Kiln Lagoon Vault Version 0.6.0

Security Assessment

May 11, 2026

Prepared for:
Loïc Titren
Kiln

Prepared by: **Tarun Bansal**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Tara Goodwin-Ruffus, Project Manager
tara.goodwin-ruffus@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:

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Project Timeline

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

Date	Event
April 20, 2026	Pre-project kickoff call
April 28, 2026	Delivery of report draft
April 28, 2026	Report readout meeting
April 29, 2026	Completion of fix review
May 11, 2026	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Kiln engaged Trail of Bits to review the security of version 0.6.0 of the Lagoon vault protocol. Lagoon is an ERC-7540-compliant tokenized vault that supports asynchronous deposit and redemption flows through an epoch-based settlement mechanism, alongside a synchronous path for deposit and withdrawal operations. Version 0.6.0 introduces entry and exit fees, a haircut mechanism for synchronous redemptions, integration with an external sanctions list oracle for access control, and expanded support for both whitelist and blacklist operating modes.

One consultant conducted the review from April 20 to April 24, 2026, for a total of one engineer-week of effort. Our testing efforts focused on the correctness of the ERC-7540 implementation, the mathematical soundness of the new fee calculation logic, the consistency of share and asset conversions between settlement and claim operations, and the correct integration of new features, including the sanctions list and synchronous withdrawal paths. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes. Our review did not cover deployment scripts, the off-chain components associated with the valuation manager and safe accounts, or other off-chain infrastructure.

Observations and Impact

The codebase reflects care in its organization and documentation. The library-based modular architecture and thorough inline comments reduced the effort required to understand component interactions and improved the system's overall reviewability during this engagement.

We identified one medium-severity and one low-severity finding during the review. Both relate directly to new features introduced in v0.6.0.

The entry fee calculation introduces an arithmetic inconsistency between settlement and claim operations (TOB-KILGND-2). The settlement function computes the entry fee on aggregate shares after conversion; the two claim functions instead compute it on asset amounts. Under integer division, these approaches produce different results whenever the vault's share-to-asset price ratio deviates from one, which is the normal state for any vault that has appreciated or declined in value. As a result, users who call the standard deposit claim function with round-number amounts encounter reverts. The workaround path overcharges users in assets and leaves a small residual permanently unreachable. This finding is notable because the triggering conditions require no special privileges or knowledge of timing: any user depositing a round-number amount into a profitable vault will encounter it.

The sanctions list integration creates a single external point of failure for access control (TOB-KILGND-1). The oracle is an upgradeable third-party contract; if it reverts, all vault functions that enforce access control, including deposit, redemption, cancellation of pending requests, and share transfers, stop responding.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Kiln take the following steps prior to deploying v0.6.0 in production:

- **Remediate the findings disclosed in this report.** Both findings affect core user operations, the ability to claim settled deposits, and the continued availability of access-controlled vault functions during oracle disruptions. These findings should be addressed through direct code fixes.
- **Extend the test suite with property-based tests covering fee arithmetic.** The entry fee inconsistency is not exposed by tests that use a 1:1 price ratio. Property-based tests that vary deposit amounts, fee rates, and price ratios, including configurations representing low-decimal underlying tokens with large virtual reserves, will prevent similar arithmetic divergences from being introduced in future versions.
- **Establish a defensive integration policy for external dependencies.** The sanctions list oracle is the first external contract dependency introduced into the vault's critical access-control path. A consistent practice of wrapping such calls in error-handling logic with defined fallback behavior will bound the impact of future dependency failures and allow users to maintain access to their funds.

