ICatalystV1PoolPermissionless.withdrawAll`

However, none of these methods are marked with the `override` keyword.

**Impact** Marking the above methods with the `override` keyword will improve readability. Furthermore, this will help prevent errors when a developer removes an interface method with the intention to remove all implementations as well: the Solidity compiler will raise an error if the developer forgets to remove an implementation.

**Recommendation** The developer should mark the implementations of the above methods with the `override` keyword.

**Developer Response** The recommendation has been applied.

 

### 4.1.18 V-CAT-VUL-018: Move initialization array parameter checks to factory

| Severity | Warning | Commit | b3b5a17 |
|---|---|---|---|
| Type | Data Validation | Status | Fixed |
| Files | SwapPoolFactory.sol,SwapPoolVolatile.sol,SwapPoolAmplified.sol | | |
| Functions | deploy_swappool(), initializeSwapCurves() | | |

The `SwapPoolFactory.deploy_swappool()` method accepts three arrays as parameters: `assets`, `init_balances`, and `weights`. It is assumed that they are of the same length, but the checks are performed in the swap pool implementation's `ICatalystV1Pool.initializeSwapCurves()` method, rather than in the factory.

**Recommendation** The developers should add the following checks to the beginning of `deploy_swappool()` and remove them from the implementations of `initializeSwapCurves()`. This will help clarify the shared assumptions about the different swap pool types.

```
require(assets.length > 0);
require(weights.length == assets.length);
```

**Developer Response** The recommendation has been applied.

### 4.1.19 V-CAT-VUL-019: IBC packet sender needs stricter validation

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | b3b5a17 |
| **Type** | Data Validation | **Status** | Acknowledged |
| **Files** | `CatalystIBCInterface.sol` | | |
| **Functions** | onAcknowledgementPacket(), onTimeoutPacket(), onRecvPacket() | | |

Currently, the functions in `CatalystIBCInterface` which receive a message determine the sender by reading from the packet data. For instance,

```
function onAcknowledgementPacket(IbcPacket calldata packet) external {
    // ...
    bytes calldata data = packet.data;

    bytes1 context = data[CONTEXT_POS];
    address fromPool = abi.decode(data[ FROM_POOL_START : FROM_POOL_END ], (address))
    ;
    // ...
}
```

**Snippet 4.11:** Code snippet from `onAcknowledgementPacket`

Depending on the permissions, it is possible for anyone to send the same data packet to a contract using IBC.

**Impact** An adversary could compute the sendAsset/sendLiquidity packet which some pool `P0` would send to another. Then, that adversary can send the packet to a target pool `P1` which is connected to `P0`, causing the `sendAsset` or `sendLiquidity` callbacks to occur without having to put down any funds.

**Recommendation** Sender information should be gathered directly from the IBC protocol whenever possible. For instance, the developers should rely on IBC dynamic capability stores to ensure that pool-to-pool (e.g. `P0` to `P1`) communications occur along a port which is owned by the sending pool (e.g. `P0`).

**Developer Response** The developers indicated that the API that are using for IBC communications are still a work-in-progress, so they are currently making assumptions about the safety of the IBC light client they are interacting with. They will add stricter validation of the IBC packets when the API they are using is completed.

 

### 4.1.20 V-CAT-VUL-020: Amplified calcSendAsset reverts on 0 weight instead of returning 0

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | 6f066be |
| **Type** | Logic Error | **Status** | Fixed |
| **Files** | SwapPoolAmplified.sol | | |
| **Functions** | calcSendAsset() | | |

`_calcPriceCurveArea` and `calcSendAsset` both indicate that they return 0 when used on a token which is not in the pool, however they actually revert.

```
/**
 * @notice Computes the return of SendAsset.
 * @dev Returns 0 if from is not a token in the pool
 ...
 */
function calcSendAsset(address fromAsset, uint256 amount) public view returns (uint256) {
  // ...

    // If a token is not part of the pool, W is 0. This returns 0 since
    // 0^p = 0.
  uint256 U = _calcPriceCurveArea(amount, A, W, _oneMinusAmp);

    // ...
```

**Snippet 4.12:** Snippet from `calcSendAsset`.

`_calcPriceCurveArea` computes `0^p` using `FixedPointMathLib.powWad`. However, `FixedPointMathLib` computes `x**y` as `exp(ln(x) * y)`. Consequently, any values for `x` not in the range of the natural logarithm are rejected.

