MixBytes()

# Gearbox Kelp Integration Security Audit Report

JANUARY 12, 2026

# Table of Contents

# 1. Introduction

## 1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

## 1.2 Executive Summary

Gearbox Protocol's Kelp integration enables Credit Account users to interact with Kelp's rsETH. Users deposit supported assets into Kelp's LRT Deposit Pool to mint rsETH tokens. When redeeming rsETH for underlying assets, users initiate withdrawal requests that are queued in Kelp's withdrawal manager and subject to a delay period before becoming claimable. The integration employs a gateway architecture where KelpLRTWithdrawalManagerGateway creates per-account helper contracts that interact with Kelp's withdrawal queue and handle the claiming of matured withdrawals. Throughout the withdrawal lifecycle, pending and claimable positions are represented via KelpLRTWithdrawalPhantomToken, allowing Gearbox to value these delayed withdrawals as collateral by reflecting both locked and unlocked request states in terms of the output asset.

This audit was performed over 3 days by 3 auditors and included comprehensive manual code review and automated static analysis.

During the audit, in addition to reviewing common attack vectors and our internal security checklist, we conducted a thorough analysis of the following areas:

- **Withdrawal Contract Residual Balance Accessibility.** After all withdrawal requests have been processed, any tokens that remain on the KelpLRTWithdrawer contract can still be recovered. We confirmed that the gateway allows the rightful user to withdraw these residual tokens once every request has been completed.
- **Phantom Token Collateral Accounting.** The KelpLRTWithdrawalPhantomToken includes assets already held by KelpLRTWithdrawer in its balanceOf() calculation. We validated that these amounts are correctly counted in the Credit Account's collateral, preventing users from artificially lowering their health factor.
- **Handling of Native ETH and WETH Conversions.** The integration relies on seamless wrapping and unwrapping between ETH and WETH. We confirmed that KelpLRTWithdrawer and KelpLRTDepositPoolGateway correctly manage these conversions—wrapping incoming ETH to WETH, unwrapping when required, and accounting for balances accurately—so that no user funds are lost or misattributed.
- **Correctness of Kelp Protocol Interactions.** The integration makes external calls to Kelp's deposit and withdrawal contracts (LRTDepositPool and LRTWithdrawalManager). We confirmed that all interactions, including deposit operations, withdrawal initiation, and withdrawal completion, are correctly implemented with proper parameter passing and interface compliance, ensuring reliable communication with the Kelp protocol.

- **Isolation of Per-Account Withdrawal Contracts.** The KelpLRTWithdrawer contracts are per-account helper contracts created via minimal proxy clones, with each Credit Account having its own dedicated withdrawer instance. We verified that all functions on KelpLRTWithdrawer are protected by the gatewayOnly modifier, ensuring that only the KelpLRTWithdrawalManagerGateway can interact with these contracts on behalf of Credit Accounts, and that unauthorized users cannot directly execute withdrawal operations or access funds belonging to other accounts.
- **Access Control for Administrative Functions.** The integration includes configuration functions that manage allowed assets and token mappings, such as setAssetStatusBatch() in KelpLRTDepositPoolAdapter and setTokensOutBatchStatus() in KelpLRTWithdrawalManagerAdapter. We confirmed that all administrative and configuration functions are protected by the configuratorOnly modifier, ensuring that only authorized configurator addresses can modify critical integration parameters and that these functions cannot be called by unauthorized users.

Integrations with delayed withdrawal protocols, such as Kelp, present a known challenge with pending withdrawal requests. The Gearbox team is aware of this limitation and has implemented specific mitigation mechanisms. When a Credit Account's collateral is largely represented by a non-transferable phantom token whose underlying is still pending/locked, that collateral cannot be immediately realized on-chain. As a result, liquidation flows that rely on unwrapping the phantom position may revert and be delayed until withdrawals become claimable. To address this, Gearbox uses safe prices for such positions and enforces a minimum health factor threshold that prevents accounts close to liquidation from initiating delayed withdrawals to avoid liquidation during volatile periods.

The underlying Kelp Protocol contracts (LRTDepositPool, LRTWithdrawalManager, and related components) were not audited as part of this audit. However, all interactions between Gearbox and Kelp contracts were thoroughly analyzed, including interface verification, integration patterns, and the security implications of external calls to Kelp's withdrawal queue and deposit mechanisms.

Overall, this integration demonstrates a solid implementation that adheres to security best practices. The architecture maintains clear separation between adapters, gateways, and phantom token components, with each layer functioning effectively within the integration. The codebase maintains high quality standards and security posture, and no severe vulnerabilities or critical issues were discovered during the review.

## 1.3 Project Overview

### Summary

| Title | Description |
|---|---|
| **Client** | Gearbox |
| **Category** | Lending |
| **Project** | Kelp Integration |

| Title | Description |
|---|---|
| **Type** | Solidity |
| **Platform** | EVM |
| **Timeline** | 19.12.2025 - 12.01.2026 |

## Scope of Audit

| File | Link |
|---|---|
| **contracts/adapters/kelp/KelpLRTDepositPoolAdapter.sol** | KelpLRTDepositPoolAdapter.sol |
| **contracts/adapters/kelp/KelpLRTWithdrawalManagerAdapter.sol** | KelpLRTWithdrawalManagerAdapter.sol |
| **contracts/helpers/kelp/KelpLRTDepositPoolGateway.sol** | KelpLRTDepositPoolGateway.sol |
| **contracts/helpers/kelp/KelpLRTWithdrawalManagerGateway.sol** | KelpLRTWithdrawalManagerGateway.sol |
| **contracts/helpers/kelp/KelpLRTWithdrawalPhantomToken.sol** | KelpLRTWithdrawalPhantomToken.sol |
| **contracts/helpers/kelp/KelpLRTWithdrawer.sol** | KelpLRTWithdrawer.sol |