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	1
Informational	0
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Denial of Service	1

Project Goals

The engagement was scoped to provide a security assessment of version 0.6.0 of the Kiln Lagoon vault protocol. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are the entry and exit fee calculations mathematically sound, and are fees applied consistently across settlement, asynchronous claim, and synchronous execution paths?
- Is the haircut mechanism for synchronous redemptions correctly implemented, and does it accurately redistribute value to remaining shareholders?
- Does the synchronous deposit and redemption path correctly enforce all preconditions, including `SyncMode` gating, `totalAssets` freshness, and `MaxCap` limits?
- Is the external sanctions list oracle integrated safely, and can a revert or malfunction of the oracle deny users access to their assets or prevent recovery by privileged roles?
- Does the expanded access control model, supporting both whitelist and blacklist modes and the new `SuperOperator` role, correctly enforce per-operation restrictions and bypass rules as specified?
- Do the `cancelRequestRedeem`, `claimSharesOnBehalf`, and `claimAssetsOnBehalf` functions correctly enforce access control and provide the intended recovery paths for affected users?
- Are the APR-based guardrails correctly enforced on `totalAssets` updates, and does the `SecurityCouncil` bypass path prevent circumvention of these bounds in unintended scenarios?
- Is the `VaultInit` `delegatecall` pattern secure against reinitialization, and do the ERC-7201 namespaced storage slots used across all libraries remain free of collisions after the v0.5.1-to-v0.6.0 upgrade?
- Can the vault owner control the share tokens issued to users?

Project Targets

The engagement involved reviewing and testing the following target.

lagoon-v0

Repository	https://github.com/hopperlabsxyz/lagoon-v0
Version	`d07028d43c77d3519bae8689d2af4b8ee313c39f`
Type	Solidity
Platform	EVM

lagoon-v0 safe and super user lock feature changes

Repository	https://github.com/hopperlabsxyz/lagoon-v0
Version	`37a84cb6bd6c65d81da437cc67639af1f1b32a3a`
Type	Solidity
Platform	EVM

lagoon-v0 changes made to remove onlyAsyncDeposit modifier

Repository	https://github.com/hopperlabsxyz/lagoon-v0
Version	`a8e73f5a5276aa4047b901083cbce127d7f7b470`
Type	Solidity
Platform	EVM

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:

- **`VaultInit` `delegatecall`, storage layout, and upgradeability:** We reviewed the `VaultInit` contract, which contains the initialization logic and is executed via a `delegatecall` from the vault's own `initialize` function to keep the initialization bytecode out of the main contract. We checked that reinitialization is prevented, that all parameters from `InitStruct` are correctly decoded and forwarded to the respective module initializers, and that the no-op overrides in `VaultInit` do not expose callable state-modifying paths. We also examined the ERC-7201 namespaced storage slots used across all libraries, verifying that each library consistently uses the same `_get...Storage()` helper to target the vault's storage rather than its own, and that the slot assignments do not collide with each other or with OpenZeppelin base contracts. Finally, we reviewed the upgrade path from v0.5.1 to v0.6.0, checking that new storage variables are appended within their respective namespaced structs and that no existing slot assignments were displaced.

- **Entry and exit fee arithmetic review:** We analyzed the entry fee computation across all three execution paths—`settleDeposit`, `_deposit`, and `_mint`—examining whether the fee domain, conversion direction, and rounding mode are consistent between settlement and claim operations. We applied the same analysis to the exit fee paths in `settleRedeem`, `_redeem`, and `_withdraw`, and evaluated the arithmetic consequences for all price-ratio regimes, including vaults with low-decimal underlying tokens and large virtual reserves.

- **Synchronous deposit and redeem review:** We examined the `syncDeposit` and `syncRedeem` functions, checking enforcement of preconditions, `SyncMode` gating, `totalAssets` freshness and expiration, `MaxCap` enforcement, and asynchronous-only activation, as well as the inline entry fee and haircut application. We verified that access control and pause checks are correctly enforced on these paths.

- **External sanctions list integration:** We reviewed how `AccessableLib.isAllowed` integrates the external oracle and examined whether a revert from the oracle propagates unchecked into all guarded operations. We enumerated every `isAllowed` call site to map the scope of operations affected by an oracle failure and identified which execution paths have privileged bypass routes and which do not.

