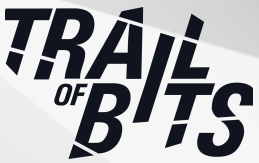


# Reserve Protocol Solidity 4.0.0

## Security Assessment

**June 17, 2025**

*Prepared for:*
**Patrick McKelvy**
Reserve

*Prepared by:* **Samuel Moelius, Benjamin Samuels, and Coriolan Pinhas**

# Table of Contents

# Project Summary

## Contact Information

The following project manager was associated with this project:

**Mary O'Brien**, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

**Benjamin Samuels**, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

**Samuel Moelius**, Consultant
samuel.moelius@trailofbits.com

**Coriolan Pinhas**, Consultant
coriolan.pinhas@trailofbits.com

## Project Timeline

The significant events and milestones of the project are listed below.

| Date | Event |
|---|---|
| **April 15, 2025** | Pre-project kickoff call |
| **April 25, 2025** | Status update meeting #1 |
| **May 2, 2025** | Delivery of first report draft |
| **May 2, 2025** | First report readout meeting |
| **May 20, 2025** | Kickoff call for the second review |
| **May 30, 2025** | Delivery of the second report draft |
| **June 2, 2025** | Second report readout meeting |
| **June 17, 2025** | Delivery of final comprehensive report |

# Executive Summary

## Engagement Overview

Reserve engaged Trail of Bits to review the changes made to its Solidity smart contracts between commits c88e958 and e82bfd6. The changes included the integration of trusted fillers (e.g., CoW Swap), a new method for rebalancing `Folio` contract token balances, and once-a-day inflation of `Folio` contracts' token supplies.

A team of two consultants conducted the review from April 21 to May 2, 2025, for a total of two engineer-weeks of effort. Our testing efforts focused on the newly added features mentioned above. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

The review continued from May 21 to May 30, 2025, for a total of seven engineer-days of effort. This second part of the review was performed on the commit 3870ce8, including minor changes to the `AUCTION_LAUNCHER` price controls.

## Observations and Impact

**First Review**

During the review, we found nothing that would affect `Folio` users' token holdings. However, we did notice deficiencies in the code's readability and test quality. Moreover, we found one issue that could affect off-chain code monitoring `Folio` events.

First, many functions have NatSpec comments. However, many of those comments are out of sync with the functions they describe; for example, the comments omit certain function parameters (an example is given in appendix B). Many contract and struct fields lack descriptive comments. Few structs have comments explaining their purpose.

Second, the code introduced between commits c88e958 and e82bfd6 is not sufficiently tested. Some newly introduced functions are not tested at all, while some are only partially tested. Critical conditions, such as consecutive rebalances and the correct use of nonces by auctions, are not tested. CoW Swap integration is only partially tested.

Finally, some event-emitting functions can be called repeatedly in a way that would lead to confusing event sequences. For example, `endRebalance` can be called repeatedly, resulting in multiple `RebalanceEnded` events with no intervening `RebalanceStarted` events. Such an event sequence could cause off-chain code monitoring the events to behave incorrectly.

**Second Review**

The second review focused on deeper validation of external protocol integrations, economic mechanisms, and test quality. We identified several areas where behavior under

edge conditions or specific attacker strategies could lead to undesirable outcomes or degraded protocol guarantees.

One area of concern is the validation of off-chain data and order parameters in the integration with the CoW Protocol. For example, we found that the `CowSwapFiller` does not validate the `appData` field, leaving room for abuse through inflated partner fees (TOB-FOLIOTF-3).

We also noted that protocol mechanisms depending on time-sensitive or per-block logic may be susceptible to manipulation. For instance, by carefully timing transactions, attackers can delay contract operations or circumvent intended fee accrual, as we found with Dutch auction denial of service (TOB-FOLIOTF-5). These issues indicate that more resilient economic modeling and tighter timing assumptions may be necessary to prevent adversarial behavior.

In terms of testing, mutation testing using `slither-mutate` showed that many `require` statements are not meaningfully exercised: commenting them out often did not affect test outcomes. This strongly suggests that failure cases, edge conditions, and invalid inputs are not adequately tested, leaving critical logic under-protected. Ensuring that these validations are covered is essential for catching regressions, particularly in complex systems like auctions or share accounting.

## Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Reserve take the following steps:

- **Review NatSpec comments to ensure that they are complete and accurate.** Inaccuracies and omissions in comments make them less effective. Addressing such deficiencies will help maximize the comments' benefit. The `lintspec` tool may help with identifying comments needing revision.
- **Expand the tests of the features introduced between commits c88e958 and e82bfd6 (TOB-FOLIOTF-1).** Tests help expose errors, protect against regressions, and provide a sort of documentation to users. The benefits of a robust test suite cannot be overstated.
- **Adjust event-emitting functions so that repeated calls to a function do not result in multiple events (TOB-FOLIOTF-2).** Doing so will reduce the likelihood of such events causing off-chain code to behave incorrectly.
- **Add fuzz tests to the codebase** to uncover edge cases and potential vulnerabilities that may arise under specific Folio conditions. Fuzzing can help identify unexpected behaviors not covered by standard tests. Medusa, one of our tools, can assist with this process.

In addition to the above, we've prepared several appendices with recommendations to improve the code's long-term maintainability:

- Appendix B contains general, non-security-related recommendations.
- Appendix C contains architectural recommendations.
- Appendix D contains recommendations for checking functions' gas usage to ensure that the functions do not become uncallable.

## Finding Severities and Categories

The following tables provide the number of findings by severity and category.

**EXPOSURE ANALYSIS**

| *Severity* | *Count* |
|---|---|
| High | 0 |
| Medium | 1 |
| Low | 2 |
| Informational | 3 |
| Undetermined | 0 |

**CATEGORY BREAKDOWN**

| *Category* | *Count* |
|---|---|
| Auditing and Logging | 1 |
| Data Validation | 2 |
| Denial of Service | 1 |
| Testing | 1 |
| Timing | 1 |

# Project Goals

The engagement was scoped to provide a security assessment of the code introduced between commits c88e958 and e82bfd6. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can a user receive funds without providing funds in return?
- Can an auction be initiated with a sell-token price of 0?
- Are there ways to extract value from a `Folio` (e.g., by exploiting the new 24-hour fee inflation cycle)?

# Project Targets

The engagement involved 2 reviews and testing of the targets listed below. Note that the first is a diff between two commits: an earlier commit (c88e958) and a later commit (e82bfd6/3870ce8).