Implementation of `FixedPointMathLib.powWad` and snippet from `FixedPointMathLib.lnWad`

```
function powWad(int256 x, int256 y) internal pure returns (int256) {
    // Equivalent to x to the power of y because x ** y = (e ** ln(x)) ** y = e ** (ln(x) * y)
    return expWad((lnWad(x) * y) / int256(WAD)); // Using ln(x) means x must be greater than 0.
}
```

```
function lnWad(int256 x) internal pure returns (int256 r) {
  unchecked {
      require(x > 0, "UNDEFINED");
          // ...
```

**Impact** Any transaction expecting `calcSendAsset` to return 0 when `fromAsset` is not in the pool will instead be reverted.

**Recommendation** Change the documentation, or handle the 0 case separately.

**Developer Response** It used to be 0 but code has changed, so documentation was out of date. The documentation has been updated accordingly.

**Updated Developer Response** After viewing issue V-CAT-VUL-006, the zero case is now handled separately.

 

### 4.1.21 V-CAT-VUL-021: Gas savings by using calldata

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 53c5518 |
| **Type** | Gas Optimization | **Status** | Acknowledged |
| **Files** | SwapPoolFactory.sol | | |
| **Functions** | deploy_swappool() | | |

The function `deploy_swappool` marks a few arguments as `memory` which could instead be marked as `calldata` to save gas.

```
function deploy_swappool(
        address poolTemplate,
        address[] memory assets,
        uint256[] memory init_balances,
        uint256[] memory weights,
        uint256 amp,
        uint256 poolFee,
        string memory name,
        string memory symbol,
        address chainInterface
    ) external returns (address) {
```

**Impact** Using `memory` takes up slightly more gas.

**Recommendation** Switch the `assets`, `init_balances`, `weights`, `name`, and `symbol` arguments from `memory` to `calldata`.

**Developer Response** There are several places in the code where `memory` is used instead of `calldata` as the code would otherwise exceed the stack size. The developers plan to adjust the `solc` compilation flags to avoid stack size issues, and then they will use calldata where appropriate.

 

#### 4.1.22 V-CAT-VUL-022: Reimplementation of FixedPointMathLib.mulWadDown

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 53c5518 |
| **Type** | Maintainability | **Status** | Fixed |
| **Files** | `SwapPoolVolatile.sol` | | |
| **Functions** | receiveLiquidity() | | |

The variable `poolTokens` is computed by multiplying two `uint256`s and dividing by `FixedPointMathLib.WAD`, which is the same functionality as `FixedPointMathLib.mulWadDown`. The latter function invokes the gas-optimized function `FixedPointMathLib.mulDivDown`.

```
// On totalSupply. Do not add escrow amount, as higher amount results in a larger
    return.
uint256 poolTokens = (_calcPriceCurveLimitShare(U, wsum) * totalSupply)/
    FixedPointMathLib.WAD;
```

**Snippet 4.13:** The location in `receiveLiquidity()` that performs the calculation

```
function mulWadDown(uint256 x, uint256 y) internal pure returns (uint256) {
  return mulDivDown(x, y, WAD); // Equivalent to (x * y) / WAD rounded down.
}
```

**Snippet 4.14:** The implementation of `FixedPointMathLib.mulWadDown`

**Impact** Switching to `FixedPointMathLib.mulWadDown` would increase consistency with other parts of the code, slightly increase clarity, and possibly cause a slight decrease in gas costs.

**Recommendation** Replace the computation from the code snippet with a call to `FixedPointMathLib.mulWadDown`.

**Developer Response** Fixed in PR. Several other instances were found and one which uses unchecked math to save gas was skipped.

### 4.1.23 V-CAT-VUL-023: Duplicated functionality in initializeSwapCurves

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 6f066be |
| **Type** | Maintainability | **Status** | Acknowledged |
| **Files** | `SwapPoolVolatile.sol`, `SwapPoolAmplified.sol` | | |
| **Functions** | initializeSwapCurves() | | |

Both `SwapPoolVolatile` and `SwapPoolAmplified` contain almost identical code in the function `initializeSwapCurves`.

```
uint256[] memory initialBalances = new uint256[](MAX_ASSETS);
uint256 maxUnitCapacity = 0;
for (uint256 it; it < assets.length;) {

    address tokenAddress = assets[it];
    _tokenIndexing[it] = tokenAddress;

    uint256 weight = weights[it];
    require(weight != 0);
    _weight[tokenAddress] = weight;

    uint256 balanceOfSelf = ERC20(tokenAddress).balanceOf(address(this));
    require(balanceOfSelf != 0);
    initialBalances[it] = balanceOfSelf;

    maxUnitCapacity += weight;
        // AMPLIFIED VERSION: maxUnitCapacity += weight * balanceOfSelf

    unchecked {
        it++;
    }
}

_maxUnitCapacity = maxUnitCapacity * FixedPointMathLib.LN2;
// AMPLIFIED VERSION: _maxUnitCapacity

_mint(depositor, INITIAL_MINT_AMOUNT);

emit Deposit(depositor, INITIAL_MINT_AMOUNT, initialBalances);
```

