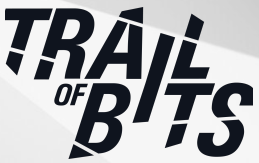

Reserve Folio Solidity Contracts

Security Assessment (Summary Report)

April 18, 2025

Prepared for:
Patrick McKelvy
Reserve.org

Prepared by: **Benjamin Samuels, Coriolan Pinhas, and Nicolas Donboly**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Benjamin Samuels, Consultant
benjamin.samuels@trailofbits.com

Coriolan Pinhas, Consultant
coriolan.pinhas@trailofbits.com

Nicolas Donboly, Consultant
nicolas.donboly@trailofbits.com

Project Timeline

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

Date	Event
January 23, 2025	Pre-project kickoff call
February 7, 2025	Delivery of report draft
February 10, 2025	Report readout meeting
April 18, 2025	Delivery of final summary report

Project Targets

The engagement involved a review and testing of the following target.

Reserve Folio Solidity Contracts

Repository	github.com/reserve-protocol/reserve-index-dtf
Branch	`main`
Commit	`c88e958bb70211bbb73c601382ef9e9b1611f00c`
Type	Solidity

Executive Summary

Engagement Overview

Reserve.org engaged Trail of Bits to review the security of Reserve's Solidity-based Folio contracts. The project allows the construction of managed ETF-like funds called Folios, which track a basket of different tokens, creating a synthetic fund that owns certain proportions of the basket tokens.

A team of three consultants conducted the review from February 3 to February 7, 2025, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The review focused on the `Folio` contract, the `StakingVault` contract, the `UnstakingManager` contract, and associated deployment contracts. During the review, coverage was focused on the system's use of third-party OpenZeppelin dependencies, governance risks, token support issues, and arithmetic issues.

The code under audit was previously reviewed by another company several months before this engagement, and its unfixed findings were excluded from this review's scope.

Due to time constraints, we were unable to fully cover the following, which deserve further investigation:

- The system's end-to-end deployment cycle (deployment, configuration, eventual upgrade, etc.)
- Tolerance to unusual tokens such as rebasing and ERC-777 tokens
- Certain arithmetic issues including precision loss and rounding issues
- Incorrect configuration of the third-party OpenZeppelin contracts
- Edge cases related to auctions (e.g., buying shares of a basket of low decimal tokens and selling when the basket is composed of high decimal tokens, etc.)

The codebase is relatively well assembled; however, as detailed in TOB-FOLIO-3, some documentation shortcomings prevent third parties from performing adequate due diligence. In addition, while the documentation states that the Folio protocol supports ERC-20 extensions such as ERC-777, the review found no explicit support for such tokens (TOB-FOLIO-7).

Recommendations

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.
- **Improve the system's unit tests around boundary conditions.** One finding disclosed in the report (TOB-FOLIO-4) would have been detected by unit tests that test the system's boundary conditions. Adding more boundary condition unit tests could help the team uncover similar issues.
- **Improve the system's unit tests for nonstandard tokens.** The system's documentation specifies that ERC-777 and rebasing tokens are intended to be supported by the Folio protocol; however, there were no tests for these tokens at the time of the engagement. Tests for these tokens can help detect bugs that may have slipped past manual review.

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	Fixed-point arithmetic is used consistently with explicit decimal scaling (D18 and D27) and clear documentation of precision handling. The code shows careful consideration of rounding directions, using explicit libraries such as OpenZeppelin and PRBMath libraries.	**Satisfactory**
Authentication / Access Controls	The system's different roles and responsibilities are well documented; however, one role that is critical to the system's operation has insufficient documentation (TOB-FOLIO-3).	**Satisfactory**
Complexity Management	The code is well organized and makes good use of third-party libraries to reduce first-party code risk. However, the system could be considered overly configurable, which increases the risk of issues occurring during deployment. For example, critical security parameters such as Folio guardians, timelock parameters, and governance parameters are set on a Folio-to-Folio basis, increasing the system's risk.	**Satisfactory**
Cryptography and Key Management	There are no instances of cryptography or key management in the reviewed codebase.	**Not Applicable**
Decentralization	While the protocol maintains control over Folio whitelisting and front end visibility, the key decentralization aspect lies in the Folio structure itself: each Folio operates independently with its creator having significant control. The protocol's guardian system represents another centralization point, in that it is capable of blocking protocol operations but cannot directly access or manipulate user funds. The system	**Moderate**