## Versions Log

| Date | Commit Hash | Note |
|---|---|---|
| **19.12.2025** | 9d3c72a06d82c52418fa86b8d4a8385052af7727 | Initial Commit |
| **09.01.2026** | 646ed933878a0acced22b675d5f1e02e6c33655b | Commit for the reaudit |
| **12.01.2026** | 047163d347febcfdd09a609edfe192355a6ba529 | Commit with updates |

## Mainnet Deployments

Deployment verification will be conducted via
https://permissionless.gearbox.foundation/bytecode/

## 1.4 Security Assessment Methodology

### Project Flow

| Stage | Scope of Work |
|---|---|
| **Interim audit** | **Project Architecture Review:**<br><br>· Review project documentation<br>· Conduct a general code review<br>· Perform reverse engineering to analyze the project's architecture based solely on the source code<br>· Develop an independent perspective on the project's architecture<br>· Identify any logical flaws in the design<br><br>OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS. |
| | **Code Review with a Hacker Mindset:**<br><br>· Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities.<br>· Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead.<br>· Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws.<br>· Review test cases and in-code comments to identify potential weaknesses.<br><br>OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY. |
| | **Code Review with a Nerd Mindset:**<br><br>· Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits.<br>· Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors.<br><br>OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS. |

| Stage | Scope of Work |
|---|---|
| | **Consolidation of Auditors' Reports:**<br><br>• Cross-check findings among auditors<br>• Discuss identified issues<br>• Issue an interim audit report for client review<br><br>OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT. |
| Re-audit | **Bug Fixing & Re-Audit:**<br><br>• The client addresses the identified issues and provides feedback<br>• Auditors verify the fixes and update their statuses with supporting evidence<br>• A re-audit report is generated and shared with the client<br><br>OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED. |
| Final audit | **Final Code Verification & Public Audit Report:**<br><br>• Verify the final code version against recommendations and their statuses<br>• Check deployed contracts for correct initialization parameters<br>• Confirm that the deployed code matches the audited version<br>• Issue a public audit report, published on our official GitHub repository<br>• Announce the successful audit on our official X account<br><br>OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT. |

## 1.5 Risk Classification

### Severity Level Matrix

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likehood: High | Critical | High | Medium |
| Likehood: Medium | High | Medium | Low |
| Likehood: Low | Medium | Low | Low |

### Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

### Likelihood

- **High** – The event has a 50-60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10-20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

### Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

### Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

## 1.6 Summary of Findings

### Findings Count

| Severity | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 2 |

### Findings Statuses

| ID | Finding | Severity | Status |
|---|---|---|---|
| L-1 | Redundant Computation in `KelpLRTWithdrawalPhantomToken.balanceOf()` | Low | Fixed |
| L-2 | Withdrawal Completion Blocked When Token Is Removed From Whitelist | Low | Acknowledged |

# 2. Findings Report

## 2.1 Critical

Not Found

## 2.2 High

Not Found

## 2.3 Medium

Not Found

## 2.4 Low

| L-1 | Redundant Computation in KelpLRTWithdrawalPhantomToken.balanceOf() | | |
|---|---|---|---|
| **Severity** | Low | **Status** | Fixed in 646ed933 |

**Description**
The balanceOf() function calculates the user's phantom token balance by calling getPendingAssetAmount() and getClaimableAssetAmount() on the KelpLRTWithdrawalManagerGateway. Both of these gateway functions delegate the computation to the same per-account withdrawer contract.
Inside the withdrawer, each call performs a traversal over all the user's withdrawal requests to determine whether requests are still pending or already claimable. As a result, for a single balanceOf() call, the same set of withdrawal requests is iterated twice, once to compute pending assets and once to compute claimable assets.
This duplication does not affect correctness or safety, but it introduces unnecessary external calls and repeated loops, increasing gas usage and reducing efficiency for a commonly accessed view function.

**Recommendation**
We recommend refactoring the logic to compute pending and claimable amounts in a single traversal of the withdrawal requests and reuse the results.

***Client's Commentary:***
*Fixed in 646ed933878a0acced22b675d5f1e02e6c33655b*

| L-2 | Withdrawal Completion Blocked When Token Is Removed From Whitelist | | |
|---|---|---|---|
| **Severity** | Low | **Status** | Acknowledged |

**Description**

In KelpLRTWithdrawalManagerAdapter.completeWithdrawal(), a check enforces that asset is present in _allowedTokensOut. If the configurator removes a token from this whitelist while there are pending withdrawals for that token, users cannot complete the withdrawal—they remain stuck until the token is re-added to the whitelist.

**Recommendation**

We recommend not restricting completeWithdrawal() and withdrawPhantomToken() by the _allowedTokensOut whitelist, allowing completion of already-initiated withdrawals even if the asset was later disallowed, ensuring that queued requests can always be finalized.

***Client's Commentary:***

*Removing the whitelist check would allow control flow capture through the use of a custom token. Note that Kelp's contracts do not revert when a non-existing output token is passed to `userAssociatedNonces` or `nextLockedNonce`. This means that an attacker can send a custom token to the withdrawer, and then call `completeWithdrawal`, which will call `transfer` on that token (as there is existing balance on the withdrawer, even though the token is not supported by Kelp). As we want to avoid control capture as much as possible, it's simpler to keep the check intact. If any withdrawals become stuck due to actions by a curator, the curator can resolve them ad-hoc with users.*

# 3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

## Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

## Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

## Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes