MixBytes()

Gearbox Adapters Security Audit Report

OCTOBER 30, 2025

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 is an on-chain credit protocol that allows users to borrow funds and deploy leverage across major DeFi protocols using portable "Credit Accounts".

This audit focused on the newly introduced integrations with Mellow Finance and Kodiak Island. Our review examined their adapters to ensure compliance with Gearbox safety standards, prevention of asset lock-ups, and preservation of the core risk assumptions of both integrated protocols.

The codebase was audited over four days by a team of three auditors, using a combination of manual review and automated tooling.

In addition to reviewing known attack vectors and our internal checklist, we conducted an in-depth analysis of the following areas:

- **Correctness of the Kodiak oracle implementation.**
  The oracle calculates the value of the Kodiak LP token based on the prices of token0, token1, and the state of the Kodiak pool. We verified that the value is computed correctly and safely, including typical oracle-related issues such as incorrect handling of decimals and stale prices.

- **Correct accounting of assets related to Mellow.**
  Since Mellow implements a non-trivial withdrawal procedure, it is important to accurately account for all asset states, including unclaimed funds and those in a pending withdrawal state. We confirmed that the mechanism based on phantom tokens does not lead to either overestimation or underestimation of user assets in any system state.

- **Slippage protection in the Kodiak adapter.**
  The adapter implements its own slippage protection mechanism. We verified that it functions correctly, including in edge cases.

- **Absence of risk of user funds being lost or locked.**
  While the Gearbox architecture protects against most threats when adapter guidelines are followed, the risk of user funds being lost or permanently locked requires separate validation. We performed such a check.

- **Compliance with Gearbox adapter guidelines.**
  We reviewed the adapter implementations to ensure they follow Gearbox architecture safety rules. In particular:
  - Every external method interacting with a Credit Account is protected with the `creditFacadeOnly` modifier.
  - No user-specific state is stored, the contract holds only immutable parameters and whitelisted settings managed by the configurator.
  - The contract adheres to the `IVersion` interface, exposing immutable version and `contractType` constants.
  - Public functions correctly return the `useSafePrices` flag based on whether the call may produce new assets without a reliable oracle price.
  - Token allowances follow the safe-approve pattern: setting `type(uint256).max` before the external call and resetting to `1` immediately after.
  - The adapter implements a `serialize()` function, allowing off-chain reconstruction of its configuration.
  - All configuration operations are isolated in dedicated functions restricted by the `configuratorOnly` modifier.

Our review found no significant security concerns. Overall, the codebase is well-structured and reflects solid development practices. Nonetheless, we identified several findings that could improve security, logic, clarity, or maintainability. These are summarized in the **Findings Report** below.
We also recommend increasing test coverage for the Kodiak adapters to improve overall security and system robustness.

1.3 Project Overview

Summary

Title	Description
Client Name	Gearbox
Project Name	Kodiak & Mellow Adapters
Type	Solidity
Platform	EVM
Timeline	22.07.2025 - 31.10.2025

Scope of Audit

File	Link
contracts/adapters/mellow/Mellow4626VaultAdapter.sol	Mellow4626VaultAdapter.sol
contracts/adapters/mellow/MellowClaimerAdapter.sol	MellowClaimerAdapter.sol
contracts/helpers/mellow/MellowWithdrawalPhantomToken.sol	MellowWithdrawalPhantomToken.sol
contracts/adapters/kodiak/KodiakIslandGatewayAdapter.sol	KodiakIslandGatewayAdapter.sol
contracts/helpers/kodiak/KodiakIslandGateway.sol	KodiakIslandGateway.sol
contracts/oracles/kodiak/KodiakIslandPriceFeed.sol	KodiakIslandPriceFeed.sol
contracts/migration/LiquidityMigrator.sol	LiquidityMigrator.sol

Versions Log

Date	Commit Hash	Note
22.07.2025	ba55ede78d351695b20624714f51aa49843b4f4c	Initial commit (integrations-v3)
22.07.2025	4220625c6f46fdb816f3eb52396d8d62c2b96afe	Initial commit (oracles-v3)
29.07.2025	0d992f9d01f4f936b13ba242e7b6ffaf2a88a976	Re-audit commit (integrations-v3)
11.09.2025	cc8723b148eb6fbc9e6a0c13169fc3929449824d	Audit scope commit for LiquidityMigrator
12.09.2025	974034013e36853e5d001b762cc3a74a36037149	Re-audit commit for LiquidityMigrator