### Reserve Folio (diff): First Review

| | |
|---|---|
| Repository | https://github.com/reserve-protocol/reserve-index-dtf |
| Later commit | `e82bfd60b391832640800f32bb41f735a0e1e56f` |
| Earlier commit | `c88e958bb70211bbb73c601382ef9e9b1611f00c` |
| Type | Solidity |
| Platform | Ethereum |

### Reserve Folio (diff): Second Review

| | |
|---|---|
| Repository | https://github.com/reserve-protocol/reserve-index-dtf |
| Later commit | `3870ce825242498c773c767094120b7130590248` |
| Earlier commit | `c88e958bb70211bbb73c601382ef9e9b1611f00c` |
| Type | Solidity |
| Platform | Ethereum |

### 3.0.0 Upgrade Spell

| | |
|---|---|
| Repository | https://github.com/reserve-protocol/reserve-index-dtf |
| Pull request | 124 |
| Commit | `f2067bcc23a8b24671f8a06bab2ad27fe45d67b8` |
| Type | Solidity |
| Platform | Ethereum |

### Reserve Trusted Fillers

| | |
|---|---|
| Repository | https://github.com/reserve-protocol/trusted-fillers |
| Commit | `20ec35af10212eefaa3963cefce7318231c20567` |
| Type | Solidity |
| Platform | Ethereum |

# Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Test coverage review:** We verified that both the `reserve-index-dtf` and `trusted-filler` repositories' tests pass. We also computed the tests' code coverage and looked for gaps.
- **Static analysis:** We ran Slither over the codebases and reviewed the results.
- **Manual review:** We manually reviewed the diff between commits c88e958 and e82bfd6, focusing on the following elements:
    - Trusted fillers
    - Rebalancing
    - Discrete fee inflation

  We also manually reviewed the 3.0.0 upgrade spell.

## Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- CoW Swap integration is only partially tested. Reserve has begun work on a bot for interacting with CoW Swap; however, it is still considered a work in progress. There could be bugs in the interaction of the `reserve-index-dtf` or `trusted-filler` repositories with CoW Swap that would be revealed only by complete end-to-end testing.
- We found no problems with the 3.0.0 upgrade spell. However, we may be failing to anticipate the full set of configurations a `Folio` could be in when it is upgraded. To increase confidence in the upgrade spell's correctness, we recommend thoroughly testing it on fake (i.e., not for production use) `Folio` deployments.

# Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

| Category | Summary | Result |
|---|---|---|
| Arithmetic | In most cases, arithmetic operations are commented with the formulae they are meant to implement, and the units used. Well-established libraries are used for more complicated operations. | **Satisfactory** |
| Auditing | Events are generated in appropriate circumstances and with pertinent information. In some cases, events can be generated in a confusing manner; for example, two `RebalanceStarted` events may have no intervening `RebalanceEnded` event, or two `RebalanceEnded` events may have no intervening `RebalanceStarted` event. However, we do not feel that this warrants lowering the Satisfactory rating. | **Satisfactory** |
| Authentication / Access Controls | Well-established libraries are used for authentication and access controls. We found no problems related to their use. | **Satisfactory** |
| Complexity Management | Functions are of appropriate size and complexity. Most are documented, despite some shortcomings noted below. | **Satisfactory** |
| Cryptography and Key Management | The changes to the codebase do not involve hashing or key management. As a result, this category is not applicable. | **Not Applicable** |
| Decentralization | Certain aspects of a `Folio`'s operation are permissioned. The `DEFAULT_ADMIN_ROLE` has significant control over the Folio contract. | **Moderate** |
| Documentation | Most comments look like they were written with care and | **Moderate** |

| | | |
|---|---|---|
| | with a focus on accuracy. However, several NatSpec comments are out of sync with the functions they describe, reducing the NatSpec comments' usefulness. Many contract and struct fields lack descriptive comments. Few structs have comments explaining their purpose. | |
| Low-Level Manipulation | The only use of assembly is in the `GPv2OrderLib` library, which comes from, and is tested by, the cowprotocol/contracts repository. | **Satisfactory** |
| Testing and Verification | Several functions introduced by the changes are not tested at all, while some are only partially tested. Critical conditions, such as consecutive rebalances and the correct use of nonces by auctions, are not tested. The CoW Swap integration is only partially tested.<br><br>Note that this rating is for the code introduced by the changes. The testing and verification of the `reserve-index-dtf` repository as a whole would likely be rated higher. | **Weak** |
| Transaction Ordering | We found no problems related to transaction ordering. However, we are concerned that the current API's design could allow transaction-ordering-related bugs to be introduced. Appendix C gives architectural recommendations that should help reduce this risk. | **Satisfactory** |

# Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

| ID | Title | Type | Severity |
|---|---|---|---|
| 1 | Insufficient test coverage of new features | Testing | Informational |
| 2 | Functions can be called repeatedly, causing misleading events | Auditing and Logging | Low |
| 3 | Order surplus extraction | Data Validation | Low |
| 4 | DAO can receive slightly lower fees than expected | Data Validation | Informational |
| 5 | Denial of service allows blocking operations during Dutch auctions | Denial of Service | Medium |

# Detailed Findings

### 1. Insufficient test coverage of new features

| Severity: **Informational** | Difficulty: **High** |
|---|---|
| Type: Testing | Finding ID: TOB-FOLIOTF-1 |
| Target: Entire codebase | |

**Description**

Code that was added to `reserve-index-dtf` to support trusted fillers, and rebalancing is not sufficiently tested (figures 1.1–1.3). Several new functions in the `Folio` contract and the `TrustedFillerRegistry` contract are not tested at all, and some functions in the `CowSwapFiller` contract are only partially tested. Insufficient testing of code could allow bugs to go unnoticed.

| Directory | Line Coverage | | | Function Coverage | | |
|---|---|---|---|---|---|---|
| | Rate | Total | Hit | Rate | Total | Hit |
| contracts/ | 93.0 % | 300 | 279 | 91.3 % | 46 | 42 |
| contracts/deployer/ | 100.0 % | 56 | 56 | 100.0 % | 6 | 6 |
| contracts/folio/ | 100.0 % | 85 | 85 | 100.0 % | 20 | 20 |
| contracts/governance/ | 100.0 % | 35 | 35 | 100.0 % | 14 | 14 |
| contracts/spells/31-03-2025/ | 0.0 % | 82 | | 0.0 % | 4 | |
| contracts/staking/ | 100.0 % | 129 | 129 | 100.0 % | 24 | 24 |
| contracts/utils/ | 98.8 % | 80 | 79 | 100.0 % | 10 | 10 |
| script/ | 0.0 % | 55 | | 0.0 % | 5 | |
| test/base/ | 82.1 % | 168 | 138 | 14.3 % | 21 | 3 |
| test/utils/ | 58.8 % | 51 | 30 | 58.3 % | 24 | 14 |
| test/utils/upgrades/ | 100.0 % | 6 | 6 | 100.0 % | 3 | 3 |
| trusted-fillers/contracts/ | 65.0 % | 20 | 13 | 60.0 % | 5 | 3 |
| trusted-fillers/contracts/fillers/cowswap/ | 74.6 % | 67 | 50 | 50.0 % | 10 | 5 |

*Figure 1.1: Combined test coverage of the `reserve-index-dtf` and `trusted-fillers` repositories*

| File | Line Coverage | | | Function Coverage | | |
|---|---|---|---|---|---|---|
| | Rate | Total | Hit | Rate | Total | Hit |
| TrustedFillerRegistry.sol | 65.0 % | 20 | 13 | 60.0 % | 5 | 3 |

*Figure 1.2: Test coverage of the `TrustedFillerRegistry` contract*

| File | Line Coverage | | | Function Coverage | | |
|---|---|---|---|---|---|---|
| | Rate | Total | Hit | Rate | Total | Hit |
| CowSwapFiller.sol | 92.7 % | 41 | 38 | 66.7 % | 6 | 4 |
| GPv2OrderLib.sol | 46.2 % | 26 | 12 | 25.0 % | 4 | 1 |

*Figure 1.3: Test coverage of the `CowSwapFiller` contract and `GPv2OrderLib` library*[1]

The following `Folio` functions are not tested at all:

- `stateChangeActive`
- `setTrustedFillerRegistry`
- `getRebalance`
- `endRebalance`

Note that the lack of testing of `endRebalance` implies that there is no test that `startRebalance` can be called following a call to `endRebalance`. This is a critical condition that the tests should cover.

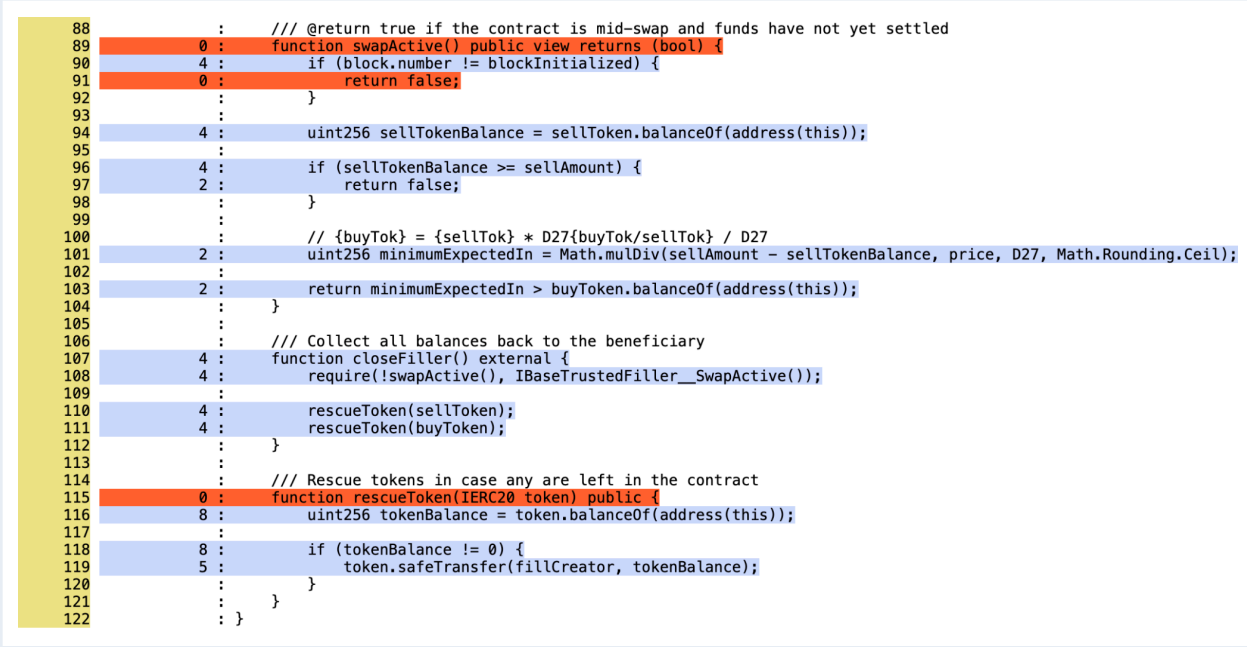
There are also no tests to verify proper handling of nonces. More specifically, each `Folio` deployment holds a `Rebalance` struct with a nonce. `Folio` functions that act on an `Auction` check that its nonce matches the `Rebalance`'s nonce and revert if there is a mismatch. However, there are currently no tests to verify that `Auction`-acting functions revert when expected to.

The following `TrustedFillerRegistry` functions are not tested at all:

- `deprecateTrustedFiller`
- `isAllowed`

The `swapActive` and `rescueToken` functions in the `CowSwapFiller` contract are only partially tested (figure 1.4). First, `swapActive`'s `block.number != blockInitialized` branch is not tested. Second, while `swapActive` and `rescueToken` are called internally, they are not called externally.

[1] The low coverage numbers for the `GPv2OrderLib` library are not concerning. The functions in that library are tested in the cowprotocol/contracts repository.



```
    88            :     /// @return true if the contract is mid-swap and funds have not yet settled
    89          0 :     function swapActive() public view returns (bool) {
    90          4 :         if (block.number != blockInitialized) {
    91          0 :             return false;
    92            :         }
    93            :
    94          4 :         uint256 sellTokenBalance = sellToken.balanceOf(address(this));
    95            :
    96          4 :         if (sellTokenBalance >= sellAmount) {
    97          2 :             return false;
    98            :         }
    99            :
   100            :         // {buyTok} = {sellTok} * D27{buyTok/sellTok} / D27
   101          2 :         uint256 minimumExpectedIn = Math.mulDiv(sellAmount - sellTokenBalance, price, D27, Math.Rounding.Ceil);
   102            :
   103          2 :         return minimumExpectedIn > buyToken.balanceOf(address(this));
   104            :     }
   105            :
   106            :     /// Collect all balances back to the beneficiary
   107          4 :     function closeFiller() external {
   108          4 :         require(!swapActive(), IBaseTrustedFiller__SwapActive());
   109            :
   110          4 :         rescueToken(sellToken);
   111          4 :         rescueToken(buyToken);
   112            :     }
   113            :
   114            :     /// Rescue tokens in case any are left in the contract
   115          0 :     function rescueToken(IERC20 token) public {
   116          8 :         uint256 tokenBalance = token.balanceOf(address(this));
   117            :
   118          8 :         if (tokenBalance != 0) {
   119          5 :             token.safeTransfer(fillCreator, tokenBalance);
   120            :         }
   121            :     }
   122            : }
```

*Figure 1.4: Excerpt of the `CowSwapFiller` contract's test coverage*

Furthermore, while there is a test of `isValidSignature`, there is no test of a complete interaction with the Cow Swap contracts. A function could appear correct when tested in isolation, yet it could fail when integrated into a larger system.