	implements clear boundaries between protocol-level controls and Folio-level autonomy, with users maintaining sovereignty over their assets despite the presence of privileged actors.	
Documentation	The documentation provides clear explanations of the roles and key functionalities of the protocol. However, it lacks detailed descriptions of the underlying mathematical mechanisms, the guardian role, and the whitelisting of Folios in the front end.	**Satisfactory**
Low-Level Manipulation	There are no instances of low-level manipulation in the codebase.	**Strong**
Testing and Verification	The project demonstrates good test coverage, approaching 100%. However, given the variety of possible cases, such as decentralized versus centralized Folios and interactions with unusual tokens, the test suite could be expanded. It currently lacks comprehensive tests for all token types that should be supported, including ERC-777 and rebasing tokens. The codebase would benefit from fuzzing test cases.	**Moderate**

Summary of Findings

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

ID	Title	Type	Severity
1	GovernanceDeployer does not enforce minimum values for timelock contract	Configuration	**Informational**
2	StakingVault is vulnerable to ERC-4626 griefing attack	Timing	**Low**
3	Fully on-chain governance creates existential governance attack risks	Documentation	**Low**
4	Users do not receive shares for low mint requests	Data Validation	**Informational**
5	Missing slippage protection on the Folio contract's mint function	Timing	**Medium**
6	Denial of service vulnerability via configurable initial supply	Denial of Service	**Medium**
7	ERC-777 compatibility issue	Undefined Behavior	**Medium**
8	Folio.bid() is vulnerable to denial of service through 1 wei donation attack	Denial of Service	**Medium**
9	Wei loss occurs when transferring stETH rebasing tokens	Configuration	**Informational**

Detailed Findings

1. GovernanceDeployer does not enforce minimum values for timelock contract

Severity: **Informational**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-FOLIO-1
Target: `contracts/deployer/GovernanceDeployer.sol`	

Description
The system's governance deployer does not enforce minimum values for the governance executor timelock contract.

The governance timelock contract is a critical component of the system, and its presence and correct configuration prevent several more serious attacks from being viable (some of which are described in TOB-FOLIO-6).

Recommendations
Short term, enforce a minimum amount of time for the timelock in the governance deployer contract. Ideally, this amount of time should be the minimum amount of time required for a guardian to detect and cancel a successful malicious proposal.

Long term, maintain a threat model that can be used to identify the minimum safe values for various parameters in the Folio system and ensure safeguards are in place to prevent Folios from being created with parameters outside the safe range.

2. StakingVault is vulnerable to ERC-4626 griefing attack	
Severity: **Low**	Difficulty: **Medium**
Type: Timing	Finding ID: TOB-FOLIO-2
Target: `contracts/staking/StakingVault.sol`	

Description

The `StakingVault` contact is vulnerable to an inflation attack, where an attacker donates assets to manipulate share issuance. Also, it inherits from ERC-4626 but does not generate yield, making price-per-share tracking unnecessary. OpenZeppelin's `_decimalsOffset()` function helps mitigate the issue but does not fully eliminate it.

```
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view
virtual returns (uint256) {
    return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1,
rounding);
}
```

Figure 2.1: contracts/token/ERC20/extensions/ERC4626.sol#L225–L227

Exploit Scenario

An attacker deposits 1 wei, receiving 1 share. The attacker front runs a user's 1e18 deposit by donating 1e18 to the vault. The user deposits 1e18 but receives only 1 share. The attacker redeems 1 share, receiving 666666666666666667 tokens (and losing 333333333333333333 tokens). The user redeems 1 share, receiving 666666666666666667 tokens (and losing 333333333333333333 tokens).

Recommendations

Short term, override `totalAssets()` to return `totalSupply()`, ensuring a fixed one-to-one redemption rate.

Long term, thoroughly evaluate third-party code before building on it. This will help identify potential vulnerabilities and ensure a more secure implementation.

3. Fully on-chain governance creates existential governance attack risks	
Severity: **Low**	Difficulty: **High**
Type: Documentation	Finding ID: TOB-FOLIO-3
Target: N/A	

Description
Folios deployed using the `deployGovernedFolio` function have their owner set to a governance timelock governed by a `FolioGovernor` contract, which manages proposals and submits successful proposals to the governance timelock contract.

The Folio owner has complete control over the Folio and can upgrade the `Folio` contract to steal funds if desired. An attacker may be able to use this to carry out a larger governance attack by purchasing or borrowing a large amount of governance tokens, depositing them in the `StakingVault` contract, and creating proposals to upgrade the Folio contracts and steal their funds.

It should be noted that if such proposals pass, they must still wait a certain amount of time specified by the timelock contract before execution. The Folio's guardian role can choose to cancel a pending proposal; however, this is effectively the only defense against the attack.

Without additional documentation, users must trust guardians as the only line of defense against governance attacks and assume they are implemented adequately.

Exploit Scenario
An EOA address is assigned the guardian role for one or more Folios. There is no documented automation in place, so users are left to assume that guardianship is performed manually on a case-by-case basis.

An attacker then performs a governance attack when the team is unavailable or no longer actively maintaining the project.

Recommendations
Short term, create publicly available documentation and automation for the guardian role. Ensure that the guardian software is highly available and can handle network edge cases such as unusually high gas prices. While this solution is not complete, publicly documenting the guardian and its controls should allow greater trust in the protocol.

Long term, use a management solution that is not vulnerable to on-chain governance attacks or other governance issues such as inactivity. Some DAOs use subcommittees to

achieve this goal while remaining permissionless. Subcommittees are usually implemented as X-of-N multisignature addresses composed of members who are elected by the DAO. While subcommittees can create their own problems, they prevent the immediate threat of pure on-chain governance attacks.

4. Users do not receive shares for low mint requests

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-FOLIO-4
Target: `contracts/Folio.sol`	

Description

Minting 1 share via `Folio.mint()` results in no shares because the share amount in the function rounds down. This causes deposits of low amounts, such as 1 wei, to round down the amount of minted shares to zero, so they effectively lose their collateral when they try to mint a low amount of shares.

```
(_assets, _amounts) = _toAssets(shares, Math.Rounding.Ceil);

uint256 assetLength = _assets.length;
for (uint256 i; i < assetLength; i++) {
    if (_amounts[i] != 0) {
        SafeERC20.safeTransferFrom(IERC20(_assets[i]), msg.sender, address(this),
_amounts[i]);
    }
}
```

Figure 4.1: `contracts/Folio.sol#L314–L321`

Recommendations

Short term, implement a minimum share minting threshold and add a check to ensure that `shares - fees` is greater than `0`.

Long term, add unit tests that verify the system's edge case boundaries. In addition, consider using stateful fuzzing to test the system's boundary condition behavior.

5. Missing slippage protection on the Folio contract's mint function

Severity: **Medium**	Difficulty: **High**
Type: Timing	Finding ID: TOB-FOLIO-5
Target: `contracts/Folio.sol`	

Description

There is no slippage protection on the `Folio` contract's `mint` function, and the existing slippage protection mechanism fails to protect against excessive fees.

```
_mint(receiver, shares - totalFeeShares);
```

Figure 5.1: contracts/Folio.sol#326

The Reserve team explained to us it will rely on token allowance to prevent users from sending too many tokens; however, the `mint` function can also mint fewer shares than expected if the Folio system's fees change (the `totalFeeShares` value in figure 5.1), such as if mint fees change just before a user mints shares.

Exploit Scenario

The following are proof-of-concept tests that can be added to `Folio.t.sol`.

```
function test_POC_without_fees() public {
    assertEq(folio.balanceOf(user1), 0, "wrong starting user1 balance");
    uint256 startingUSDCBalance = USDC.balanceOf(address(folio));
    uint256 startingDAIBalance = DAI.balanceOf(address(folio));
    uint256 startingMEMEBalance = MEME.balanceOf(address(folio));

    // set mintFee to 5%
    vm.prank(owner);
    folio.setMintFee(0);
    // DAO cut is at 50%

    vm.startPrank(user1);
    USDC.approve(address(folio), type(uint256).max);
    DAI.approve(address(folio), type(uint256).max);
    MEME.approve(address(folio), type(uint256).max);

    uint256 amt = 1e22;
    folio.mint(amt, user1);
    console.log("User 1 shares when no fees: ", folio.balanceOf(user1));
}

function test_POC_fee_change_before_the_user_tx() public {
```

```
    assertEq(folio.balanceOf(user1), 0, "wrong starting user1 balance");
    uint256 startingUSDCBalance = USDC.balanceOf(address(folio));
    uint256 startingDAIBalance = DAI.balanceOf(address(folio));
    uint256 startingMEMEBalance = MEME.balanceOf(address(folio));

    // set mintFee to 5%
    vm.prank(owner);
    folio.setMintFee(MAX_MINT_FEE);
    // DAO cut is at 50%

    vm.startPrank(user1);
    USDC.approve(address(folio), type(uint256).max);
    DAI.approve(address(folio), type(uint256).max);
    MEME.approve(address(folio), type(uint256).max);

    uint256 amt = 1e22;
    folio.mint(amt, user1);
    console.log("User 1 shares when fees change before user's tx: ",
folio.balanceOf(user1));
}
```

Figure 5.2: Proof-of-concept tests

Then in the console, running `forge test --match-test test_POC -vv` will produce the output:

```
[PASS] test_POC_fee_change_before_the_user_tx() (gas: 351383)
Logs:
  User 1 shares when fees change before user's tx:  950000000000000000000

[PASS] test_POC_without_fees() (gas: 311523)
Logs:
  User 1 shares when no fees:  998500000000000000000
```

Figure 5.3: Output of the test file

Recommendations

Short term, add a `minShares` argument to the function and add a check at the end of the function for the following requirement:

```
require(minShares <= shares - totalFeeShares);
```

Figure 5.4: Protection against slippage

Long term, add tests to make sure users are not vulnerable to slippage.

6. Denial of service vulnerability via configurable initial supply

Severity: **Medium**	Difficulty: High
Type: Denial of Service	Finding ID: TOB-FOLIO-6
Target: `contracts/Folio.sol`	

Description

The `Folio` contract of the Reserve protocol is vulnerable to a denial-of-service attack due to an overflow condition in the `totalSupply` computation. The root cause of this issue lies in the fact that the initial supply of Folio tokens can be set too high; if that occurs, all of the Folio's functions will revert.

Exploit Scenario

A malicious actor can freeze all of a Folio's actions by exploiting an overflow in the `_getPendingFeeShares` computation. This can occur as follows:

- Suppose a Folio's current supply is `1.16e35 * 1e18` with a basket containing 1 WETH and 100 USDC.
- The attacker mints shares to make the token supply equal to `(2**256 - 1) / 1e18`, which requires depositing 1e6 WETH and 1e8 USDC.
- This leads to a denial of service due to an overflow in the `_getPendingFeeShares` computation, effectively freezing the Folio.

This attack can be profitable if mixed with other attacks such as a governance attack to make sure users cannot redeem their tokens during the attack.

The following is a proof of concept for this attack that can replace the current `Folio.t.sol` content:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.28;

import { IFolio } from "contracts/interfaces/IFolio.sol";
import { Folio, MAX_AUCTION_LENGTH, MIN_AUCTION_LENGTH, MAX_AUCTION_DELAY, MAX_TTL,
MAX_FEE_RECIPIENTS, MAX_TVL_FEE, MAX_MINT_FEE, MAX_PRICE_RANGE, MAX_RATE } from
"contracts/Folio.sol";
import { MAX_DAO_FEE } from "contracts/folio/FolioDAOFeeRegistry.sol";
import { Math } from "@openzeppelin/contracts/utils/math/Math.sol";
import { FolioProxyAdmin, FolioProxy } from "contracts/folio/FolioProxy.sol";
import { IAccessControl } from "@openzeppelin/contracts/access/IAccessControl.sol";
```

```
import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
import { ITransparentUpgradeableProxy } from
"@openzeppelin/contracts/proxy/transparent/TransparentUpgradeableProxy.sol";
import { ReentrancyGuardUpgradeable } from
"@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
import { ERC1967Utils } from
"@openzeppelin/contracts/proxy/ERC1967/ERC1967Utils.sol";
import { FolioDeployerV2 } from "test/utils/upgrades/FolioDeployerV2.sol";
import { MockReentrantERC20 } from "test/utils/MockReentrantERC20.sol";
import "./base/BaseTest.sol";

contract FolioTest is BaseTest {
    uint256 internal constant INITIAL_SUPPLY = 1e35 * 1e18; // 1e35 tokens with 18
decimals
    uint256 internal constant MAX_TVL_FEE_PER_SECOND = 3340960028; // D18{1/s} 10%
annually, per second

    IFolio.BasketRange internal FULL_SELL = IFolio.BasketRange(0, 0, MAX_RATE);
    IFolio.BasketRange internal FULL_BUY = IFolio.BasketRange(MAX_RATE, 1,
MAX_RATE);

    IFolio.Prices internal ZERO_PRICES = IFolio.Prices(0, 0);

    function _testSetup() public virtual override {
        super._testSetup();
        _deployTestFolio();
    }

    function _deployTestFolio() public {
        address[] memory tokens = new address[](3);
        tokens[0] = address(USDC);
        tokens[1] = address(DAI);
        tokens[2] = address(MEME);
        uint256[] memory amounts = new uint256[](3);
        amounts[0] = 10 * 1e6; // 10 tokens with 6 decimals
        amounts[1] = 10 * 1e18; // 10 tokens with 18 decimals
        amounts[2] = 10 * 1e27; // 10 tokens with 27 decimals
        IFolio.FeeRecipient[] memory recipients = new IFolio.FeeRecipient[](2);
        recipients[0] = IFolio.FeeRecipient(owner, 0.9e18);
        recipients[1] = IFolio.FeeRecipient(feeReceiver, 0.1e18);

        // 50% tvl fee annually
        vm.startPrank(owner);
        USDC.approve(address(folioDeployer), type(uint256).max);
        DAI.approve(address(folioDeployer), type(uint256).max);
        MEME.approve(address(folioDeployer), type(uint256).max);

        (folio, proxyAdmin) = createFolio(
            tokens,
            amounts,
            1e35 * 1e18, // 1e35 tokens with 18 decimals of initial supply
            MAX_AUCTION_DELAY,
            MAX_AUCTION_LENGTH,
```

```
            recipients,
            MAX_TVL_FEE,
            0,
            owner,
            dao,
            auctionLauncher
        );
        vm.stopPrank();
    }

    function test_POC() public {
        vm.startPrank(user1); //user1 will be the attacker here

        deal(address(USDC), address(user1), 1e50);
        deal(address(DAI), address(user1), 1e50);
        deal(address(MEME), address(user1), 1e50);
        USDC.approve(address(folio), type(uint256).max);
        DAI.approve(address(folio), type(uint256).max);
        MEME.approve(address(folio), type(uint256).max);

        folio.mint(1e60, user1);
        // The attacker has sent:
        // - 100.000.000 USDC
        // - 100.000.000 DAI
        // - 100 000 000 MEME

        // All the Folio is frozen now
        // console.log(folio.totalSupply());
        vm.expectRevert();
        folio.totalSupply();
    }
}
```

Figure 6.1: Proof of concept

Recommendations

Short term, add checks on initialization to make sure users cannot create Folios with big numbers of shares for a small number of tokens.

Long term, add tests with extreme cases, or use fuzzing to identify configurations that should be prevented using initialization constraints.

7. ERC-777 compatibility issue	
Severity: **Medium**	Difficulty: **Low**
Type: Undefined Behavior	Finding ID: TOB-FOLIO-7
Target: `contracts/Folio.sol`	

Description

The `Folio` contract of the Reserve protocol may face compatibility issues when interacting with ERC-777 tokens due to missing support for the ERC-1820 registry and the handling of token granularity.

Incompatible ERC-777 Transfers

According to the ERC-777 standard, when tokens are transferred to a contract, the transaction should revert if the recipient contract does not implement the `ERC777TokensRecipient` interface via ERC-1820. Currently, the `Folio` contract does not register itself in the ERC-1820 registry, preventing it from receiving ERC-777 tokens.

Denial of Service via Token Granularity

ERC-777 tokens have a granularity defined by the `granularity()` function. If the protocol attempts to send an amount that is not a multiple of the granularity, the transaction will fail. While best practice suggests using a granularity of 1, some tokens may deviate from this, causing potential operational issues.

Exploit Scenario

Incompatible ERC-777 Transfers

A user attempts to transfer ERC-777 tokens to the `Folio` contract, but the transaction fails since the contract is not registered in the ERC-1820 registry.

Denial of Service via Token Granularity

The protocol tries to transfer an amount not divisible by a token's granularity. The transaction fails, disrupting operations.

Recommendations

Short term, add a registry update in the `Folio` contract's initialization function to register itself as an `ERC777TokensRecipient` using the `setInterfaceImplementer` function of the ERC-1820 registry. Moreover, consider adding a check that executes when any token is added to make sure the token's granularity is equal to 1.

Long term, add tests with all kinds of tokens that are supposed to work with the protocol.

References

- ERC-777 specification

8. Folio.bid() is vulnerable to denial of service through 1 wei donation attack	
Severity: **Medium**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-FOLIO-8
Target: `contracts/Folio.sol`	

Description

An attacker can front run a user's bid by donating 1 wei of the buy token to the contract to make the bid transaction revert.

When a user's bid would cause the contract's buy token balance to exactly match `maxBuyBal` (`auction.buy.balanceOf(address(this)) == maxBuyBal`), the donation causes the transaction to revert at the final balance check.

The user would then have to rebid with a lower amount, potentially losing to another bidder in the meantime or to an attacker who wanted to buy the same lot for a lower price.