Date	Commit Hash	Note
31.10.2025	64a386a25d24ad5984926a3f624c93c7c32c92c9	Commit with updates for the KodiakIslandPriceFeed (oracles-v3)

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

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client 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:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account 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	1
Low	8

Findings Statuses

ID	Finding	Severity	Status
M-1	Incorrect Status Assignment Blocks Swaps	Medium	Fixed
L-1	Assignment Operator Mistakenly Replaced with Comparison in `_getSwapAdjustedMintAmounts()` function	Low	Fixed
L-2	Missing Mask Check for Island in `SWAP_AND_EXIT_ONLY` Branch	Low	Fixed
L-3	Use of Revert Strings Instead of Custom Errors	Low	Fixed
L-4	Incorrect NatSpec Comment on Return Values in `_getRatios()` Function	Low	Fixed
L-5	Missing Events in Batch Status Updates	Low	Fixed
L-6	Contract Deployment Will Revert If Asset Is USDT Due To `approve` Usage	Low	Fixed
L-7	Typo In Code Comment	Low	Fixed
L-8	Asset Compatibility Not Verified Between Source and Destination Pools	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Incorrect Status Assignment Blocks Swaps		
Severity	Medium	**Status**	Fixed in 0d992f9d

Description

In KodiakIslandGatewayAdapter.setIslandStatusBatch if a configurator attempts to set the island status to SWAP_AND_EXIT_ONLY, the adapter silently downgrades it to EXIT_ONLY. As a result, any subsequent call that relies on _isSwapAllowed() will revert, unintentionally disabling swap functionality.

```
} else if (islands[i].status == IslandStatus.SWAP_AND_EXIT_ONLY) {
    _getMaskOrRevert(token0);
    _getMaskOrRevert(token1);
    _getMaskOrRevert(islands[i].island);
    _allowedIslands.add(islands[i].island);
    _islandStatus[islands[i].island] = IslandStatus.EXIT_ONLY;
}
```

This issue is classified as **Medium** severity because while funds are not at risk, the bug blocks legitimate swap operations, causing significant disruption to protocol functionality and user experience.

Recommendation

We recommend assigning the correct status:

```
} else if (islands[i].status == IslandStatus.SWAP_AND_EXIT_ONLY) {
    _getMaskOrRevert(token0);
    _getMaskOrRevert(token1);
    _getMaskOrRevert(islands[i].island);
    _allowedIslands.add(islands[i].island);
--  _islandStatus[islands[i].island] = IslandStatus.EXIT_ONLY;
++  _islandStatus[islands[i].island] = IslandStatus.SWAP_AND_EXIT_ONLY;
```

2.4 Low

L-1	Assignment Operator Mistakenly Replaced with Comparison in `_getSwapAdjustedMintAmounts()` function		
Severity	Low	**Status**	Fixed in `0d992f9d`

Description

The function `KodiakIslandGateway._getSwapAdjustedMintAmounts()`, uses the comparison operator `==` instead of the assignment operator `=` when calculating `lpAmount` in two branches:

```
if (balance0 == 0) {
    lpAmount == depositAmount1 * totalSupply / balance1;
} else if (balance1 == 0) {
    lpAmount == depositAmount0 * totalSupply / balance0;
}
```

As a result, `lpAmount` is never updated in these cases, and the function always returns the default value (zero). This leads to incorrect LP token estimation and may break integrations or UI logic relying on this calculation.

Recommendation

We recommend replacing the `==` operator with the assignment operator `=` in both branches to ensure `lpAmount` is correctly set:

```
if (balance0 == 0) {
--  lpAmount == depositAmount1 * totalSupply / balance1;
++  lpAmount = depositAmount1 * totalSupply / balance1;
} else if (balance1 == 0) {
--  lpAmount == depositAmount0 * totalSupply / balance0;
++  lpAmount = depositAmount0 * totalSupply / balance0;
}
```

L-2	Missing Mask Check for Island in SWAP_AND_EXIT_ONLY Branch		
Severity	Low	**Status**	Fixed in 0d992f9d

Description

In KodiakIslandGatewayAdapter.setIslandStatusBatch, the branch handling the SWAP_AND_EXIT_ONLY status does **not** check that a mask exists for islands[i].island using _getMaskOrRevert. This is inconsistent with the other status branches, which do perform this check.

```
} else if (islands[i].status == IslandStatus.SWAP_AND_EXIT_ONLY) {
    _getMaskOrRevert(token0);
    _getMaskOrRevert(token1);
    _allowedIslands.add(islands[i].island);
    _islandStatus[islands[i].island] = IslandStatus.SWAP_AND_EXIT_ONLY;
}
```