### Exploit Scenario

An exploitable bug is discovered in the `trusted-fillers` code. The bug could have been uncovered through more thorough testing.

### Recommendations

Short term, take the following steps:

- Add tests to the `reserve-index-dtf` or `trusted-fillers` repositories so that 100% statement coverage of the code added to support the new features is achieved.
- Add tests for the scenarios mentioned above:
  - A call to `endRebalance` followed by a call to `startRebalance`
  - Nonce handling (e.g., that `Auctions` with invalid nonces are rejected)
- Develop a `MockCowSwap` contract to simulate the actual CoW Swap contract, and use the mock contract in tests.

Taking these steps will help to ensure that the code added to support the new features is free of bugs.

Long term, regularly review the project's test coverage. If a critical condition is not tested despite 100% statement coverage, add a test for that condition. Doing so will provide further assurance that the code is free of bugs.

### 2. Functions can be called repeatedly, causing misleading events

| Severity: **Low** | Difficulty: **High** |
| --- | --- |
| Type: Auditing and Logging | Finding ID: TOB-FOLIOTF-2 |
| Target: `reserve-index-dtf/contracts/Folio.sol` | |

**Description**

Several event-emitting `Folio` functions can be called repeatedly, without intervening calls to related event-emitting functions. Off-chain code expecting to process multiple kinds of events could behave incorrectly when seeing just one type of event.

The following functions in question are affected:

- `startRebalance` (which emits `RebalanceStarted`) can be called repeatedly without intervening calls to `endRebalance` (which emits `RebalanceEnded`).
- `endRebalance` (which emits `RebalanceEnded`) can be called repeatedly without intervening calls to `startRebalance` (which emits `RebalanceStarted`).
- `closeAuction` (which emits `AuctionClosed`) can be called repeatedly without intervening calls to `openAuction` (which emits `AuctionOpened`).

Note that `openAuction` does not have this problem. That is, multiple calls to `openAuction` require intervening calls to `closeAuction` or `endRebalance`.

Also, note that some of the above functions have side effects besides emitting events. For example, repeated calls to `endRebalance` will result in repeated changes to `rebalance.availableUntil` (figure 2.1), though we could find no way that this would cause harm:

```
794     function endRebalance() external nonReentrant {
        ...
802         emit RebalanceEnded(rebalance.nonce);
803
804         // do not revert, to prevent griefing
805         rebalance.nonce++; // advancing nonce clears auctionEnds
806         rebalance.availableUntil = block.timestamp;
807     }
```

*Figure 2.1: Excerpt of the definition of `endRebalance` (`reserve-index-dtf/contracts/Folio.sol#794–807`)*

Reserve explained that some of the behavior described in this finding is intentional. For example, suppose an auction launcher calls `endRebalance` on a `Folio`. If a governance proposal calls `endRebalance` on the same `Folio`, the proposal's call should not revert because of the auction launcher's call.

### Exploit Scenario

Alice writes a bot that trades against Reserve `Folio`s. Alice's bot expects auction sequences to be bracketed by `RebalanceStarted` and `RebalanceEnded` events, and for each auction to begin with an `AuctionOpened` event. Bo maintains one of the `Folio`s that Alice's bot trades against. Bob calls `startRebalance`, `endRebalance`, or `closeAuction` multiple times (for unknown reasons). Bob's transactions succeed, but the emitted events cause Alice's bot to behave incorrectly. Alice suffers financial loss as a result.

### Recommendations

Short term, adjust `startRebalance` so that one or more consecutive calls to that function emit exactly one `RebalanceStarted` event. Adjust `endRebalance` and `closeAuction` similarly. Eliminate any other undesirable side effects resulting from repeated calls to these functions. Taking these steps will produce more predictable event sequences, reducing the likelihood of off-chain code behaving incorrectly.

Long term, incorporate fuzzing with Medusa into your workflow. Medusa can generate arbitrary call sequences, which could help to expose problems like the one described here.

### 3. Order surplus extraction

| Severity: **Low** | Difficulty: **Medium** |
| --- | --- |
| Type: Data Validation | Finding ID: TOB-FOLIOTF-3 |
| Target: `trusted-fillers/contracts/fillers/cowswap/GPv2OrderLib.sol` | |

#### Description

The `CowSwapFiller` contract's `isValidSignature` function ignores the `appData` field when verifying CowSwap orders. This oversight allows malicious actors to manipulate ERC-1271 orders by setting the `partnerFee` field to a huge amount, allowing the filler to drain surplus.

According to CoW Protocol documentation, the `appData` field in orders can contain encoded partner fee information. When the `CowSwapFiller` implements ERC-1271 signature verification for off-chain CoW orders, it checks various order parameters but does not validate this field. The vulnerability allows an attacker to modify the `appData` to include arbitrarily high partner fees while keeping the signature valid. When these orders are settled by the CoW Protocol, the encoded partner fees are processed, potentially redirecting most or all of the order's surplus to addresses controlled by the attacker.

```
bytes32 appData;
```

*Figure 3.1: `appData` field is not verified in the `isValidSignature` function (`trusted-fillers/contracts/fillers/cowswap/GPv2OrderLib.sol#19`)*

#### Exploit Scenario

An attacker observes a valid off-chain order to fill the `CowSwapFiller` contract. Before this order is submitted to the CoW Protocol, the attacker takes the order details but sets the appData field to include excessive partner fees directed to their address. When the CoW Protocol settlement contract calls the `CowSwapFiller`'s `isValidSignature` function, it passes validation because `appData` is not checked. During settlement, the protocol processes the partner fees encoded in the `appData`, extracting most or all of the order's surplus and sending it to the attacker's address instead of providing it to the user.

#### Recommendations

Short term, implement explicit validation of the `appData` field in the `isValidSignature` function. Additionally, it will validate pre-hook and post-hook functionality, ensuring that the pre-hook properly creates the `CowSwapFiller` and the post-hook closes it.

Long term, review the CoW Protocol documentation regularly to stay informed about known vulnerabilities in custom implementations.

### References

- CoW Protocol documentation

### 4. DAO can receive slightly lower fees than expected

| Severity: **Informational** | Difficulty: **High** |
| --- | --- |
| Type: Data Validation | Finding ID: TOB-FOLIOTF-4 |
| Target: `reserve-index-dtf/contracts/Folio.sol` | |

#### Description

The contract converts the annual percentage fee floor to a per-second rate using an approximation that introduces precision loss.

The issue stems from the `ONE_OVER_YEAR` constant, which represents 1/31536000 (seconds in a standard 365-day year) and is used in the conversion formula:

```
uint256 feeFloor = D18 - MathLib.pow(D18 - daoFeeFloor, ONE_OVER_YEAR);
```

*Figure 4.1: `feeFloor` calculation (`reserve-index-dtf/contracts/Folio.sol#937`)*

This constant is inherently imprecise due to fixed-point arithmetic limitations; here it is rounded down. As a result, the actual enforced fee floor is slightly lower than intended. For example, a configured annual fee floor of 0.15% becomes approximately 0.14999990833183% after conversion.

Additionally, this calculation does not account for leap years, which have 366 days or 31,622,400 seconds instead of 31,536,000 seconds. During leap years, the `ONE_OVER_YEAR` constant is even more inaccurate, as it still uses the number of seconds in a standard year, causing the calculated per-second fee floor to be further off from the expected value. This means that the DAO will receive a higher percentage of fees during leap years compared to regular years for the same `feeFloor` stored variable on the `FolioDAOFeeRegistry` that represents the yearly fee floor.

#### Recommendations

Short term, document this precision loss in both code comments and user-facing documentation to set appropriate expectations for the DAO regarding minimum fees and leap years. Consider adjusting the fee floor value slightly upward to account for the known precision loss if needed.

Long term, consider adding comprehensive documentation for every instance of precision loss throughout the protocol to ensure all users interacting with the system are aware of these limitations.

### 5. Denial of service allows blocking operations during Dutch auctions

| Severity: **Medium** | Difficulty: **High** |
| --- | --- |
| Type: Denial of Service | Finding ID: TOB-FOLIOTF-5 |
| Target: `reserve-index-dtf/contracts/Folio.sol` | |

**Description**

A malicious actor can trigger a denial of service (DoS) on the sync modifier, allowing them to temporarily block all operations.

The sync modifier calls the poke function, which attempts to close any active trusted filler. However, if a malicious actor creates a CowSwap filler at the beginning of the block and ensures that it remains in the "active" state by just swapping 1 or a few wei as a valid partial swap, this will cause all functions with the sync modifier to revert.

When leveraged with the Dutch auction mechanism used in the protocol, this can create an incentive for an attacker to repeat this mechanism during a few blocks to get assets at a lower price.

```
/// @return true if the contract is mid-swap and funds have not yet settled
function swapActive() public view returns (bool) {
    if (block.number != blockInitialized) {
        return false;
    }

    uint256 sellTokenBalance = sellToken.balanceOf(address(this));

    if (sellTokenBalance >= sellAmount) {
        return false;
    }

    // {buyTok} = {sellTok} * D27{buyTok/sellTok} / D27
    uint256 minimumExpectedIn = Math.mulDiv(sellAmount - sellTokenBalance, price,
D27, Math.Rounding.Ceil);

    return minimumExpectedIn > buyToken.balanceOf(address(this));
}
```

*Figure 5.1: The `swapActive` function that will return partial swaps for 1 or a few wei (`trusted-fillers/contracts/fillers/cowswap/CowSwapFiller.sol#88–104`)*

### Exploit Scenario

During a Dutch auction where prices decrease over time, a malicious actor performs the following steps:

1. Calls `createTrustedFill` to set up a new trusted fill for the auction at the beginning of the block. The attacker can try to pay MEV to perform this operation at the beginning of the block.
2. Executes a few wei partial swap to make the `swapActive` function return `true`. This prevents any operation from executing for at least the current block.

Since Dutch auction prices decrease over time, the attacker can perform this during a few blocks, causing other users' transactions to revert. After the price has decreased sufficiently, the attacker places their bid at a more favorable price. This creates a financial incentive for the attacker to spend gas on blocking the contract temporarily. Note that if the price of the asset increases during that time, the phenomenon is amplified.

### Recommendations

Short term, perform a verification on `buyAmount` that reverts on swaps fewer than x% of the total minimum `buyAmount` expected. Moreover, consider adding a check on `appData` to ensure that hook operations are performed before and after the swap.

Long term, conduct an economic analysis to ensure that the cost of executing a denial-of-service attack substantially outweighs potential profits from manipulating Dutch auction prices. Consider implementing mechanisms that increase the cost or difficulty of sustained attacks.

# A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

| Vulnerability Categories | |
|---|---|
| **Category** | **Description** |
| **Access Controls** | Insufficient authorization or assessment of rights |
| **Auditing and Logging** | Insufficient auditing of actions or logging of problems |
| **Authentication** | Improper identification of users |
| **Configuration** | Misconfigured servers, devices, or software components |
| **Cryptography** | A breach of system confidentiality or integrity |
| **Data Exposure** | Exposure of sensitive information |
| **Data Validation** | Improper reliance on the structure or values of data |
| **Denial of Service** | A system failure with an availability impact |
| **Error Reporting** | Insecure or insufficient reporting of error conditions |
| **Patching** | Use of an outdated software package or library |
| **Session Management** | Improper identification of authenticated users |
| **Testing** | Insufficient test methodology or test coverage |
| **Timing** | Race conditions or other order-of-operations flaws |
| **Undefined Behavior** | Undefined behavior triggered within the system |

| Severity Levels | |
|---|---|
| **Severity** | **Description** |
| **Informational** | The issue does not pose an immediate risk but is relevant to security best practices. |
| **Undetermined** | The extent of the risk was not determined during this engagement. |
| **Low** | The risk is small or is not one the client has indicated is important. |
| **Medium** | User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. |
| **High** | The flaw could affect numerous users and have serious reputational, legal, or financial implications. |

| Difficulty Levels | |
|---|---|
| **Difficulty** | **Description** |
| **Undetermined** | The difficulty of exploitation was not determined during this engagement. |
| **Low** | The flaw is well known; public tools for its exploitation exist or can be scripted. |
| **Medium** | An attacker must write an exploit or will need in-depth knowledge of the system. |
| **High** | An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. |