```
require(auction.buy.balanceOf(address(this)) <= maxBuyBal, Folio__ExcessiveBid());
```

Figure 8.1: `contracts/Folio.sol#L650`

Exploit Scenario

Alice submits a bid that would cause the contract's buy token balance to exactly match `maxBuyBal`. Bob front runs the bid transaction with a 1 wei donation. Alice's transaction reverts. Bob can now bid at a lower price point because of the exponential decay curve.

Recommendations

Short term, ensure that the `boughtAmt` is calculated using the contract's current balance plus the `sellAmount` instead of relying on an exact match with `maxBuyBal`.

Long term, thoroughly review each `require()` statement for denial-of-service opportunities.

9. Wei loss occurs when transferring stETH rebasing tokens

Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-FOLIO-9
Target: `contracts/Folio.sol`	

Description

Though stETH is a rebasing token, it behaves similarly to fee-on-transfer (FoT) tokens due to a known 1–2 wei rounding issue. This corner case causes the transferred amount to be slightly lower than expected because the stETH balance calculation rounds down, leading to a minor discrepancy in token balances. As a result, transfers in `Folio` and `FolioDeployer` may experience issues similar to those described in Trust Security's writeup from the previous report: "TRST-M-7 Folio is not compatible with Fee-on-Transfer tokens."

Transfers of stETH will result in the same behavior as FoT tokens, but with minimal impact, typically a 1–2 wei loss per transfer.

Recommendations

Short term, document this integration risk for users.

Long term, avoid integrating rebasing tokens. Consider using wstETH as an alternative to mitigate this issue.

References

- Lido tokens integration guide: 1–2 wei corner case
- `lidofinance/core`, #442: Account's stETH balance getting lower on 1 or 2 wei due to rounding down integer math

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. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. 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 April 14, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Reserve 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 nine issues described in this report, Reserve has resolved four issues and has accepted the risk of the remaining five issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	GovernanceDeployer does not enforce minimum values for timelock contract	Informational	Risk Accepted
2	StakingVault is vulnerable to ERC-4626 griefing attack	Low	Risk Accepted
3	Fully on-chain governance creates existential governance attack risks	Low	Risk Accepted
4	Users do not receive shares for low mint requests	Informational	Resolved
5	Missing slippage protection on the Folio contract's mint function	Medium	Resolved
6	Denial of service vulnerability via configurable initial supply	Medium	Resolved
7	ERC-777 compatibility issue	Medium	Resolved
8	Folio.bid() is vulnerable to denial of service through 1 wei donation attack	Medium	Risk Accepted
9	Wei loss occurs when transferring stETH rebasing tokens	Informational	Risk Accepted

Detailed Fix Review Results

TOB-FOLIO-1: GovernanceDeployer does not enforce minimum values for timelock contract
Reserve has acknowledged this finding and has decided to accept the finding's risk.

TOB-FOLIO-2: StakingVault is vulnerable to ERC-4626 griefing attack
Reserve has acknowledged this finding and has decided to accept the finding's risk.

TOB-FOLIO-3: Fully on-chain governance creates existential governance attack risks
Reserve has acknowledged this finding and has decided to accept the finding's risk.

TOB-FOLIO-4: Users do not receive shares for low mint requests
Resolved in PR #106. Calls to the `mint` function will now revert if no shares will ultimately be minted.

TOB-FOLIO-5: Missing slippage protection on the Folio contract's mint function
Resolved in PR #99. The changes add a `minSharesOut` parameter that prevents slippage attacks.

TOB-FOLIO-6: Denial of service vulnerability via configurable initial supply
Resolved. The Reserve team indicated that the initial supply parameter will be configured using a factory contract, which will ensure the initial supply is within acceptable bounds, preventing exploitation of this issue.

TOB-FOLIO-7: ERC-777 compatibility issue
Resolved. The Reserve team indicated that the protocol will no longer accept ERC-777 tokens as valid collateral.

TOB-FOLIO-8: Folio.bid() is vulnerable to denial of service through 1 wei donation attack
Reserve has acknowledged this finding and has decided to accept the finding's risk. Reserve believes that exploitation of this finding is not likely due to the small amount of MEV that could be extracted from each block.

TOB-FOLIO-9: Wei loss occurs when transferring stETH rebasing tokens
Reserve has acknowledged this finding and has decided to accept the finding's risk.

D. 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 projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter 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

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.