Recommendation

We recommend simplifying and unifying the logic to consistently validate all necessary masks for non-NOT_ALLOWED statuses and handle the allowed islands accordingly:

```
function setIslandStatusBatch(
    KodiakIslandStatus[] calldata islands
) external override configuratorOnly {
    uint256 len = islands.length;
    for (uint256 i; i < len; ++i) {
        (address token0, address token1) = _getIslandTokens(islands[i].island);
        if (islands[i].status != IslandStatus.NOT_ALLOWED) {
            _getMaskOrRevert(token0);
            _getMaskOrRevert(token1);
            _getMaskOrRevert(islands[i].island);
            _allowedIslands.add(islands[i].island);
        } else {
            _allowedIslands.remove(islands[i].island);
        }

        _islandStatus[islands[i].island] = status;
    }
}
```

This unified approach fixes the missing mask check and also reduces code duplication and improves readability and maintainability.

L-3	Use of Revert Strings Instead of Custom Errors		
Severity	Low	**Status**	Fixed in 0d992f9d

Description

In the KodiakIslandGateway contract, revert statements use hardcoded error strings, for example:

```
if (amountOut < minAmountOut) {
    revert("KodiakIslandGateway: Insufficient amount");
}
```

Using revert strings increases gas costs because the entire error string is stored in the contract bytecode and included in the revert data.
Switching to custom errors is more gas efficient as they encode errors with selectors and optional parameters, reducing deployment and runtime costs.

Recommendation

We recommend replacing strings with custom errors.

L-4	Incorrect NatSpec Comment on Return Values in `_getRatios()` Function		
Severity	Low	**Status**	Fixed in `0d992f9d`

Description

The NatSpec comment for the internal function `KodiakIslandGateway._getRatios()` states that it returns a `Ratios` struct with **three** values:

- `depositRatio`
- `priceRatio`
- `is0to1`

However, the actual `Ratios` struct returned by the function contains **five** values:

```
struct Ratios {
    uint256 priceRatio;
    uint256 balance0;
    uint256 balance1;
    bool swapAll;
    bool is0to1;
}
```

This discrepancy between the comment and the code may lead to confusion and misunderstanding for developers reading or maintaining the contract.

Recommendation

We recommend updating the NatSpec comment to accurately reflect all fields returned by the `Ratios` struct, listing all five values to ensure clarity and correctness.

L-5	Missing Events in Batch Status Updates		
Severity	Low	**Status**	Fixed in 0d992f9d

Description

The functions `setIslandStatusBatch` in `KodiakIslandGatewayAdapter` and `setMultiVaultStatusBatch` in `MellowClaimerAdapter` do not emit any events when configuration changes are made. As a result, off-chain systems and monitoring tools cannot reliably track changes to allowed islands or multi-vaults, which may hinder transparency and auditing.

Recommendation

We recommend emitting appropriate events in both `setIslandStatusBatch` and `setMultiVaultStatusBatch` functions.

L-6	Contract Deployment Will Revert If Asset Is USDT Due To approve Usage		
Severity	Low	**Status**	Fixed in 97403401

Description

The LiquidityMigrator contract will fail to deploy if the underlying asset of poolTo is USDT (or other tokens that don't return a bool from approve). This happens because the contract uses direct IERC20.approve() calls, which expect a bool return value, while USDT's approve() function does not return any value.

Recommendation

We recommend using the forceApprove() function from the **SafeERC20** library by OpenZeppelin.

L-7	Typo In Code Comment		
Severity	Low	**Status**	Fixed in 97403401

Description

There is a typo in the code comment on line 40 of the LiquidityMigrator contract.

```
// Shares are redeemedfrom `poolFrom`,
// which returns the amount of assets withdrawn
```

The word redeemedfrom should be corrected to redeemed from.

Recommendation

We recommend fixing the typo.

L-8	Asset Compatibility Not Verified Between Source and Destination Pools		
Severity	Low	**Status**	Fixed in 97403401

Description

The `LiquidityMigrator` contract does not verify that the underlying assets of `poolFrom` and `poolTo` are the same during construction.

Recommendation

We recommend adding validation in the constructor to ensure that both pools use the same underlying asset.

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