- **Whitelist/blacklist mode and `SuperOperator` access control extensions:** We reviewed the `AccessMode` switch between whitelist and blacklist modes, tracing

how each mode governs ERC-20 transfers and the full set of restricted vault operations. We examined the `SuperOperator` bypass rules across all guarded functions for consistency with the specification.

- **Deposit and redemption cancellation flows:** We reviewed the `cancelRequestDeposit` and `cancelRequestRedeem` functions, checking the epoch-based cancellability guard, the asset and share return paths through the `Silo`, and access control enforcement, including whether privileged roles can cancel requests on behalf of affected users.
- **Safe-initiated claim operations:** We reviewed the `claimSharesOnBehalf` and `claimAssetsOnBehalf` functions, examining their per-controller iteration flows, the application of access control checks, and their ability to serve as a recovery path when users are unable to claim directly.
- **Guardrails and security council:** We reviewed the `GuardrailsManager` contract, examining how APR upper and lower bounds gate `updateNewTotalAssets` calls, the time-weighted scaling of bounds to the elapsed interval, the `SecurityCouncil` bypass path, and the conditions under which guardrails can be disabled or overridden.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- Deployment scripts
- Upgrade and migration scripts
- The safe account smart contract and the related off-chain component
- Any other off-chain components

Summary of Findings

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

ID	Title	Type	Severity
1	A revert from the SanctionsList contract can cause a temporary denial of service	Denial of Service	Low
2	The deposit claim functions compute the entry fee on assets instead of shares	Data Validation	Medium

Detailed Findings

1. A revert from the SanctionsList contract can cause a temporary denial of service

Severity: **Low**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-KILGND-1
Target: `src/v0.6.0/libraries/AccessableLib.sol`, `src/v0.6.0/libraries/ERC7540Lib.sol`	

Description

The `isAllowed` function in `AccessableLib` makes an unguarded external call to the sanctions oracle (figure 1.1). If the oracle reverts for any reason, the revert propagates through `isAllowed` to every function that gates access with it, temporarily blocking most vault operations for all users.

```
// if the external sanctions list is defined, we check if the account is not
sanctioned
if ($.externalSanctionList != SanctionsList(address(0))) {
    externalListApproval = !$.externalSanctionList.isSanctioned(account);
}
```

Figure 2.1: Unguarded external call to the sanctions oracle (`src/v0.6.0/libraries/AccessableLib.sol#L155–L158`)

The oracle is an upgradeable contract and a single point of failure. It can revert when it is paused, when it is upgraded to an incompatible interface, or when it runs out of gas. Because `isAllowed` is called at the entry point of every user-facing operation, a reverting oracle blocks the following actions:

- `requestDeposit`: Assets cannot enter the vault.
- `syncDeposit` and `syncRedeem`: Synchronous paths are blocked.
- `deposit` and `mint`: Users cannot claim settled shares.
- `requestRedeem`: Users cannot initiate redemptions.
- `redeem` and `withdraw`: Users cannot claim settled assets.
- `transfer` and `transferFrom` (in blacklist mode): Share transfers are blocked.

Crucially, `cancelRequestDeposit` also calls `isAllowed` without a `SuperOperator` bypass:

```
if (!AccessableLib.isAllowed(controller)) {
    revert AddressNotAllowed(controller);
}
```

Figure 2.2: `cancelRequestDeposit` calls `isAllowed` with no `SuperOperator` bypass. (`src/v0.6.0/libraries/ERC7540Lib.sol#L515-L517`)

This means users whose assets are held in the `Silo` pending settlement cannot recover them while the oracle is down, and neither the `safe` nor the `SuperOperator` can cancel the request on their behalf. Settlement via `settle()` is not affected because it does not call `isAllowed`, so the valuation manager can still advance epochs.