**Snippet 4.15:** Code from `initializeSwapCurves` which appears in both swap pools, with differences shown in comments.

**Impact** Changes/bug fixes must be made in both contracts. Further, this code is intended to be executed exactly once during setup, but is outside of the initializing function, which requires extra checking on the part of the contract.

**Recommendation** Move the shared functionality into a shared function. Ideally, into the `setup` function so that it is protected by the OpenZeppelin `Initializer` trait. Note that `initializingSwapCurves` is invoked immediately after `setup` in `SwapPoolFactory`, so the deployment process should be nearly identical.

We further recommend replacing the `initializingSwapCurves` function with an overriden `setup` function which invokes `SwapPoolCommon.setup` before doing pool-specific setup. This leverages the `onlyInitializing` modifier in the parent contract, ensuring that all initialization actions are performed at most once through the OpenZeppelin API, and eliminating the extra checks required at the beginning of `initializingSwapCurves`.

**Developer Response** This split is due to the stack limit.

 

### 4.1.24 V-CAT-VUL-024: Out-of-date documentation: variable naming

| Severity | Info | Commit | 6f066be |
|---|---|---|---|
| Type | Maintainability | Status | Fixed |
| Files | SwapPoolAmplified.sol | | |
| Functions | depositMixed() | | |

The internal documentation says the name `intU` will be used since `U` is declared as `int256` instead of `uint256`. However, the name `U` is used.

```
// There is a Stack too deep issue in a later branch. To counteract this,
// wab is stored short-lived. This requires letting U get negative.
// As such, we define an additional variable called intU which is signed
int256 U;
```

**Snippet 4.16:** Snippet from `SwapPoolAmplified.depositMixed`

**Impact** Future maintainers may assume that `U` is unsigned later on in the code.

**Recommendation** Change the variable name from `U` to `intU`.

**Developer Response** The recommendation has been applied.

### 4.1.25 V-CAT-VUL-025: Code Structure Suggestion: Use Introspection

| **Severity** | Info | **Commit** | 538e6f3 |
|---|---|---|---|
| **Type** | info | **Status** | Acknowledged |
| **Files** | `SwapPoolAmplified.sol`, `SwapPoolVolatile.sol` | | |
| **Functions** | receiveLiquidity(), receiveAsset() | | |

The `ICatalystReceiver.onCatalystCall()` method is invoked on a user-supplied address `dataTarget` without any introspection.

**Impact** The user may supply a contract `dataTarget` which does not implement the interface `ICatalystReceiver`, but does implement a `fallback` function, causing unexpected behavior (e.g. if the `dataTarget` is incorrect but has a `fallback`, the transaction will not revert).

**Recommendation** Use an introspection method (such as the ERC1820 Registry) to check that `dataTarget` implements `ICatalystReceiver` before invoking the method.

**Developer Response** We don't believe the added complexity is worth it. Because of the complexity related to encoding the dataTarget, it won't be set manually. As a result, if there is an error in the dataTarget, it is more likely to be encoded incorrectly and thus point to an address which causes the call to revert.

We do appreciate the info and will add a comment describing this case.

 

# Glossary

**AMM** Automated Market Maker. 1

**Brownie** A testing framework for solidity contracts. See `https://eth-brownie.readthedocs.io/en/stable/toctree.html` for more information . 5

**EIP** Ethereum Improvement Proposal. 2

**Ethereum Improvement Proposal** Peer-reviewed proposals for the Ethereum langauge. Visit `https://eips.ethereum.org` to learn more. 47

**IBC** Inter-Blockchain Communication. 1

**Inter-Blockchain Communication** A protocol which relies on light clients to send and verify the receipt of inter-blockchain messages. See also `https://ibcprotocol.org`. 1, 47

**Liquidity Pool** Crowdsourced pools of digital assets used to facilitate trades between assets.. 1

**OpenZeppelin** A security company which provides many standard implementations of common contract specifications. See `https://www.openzeppelin.com`. 1

**Solidity** The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 1

 