# B. Non-Security-Related Recommendations

The following recommendations are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Correct the inaccurate comment in figure B.1.** The highlighted portion should be replaced with, e.g., “return the selector” or “succeed.”

```
1401    // isValidSignature should return true for the correct bid
```

*Figure B.1: Inaccurate comment (`reserve-index-dtf/test/Folio.t.sol#1401`)*

- **Either remove the event in figure B.2, or add a comment explaining why it is unused.** The event was added in a commit between c88e958 and e82bfd6, specifically commit 78a61a0.

```
24    event DustAmountSet(address token, uint256 newDustAmount);
```

*Figure B.2: Unused event (`reserve-index-dtf/contracts/interfaces/IFolio.sol#24`)*

- **Reorder the checks in the code in figure B.3 so that they match the field declaration order in `GPv2OrderLib.Data`.** Specifically, rewrite the code as in figure B.4.

```
71    require(order.sellToken == sellToken, CowSwapFiller__OrderCheckFailed(1)); //
Invalid Sell Token
72    require(order.buyToken == buyToken, CowSwapFiller__OrderCheckFailed(2)); //
Invalid Buy Token
73    require(order.feeAmount == 0, CowSwapFiller__OrderCheckFailed(3)); // Must be
a Limit Order
74    require(order.receiver == address(this), CowSwapFiller__OrderCheckFailed(4));
// Receiver must be self
75    require(order.sellTokenBalance == GPv2OrderLib.BALANCE_ERC20,
CowSwapFiller__OrderCheckFailed(5)); // Must use ERC20 Balance
76    require(order.buyTokenBalance == GPv2OrderLib.BALANCE_ERC20,
CowSwapFiller__OrderCheckFailed(6)); // Must use ERC20 Balance
77    require(order.sellAmount != 0, CowSwapFiller__OrderCheckFailed(7)); // catch
div-by-zero below
```

*Figure B.3: Excerpt of the `isValidSignature` function. The order of these checks does not match the order in which the fields are declared. (`trusted-fillers/contracts/fillers/cowswap/CowSwapFiller.sol#71-77`)*

```
require(order.sellToken == sellToken, CowSwapFiller__OrderCheckFailed(1)); //
Invalid Sell Token
require(order.buyToken == buyToken, CowSwapFiller__OrderCheckFailed(2)); // Invalid
```

```
Buy Token
require(order.receiver == address(this), CowSwapFiller__OrderCheckFailed(4)); //
Receiver must be self
require(order.sellAmount != 0, CowSwapFiller__OrderCheckFailed(7)); // catch
div-by-zero below
require(order.feeAmount == 0, CowSwapFiller__OrderCheckFailed(3)); // Must be a
Limit Order
require(order.sellTokenBalance == GPv2OrderLib.BALANCE_ERC20,
CowSwapFiller__OrderCheckFailed(5)); // Must use ERC20 Balance
require(order.buyTokenBalance == GPv2OrderLib.BALANCE_ERC20,
CowSwapFiller__OrderCheckFailed(6)); // Must use ERC20 Balance
```

*Figure B.4: Proposed rewrite of the code in figure B.3. Note that the `CowSwapFiller__OrderCheckFailed` errors were originally ordered sequentially and thus show how the checks' order has changed.*

- **Replace the comment in figure B.5 with that of figure B.6.** The library called `GPv2OrderLib` in `trusted-fillers` is called `GPv2Order` in `cowprotocol/contracts`, making it somewhat hard to find.

```
 8    /// @dev From https://github.com/cowprotocol/contracts
```

*Figure B.5: Comment that could be made more specific (`trusted-fillers/contracts/fillers/cowswap/GPv2OrderLib.sol#8`)*

```
 8    /// @dev From
https://github.com/cowprotocol/contracts/blob/main/src/contracts/libraries/GPv2Order
.sol
```

*Figure B.6: Proposed rewrite of the comment in figure B.5*

- **Correct NatSpec comments throughout the codebase.** For example, in figure B.7, the NatSpec comment for the bid function omits the `auctionId` parameter. We noticed similar problems in the NatSpec comments of other functions in `Folio.sol` and `AuctionLib.sol`. The `lintspec` tool may be used to identify broken NatSpec comments.

```
 682    /// Bid in an ongoing auction
 683    ///   If withCallback is true, caller must adhere to IBidderCallee interface
and receives a callback
 684    ///   If withCallback is false, caller must have provided an allowance in
advance
 685    /// @dev Callable by anyone
 686    /// @param sellAmount {sellTok} Sell token, the token the bidder receives
 687    /// @param maxBuyAmount {buyTok} Max buy token, the token the bidder
provides
 688    /// @param withCallback If true, caller must adhere to IBidderCallee
interface and transfers tokens via callback
 689    /// @param data Arbitrary data to pass to the callback
```

```
690     /// @return boughtAmt {buyTok} The amount bidder receives
691     function bid(
692         uint256 auctionId,
693         uint256 sellAmount,
694         uint256 maxBuyAmount,
695         bool withCallback,
696         bytes calldata data
697     ) external nonReentrant notDeprecated sync returns (uint256 boughtAmt) {
```

*Figure B.7: Function declaration whose NatSpec comment omits a parameter (reserve-index-dtf/contracts/Folio.sol#682–697)*

- **Expand the comments preceding the code in figures B.8 and B.9.** The formulae are unintuitive, and the existing comments are obscure. For example, nowhere in the code is UoA defined.

```
584     // D27{buyTok/sellTok} = D27 * D27{UoA/sellTok} / D27{UoA/buyTok}
585     uint256 oldStartPrice = (D27 * sellDetails.prices.high +
buyDetails.prices.low - 1) / buyDetails.prices.low;
586     uint256 oldEndPrice = (D27 * sellDetails.prices.low + buyDetails.prices.high
- 1) / buyDetails.prices.high;
```

*Figure B.8: Unintuitive formulae with an obscure comment (reserve-index-dtf/contracts/Folio.sol#584–586)*

```
631     // D27{buyTok/sellTok} = D27 * D27{UoA/sellTok} / D27{UoA/buyTok}
632     uint256 startPrice = (D27 * sellDetails.prices.high + buyDetails.prices.low
- 1) / buyDetails.prices.low;
633     uint256 endPrice = (D27 * sellDetails.prices.low + buyDetails.prices.high -
1) / buyDetails.prices.high;
```