Exploit Scenario

The vault operator configures an oracle as the external sanctions list. The oracle provider releases an upgrade that introduces a breaking change in the `isSanctioned` function. After the upgrade, every call to `isAllowed` reverts. All vault operations that call `isAllowed`, including `requestDeposit`, `cancelRequestDeposit`, `deposit`, `redeem`, and ERC-20 transfers in blacklist mode, immediately start reverting. Users who submitted deposit requests in the current epoch cannot cancel them to recover their assets from the `Silo`. The safe cannot call `claimSharesOnBehalf` or `claimAssetsOnBehalf` to rescue users because both internally invoke `isAllowed`. The vault remains in this degraded state until the oracle provider restores compatibility or the vault owner sets a new oracle address.

Recommendations

Short term, wrap the `isSanctioned` call in `AccessableLib.isAllowed` in a try/catch block. When `isSanctioned` reverts, it should treat the account as not sanctioned (returning `true` for `externalListApproval`) so that oracle unavailability does not block vault operations.

Long term, add an event that is emitted on failures in the sanctions list oracle contract; monitor for such events, and take action when they are detected to maintain the security and availability of the protocol's functions.

2. The deposit claim functions compute the entry fee on assets instead of shares

Severity: **Medium**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-KILGND-2
Target: `src/v0.6.0/libraries/ERC7540Lib.sol`	

Description

The settlement and claim functions apply the entry fee in different arithmetic domains, producing results that are not equivalent under integer division when the vault's price ratio deviates from one. The mismatch causes `_deposit` to revert unconditionally for common deposit amounts and causes `_mint` to permanently strand a residual portion of a user's deposited assets.

During settlement, `settleDeposit` converts all pending assets to gross shares using a floor division operation, then deducts the entry fee from that share count:

```
uint256 shares = IERC4626(address(this)).convertToShares(_pendingAssets); // Floor
// ...
uint256 entryFeeShares = FeeLib.computeFee(shares, _entryFeeRate);        // Ceil on
shares
ERC7540(address(this)).forge(address(this), shares - entryFeeShares);
```

Figure 2.1: `settleDeposit` computes the entry fee in the share domain. (src/v0.6.0/libraries/ERC7540Lib.sol#L638–L659)

At claim time, `_deposit` instead deducts the fee from the user's asset amount first and then converts the net assets to shares:

```
uint256 entryFeeAssets = FeeLib.computeFee(assets,
getSettlementEntryFeeRate(requestId)); // Ceil on assets
shares = ERC7540(address(this)).convertToShares(assets - entryFeeAssets, requestId);
// Floor
IERC20(address(this)).safeTransfer(receiver, shares);
```

Figure 2.2: `_deposit` computes the entry fee in the asset domain. (src/v0.6.0/libraries/ERC7540Lib.sol#L464–L466)

The `_mint` function takes the same asset-domain approach: it reverse-computes the required assets from the requested share count and applies `computeFeeReverse` on top:

```
assets = convertToAssets(shares, requestId, Math.Rounding.Ceil);                  //
```

```
Ceil
assets += FeeLib.computeFeeReverse(assets, getSettlementEntryFeeRate(requestId)); //
Ceil again
$.epochs[requestId].depositRequest[controller] -= assets;
IERC20(address(this)).safeTransfer(receiver, shares);
```

Figure 2.3: `_mint` computes the entry fee in the asset domain with two consecutive `Ceil` operations. (src/v0.6.0/libraries/ERC7540Lib.sol#L496–L502)

The three formulations are algebraically equivalent in continuous arithmetic, but they diverge under integer arithmetic whenever the price ratio $R = (\texttt{totalSupply} + \texttt{offset}) / (\texttt{totalAssets} + 1)$ is not one. The settlement function's floor conversion of assets to shares discards a fractional part $\delta \in [0, 1)$. The `_deposit` path converts assets net of fee to shares separately, and because the fee on assets is subtracted before the floor conversion, the claim can recover enough of δ to produce one more share than the settlement function allocated. The resulting `ERC20InsufficientBalance` revert is the only guard; it triggers consistently when the fee on assets is an exact integer—that is, for every deposit amount that is an exact multiple of `BPS/fee_rate` (e.g., every 100-asset multiple at a 1% fee rate). These are precisely the round deposit amounts that users naturally choose.

The failure modes across the three price-ratio regimes are as follows:

- **R < 1 (profitable vault—the most common operating state for vaults after NAV appreciation):** `_deposit(fullAmount)` reverts for multiples of `BPS/r` assets. The `_mint` call succeeds but consumes one more asset than settlement requires, leaving a residual share that cannot be recovered through either claim path.
- **R > 1 (vault after a loss or fee minting):** For vaults with moderate R, `_deposit` reverts under the same integer condition. For USDC vaults where $R \approx 10^{12}$, non-round deposit amounts cause `_deposit` to compute fees that are 10^{12} times too large in share terms, permanently stranding that many additional shares in the vault per surplus fee unit.
- **R = 1 (exact 1:1 price—a transient state at vault initialization):** `_deposit` does not revert for a single depositor; with multiple depositors, shares may be stuck due to the super-additivity of `Ceil`, but no revert occurs.

The shares stuck in the vault have no sweep mechanism and cannot be recovered by either the safe or any privileged role.

Exploit Scenario

A profitable vault holds WETH (`totalAssets` = `1,500`, `totalSupply` = `1,000`; each share worth 1.5 assets). A user deposits 100 assets during the epoch. The valuation manager settles the epoch: `settleDeposit` converts 100 assets to 66 gross shares (floor division), deducts 1 share as the entry fee (`Ceil` of 0.66), and mints 65 shares to the vault. When the

user calls `deposit(100)` to claim those shares, `_deposit` computes the entry fee as 1 asset (`Ceil` of 1.0), converts the net 99 assets to shares at the stored price, and obtains 66, one more than the 65 the vault holds. The transfer of shares reverts. The user must switch to `mint(65)`, which succeeds but deducts 99 assets instead of the fair 98, leaving 1 asset stranded in the `depositRequest`. That 1-asset residual cannot be claimed: `_mint(1)` requires 3 assets to satisfy the double-`Ceil` and underflows, while `_deposit(1)` charges the full 1-asset fee, producing 0 shares and silently consuming the residual. The user's 100 deposited assets yield 65 shares and a permanently stranded 1-asset remainder.

Recommendations

Short term, rewrite `_deposit` to compute the entry fee in the share domain, consistent with `settleDeposit`. Specifically, include a call to `convertToShares(assets, requestId)` first to obtain the gross share count, then a call to `computeFee` on that share count to determine the fee shares, and have the remainder transferred to the user. Apply the same share-domain approach to the `_mint` function.

Long term, add property-based tests that verify the invariant `shares_claim` ≤ `sharesForUsers` for all deposit amounts, fee rates, and price ratios, covering USDC vault configurations (with a decimal offset of 10^{12}) and both the profitable-vault ($R < 1$) and loss ($R > 1$) regimes.

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**
Not Applicable	This issue is of informational severity and does not pose an immediate risk, so difficulty does not apply.
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. 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 29, 2026, Trail of Bits reviewed the fixes and mitigations implemented by the Kiln team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, Kiln has resolved the two issues described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	A revert from the SanctionsList contract can cause a temporary denial of service	Low	Resolved
2	The deposit claim functions compute the entry fee on assets instead of shares	Medium	Resolved

Detailed Fix Review Results

TOB-KILGND-1: A revert from the SanctionsList contract can cause a temporary denial of service

Resolved. The Kiln team acknowledged the issue but stated that reverts from the `SanctionsList` contract causing denial of service on user operations is intended behavior, as it prevents the platform from being exposed to sanctioned users. The client provided the following context for this finding's fix status:

> *We acknowledge that the first issue regarding the sanction list contract reverting is intended behavior.*

TOB-KILGND-2: The deposit claim functions compute the entry fee on assets instead of shares

Resolved in PR #309. The `_deposit` and `_mint` functions have been updated to compute the gross shares first and then compute the entry fee on those shares.

C. 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 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

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.