*Figure B.9: Another instance of the formulae and comment from figure B.8 (reserve-index-dtf/contracts/Folio.sol#631–633)*

- **Replace the test in B.10 with a Forge fuzz test.** A comment in the test's body suggests dissatisfaction with the test. Replacing the test with a Forge fuzz test will gain the benefits of Forge's fuzzing engine. For the long term, consider a more sophisticated fuzzing strategy using Medusa.

```
80    it("should produce auctions across a variety of setups", () => {
```

*Figure B.10: Test that should be made into a fuzz test (reserve-index-dtf/rebalancing/helpers/getBasket.test.ts#80)*

- **Correct the grammar of the comment in figure B.11.**

```
12    * This spell a Folio to upgrade to 3.0.0.
```

*Figure B.11: Comment with incorrect grammar (`reserve-index-dtf/contracts/spells/upgrades/UpgradeSpell_3_0_0.sol#12`)*

# C. Architectural Recommendations

This appendix contains recommendations regarding changes made to the APIs between commits c88e958 and e82bfd6. The recommendations specifically concern how `Rebalance` nonces are handled, and how auction end times are referred to. First, we explain why these aspects of the APIs warrant change.

## Rebalance Nonce Handling

Each `Folio` contract contains a `Rebalance` struct that stores a nonce. Intuitively, the nonce identifies a rebalancing operation. However, the nonce is incremented within two functions: `startRebalance` and `endRebalance` (figures C.1 and C.2). Thus, in a sequence of alternating calls between the two functions (i.e., `startRebalance`, `endRebalance`, `startRebalance`, `endRebalance`, ...), twice as many nonces will be consumed than necessary, because the nonce values between the calls to `endRebalance` and `startRebalance` will effectively be unused.

```
485    function startRebalance(
486        address[] calldata newTokens,
487        BasketRange[] calldata newLimits,
488        Prices[] calldata newPrices,
489        uint256 auctionLauncherWindow,
490        uint256 ttl
491    ) external onlyRole(REBALANCE_MANAGER) nonReentrant notDeprecated sync {
           ...
544        rebalance.nonce++;
545        rebalance.startedAt = block.timestamp;
546        rebalance.restrictedUntil = block.timestamp + auctionLauncherWindow;
547        rebalance.availableUntil = block.timestamp + ttl;
548
549        emit RebalanceStarted(
550            rebalance.nonce,
551            newTokens,
552            newLimits,
553            newPrices,
554            block.timestamp + auctionLauncherWindow,
555            block.timestamp + ttl
556        );
557    }
```

*Figure C.1: Excerpt of the definition of `startRebalance` (reserve-index-dtf/contracts/Folio.sol#485–557)*

```
794    function endRebalance() external nonReentrant {
795        require(
796            hasRole(DEFAULT_ADMIN_ROLE, msg.sender) ||
797                hasRole(REBALANCE_MANAGER, msg.sender) ||
798                hasRole(AUCTION_LAUNCHER, msg.sender),
799            Folio__Unauthorized()
800        );
801
802        emit RebalanceEnded(rebalance.nonce);
803
804        // do not revert, to prevent griefing
805        rebalance.nonce++; // advancing nonce clears auctionEnds
806        rebalance.availableUntil = block.timestamp;
807    }
```

*Figure C.2: The definition of endRebalance (reserve-index-dtf/contracts/Folio.sol#794–807)*

The comment on line 805 in figure C.2 is worth noting: the nonce is advanced to "clear `auctionEnds`." What we believe this means is: the `auctionEnds` mapping (described in the next section) will have no entries for the new nonce. Hence, any entry referred to with the new nonce will consist entirely of zeroes.

## Auction End Times

Auction end times are referred to in two ways: by an `endTime` field within an `Auction` struct, and an `auctionEnds` mapping stored in a `Folio` contract. The `auctionEnds` mapping identifies auctions using two keys: a `Rebalance` nonce and a sell-token-buy-token pair. The redundancy between the `endTime` fields and the `auctionEnds` mappings has at least two adverse effects on the code.

First, in two locations, the two data structures must be updated together, so that they stay in sync (figures C.3 and C.4).

```
773     function closeAuction(uint256 auctionId) external nonReentrant {
774         require(
775             hasRole(DEFAULT_ADMIN_ROLE, msg.sender) ||
776                 hasRole(REBALANCE_MANAGER, msg.sender) ||
777                 hasRole(AUCTION_LAUNCHER, msg.sender),
778             Folio__Unauthorized()
779         );
780         Auction storage auction = auctions[auctionId];
781
782         // do not revert, to prevent griefing
783         auction.endTime = block.timestamp - 1;
784
auctionEnds[auction.rebalanceNonce][AuctionLib.pairHash(auction.sellToken,
auction.buyToken)] =
785             block.timestamp -
786             1;
787
788         emit AuctionClosed(auctionId);
789     }
```

*Figure C.3: The definition of `closeAuction` (reserve-index-dtf/contracts/Folio.sol#773–789)*

```
179     function bid(
180         IFolio.Auction storage auction,
181         mapping(bytes32 pair => uint256 endTime) storage auctionEnds,
182         uint256 totalSupply,
183         uint256 sellAmount,
184         uint256 bidAmount,
185         bool withCallback,
186         bytes calldata data
187     ) external returns (bool shouldRemoveFromBasket) {
            ...
231         if (sellBasketPresence == auction.sellLimit || buyBasketPresence >=
auction.buyLimit) {
232             auction.endTime = block.timestamp - 1;
233             auctionEnds[pairHash(auction.sellToken, auction.buyToken)] =
block.timestamp - 1;
234         }
235     }
```

*Figure C.4: Excerpt of the definition of `bid` (reserve-index-dtf/contracts/utils/AuctionLib.sol#179–235)*

Second, the `auctionEnds` mapping is used to detect "auction collisions" (figure C.5), which does not appear to be the mapping's primary purpose. Specifically, before an `auctionEnds` entry is set, the entry is checked to see whether the time it holds is at least

`auctionBuffer` (a parameter) seconds in the past. If it is not, then another auction is still active, and "collision" is considered to have occurred.

```
48    // confirm no auction collision on token pair
49    {
50        bytes32 pair = AuctionLib.pairHash(args.sellToken, args.buyToken);
51        require(
52            block.timestamp > auctionEnds[rebalance.nonce][pair] +
args.auctionBuffer,
53            IFolio.Folio__AuctionCollision()
54        );
55
56        auctionEnds[rebalance.nonce][pair] = block.timestamp + auctionLength;
57    }
```

*Figure C.5: Code for detecting "auction collisions" (reserve-index-dtf/contracts/utils/AuctionLib.sol#48–57)*

Furthermore, if there is no current auction for a `Reblance` nonce and sell-token-buy-token pair, the `auctionEnds` entry will be zero. In such a case, the just-mentioned check is still performed and is expected to succeed. Thus, the code relies on zero being `auctionBuffer` seconds in the past. Put another way, the code relies on `auctionBuffer` being less than the current number of seconds since the Unix epoch. While likely to hold for any reasonable choice of `auctionBuffer`, the constraint is artificial, as there is no a priori reason for `auctionBuffer` to be less than the number of seconds since the Unix epoch.

## Recommendations

To address the issues described above, we recommend the following:

- Store a flag in the `Rebalance` struct to record whether a rebalance operation is in progress.
- Advance the `Rebalance`'s nonce when `endRebalance` is called and not when `startRebalance` is called.
- Replace `auctionEnds` with a mapping from a `Rebalance` nonce and a sell-token-buy-token pair to an auction ID (rather than an end time).

Advancing the `Rebalance` nonce during `endRebalance` only will eliminate unused nonces. Moreover, the proposed flag could be used to determine whether the current nonce represents a rebalance operation that is in progress, or one that has yet to begin.

Eliminating `auctionEnds` would mean that just one data structure stores auction end times (i.e., the `endTime` fields in `Auction` structs). This, in turn, would eliminate the need to keep multiple such data structures in sync.

Finally, detecting auction collisions would be made more straightforward. No longer would `auctionBuffer` be required to be less than the number of seconds since the Unix epoch. Instead, the code could check whether the current `Rebalance` nonce and token pair have been assigned an auction ID. If they have, and if the auction with that ID has an `endTime` less than `auctionBuffer` seconds in the past, there is a collision. Otherwise, there is no collision.

# D. Functions that Iterate over Folio State

`Folio` functions introduced between commits c88e958 and e82bfd6 iterate over variable-sized data structures stored in the `Folio` contract. Such an approach can lead to problems. If a data structure becomes too large, a function that iterates over it may become uncallable because the gas the function requires would exceed the block gas limit.

As of this report, the block gas limit is 35.97 million. Based on our estimates, a `Folio` would need to contain 2.75 million different tokens for a call to `startRebalance` to exceed this block gas limit, for example.[2] Thus, hitting such a limit seems impractical.

Nonetheless, we recommend regularly checking the gas usage of the functions listed below. Mechanisms for doing so include the following Forge features:

- Gas Reports
- `vm.startSnapshotGas(...)` and `vm.stopSnapshotGas()`

Regularly checking the functions' gas usage can help you to know when they could become uncallable before they actually do.

## Folio Whose Gas Limits Should Be Regularly Checked

The following externally callable functions iterate over the `basket.values` function:

- Directly
  - `getRebalance`
  - `startRebalance`
- Via `_totalAssets`
  - `totalAssets`
- Via `_toAssets` -> `_totalAssets`
  - `toAssets`
  - `mint`
  - `redeem`

[2] A recent Trezor post estimates that there are 1.4 million ERC-20 tokens on the Ethereum network in total.

The following externally callable functions iterate over the `feeRecipients` function:

- Directly
    - `distributeFees`
- Via `distributeFees` and `_setFeeRecipients`
    - `setFeeRecipients`

# E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On June 6, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Reserve Protocol team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 5 issues described in this report, Reserve Protocol has partially resolved one issue and has not resolved the remaining four issues. For additional information, please see the Detailed Fix Review Results below.

| ID | Title | Severity | Status |
|---|---|---|---|
| 1 | Insufficient test coverage of new features | Informational | Partially Resolved |
| 2 | Functions can be called repeatedly, causing misleading events | Low | Unresolved |
| 3 | Order surplus extraction | Low | Unresolved |
| 4 | DAO can receive slightly lower fees than expected | Informational | Unresolved |
| 5 | Denial of service allows blocking operations during Dutch auctions | Medium | Unresolved |

## Detailed Fix Review Results

**TOB-FOLIOTF-1: Insufficient test coverage of new features**
Partially resolved in PR #136. Tests have been added that increase the coverage of the contracts folder of the Reserve Index DTF repository from 93% line coverage to 99%. The `Trusted-Fillers` repository tests were not changed.

**TOB-FOLIOTF-2: Functions can be called repeatedly, causing misleading events**
Unresolved. The Reserve Protocol team provided the following context for this finding's fix status:

> *Acknowledge, no change.*

**TOB-FOLIOTF-3: Order surplus extraction**
Unresolved. The Reserve Protocol team provided the following context for this finding's fix status:

> *Acknowledge with caveat that finding be revised to include 1% CowSwap fee max. This feature works as intended, as we intended to incentivize other entities to run the offchain bot so we don't have to, and how much they can get paid to do so is bounded but still incentivized.*

**TOB-FOLIOTF-4: DAO can receive slightly lower fees than expected**
Unresolved. The Reserve Protocol team provided the following context for this finding's fix status:

> *Acknowledge, no change.*

**TOB-FOLIOTF-5: Denial of service allows blocking operations during Dutch auctions**
Unresolved. The Reserve Protocol team provided the following context for this finding's fix status:

> *Dispute finding on mainnet; acknowledge on non-priority-fee-based L2s. The argument under priority fees is that the cost to the attacker to DoS each block is equal to the MEV that could be extracted via the bid in that block, but they cannot guarantee they will be the beneficiary of the lower price in the next block, so it does not make sense financial sense to pay the higher priority fee unless you plan to bid yourself.*

# F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

| Fix Status | |
|---|---|
| **Status** | **Description** |
| **Undetermined** | The status of the issue was not determined during this engagement. |
| **Unresolved** | The issue persists and has not been resolved. |
| **Partially Resolved** | The issue persists but has been partially resolved. |
| **Resolved** | The issue has been sufficiently resolved. |

# About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up to date with our latest news and announcements, please follow @trailofbits on X or LinkedIn, and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

**Trail of Bits, Inc.**
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

# Notices and Remarks

## Copyright and Distribution

© 2025 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be business confidential information; it is licensed to Reserve under the terms of the project statement of work and intended solely for internal use by Reserve. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications, if published, is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

## Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.