

MixBytes()

# Gearbox Upshift Integration Security Audit Report

DECEMBER 18, 2025

# Table of Contents

<b>1. Introduction</b>	<b>2</b>
1.1 Disclaimer	2
1.2 Executive Summary	2
1.3 Project Overview	3
1.4 Security Assessment Methodology	5
1.5 Risk Classification	7
1.6 Summary of Findings	8
<b>2. Findings Report</b>	<b>9</b>
2.1 Critical	9
2.2 High	9
2.3 Medium	9
2.4 Low	9
L-1 Gateway Incorrectly Delays Claims Even When Vault Processes Instantly	9
<b>3. About MixBytes</b>	<b>10</b>

# 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 Upshift integration enables credit account users to deposit assets into Upshift Vault and receive vault shares that represent their position in the yield-generating pool. Users can redeem their shares for underlying assets through a two-step process: first requesting a redemption via `requestRedeem()`, which may be processed immediately if the vault's timelock is disabled or scheduled for a future date, and then claiming the assets via `claim()` once the claimable timestamp is reached. The `UpshiftVaultGateway` contract manages redemption requests and tracks individual user balances during the redemption period. Pending redemption positions can be valued as collateral by Gearbox during the redemption period via the `UpshiftVaultWithdrawalPhantomToken` contract, which reflects the claimable amount for each user.

This audit was performed in 1 day by 3 auditors, including comprehensive manual code review and automated static analysis.

As part of the security assessment, we examined the following potential attack vectors and security considerations:

- **Fee Handling During Redemption.** The Upshift Vault might apply fees during redemption operations. We validated that the amount of assets with fees applied is correctly written to the `pendingRedeems` mapping during `requestRedeem()`, and that the same amount is correctly fetched in the `claim()` function with no rounding or subtraction errors occurring.
- **Allowance Mechanism for Redeem Requests.** During `requestRedeem()` transaction, the Upshift Vault checks for an allowance from the Credit Account to the Gateway. We verified that the Credit Account correctly grants an allowance to the Gateway before this interaction, and no revert can occur due to insufficient allowance.
- **Daily Cluster Claim Mechanism and User Limits.** When `claim()` is called, all funds related to the current day cluster are transferred to the Gateway contract. We verified that each user can only redeem up to `pendingRedeems[msg.sender].assets` and no more, ensuring proper per-user accounting even when multiple users' funds are aggregated in the same daily cluster.
- **Public Claim Function in Vault.** Any user can call `claim()` directly in the Upshift Vault contract, which sends funds to the receiver address (the Gateway). We confirmed that this

does not break the gateway functionality, as `UpshiftVaultGateway.claim()` properly handles cases where the vault's `claim()` has already been called, and users are still able to claim their funds through the gateway.

We also note that the Upshift Vault implements a blacklist system that can restrict access to certain addresses. The functions `deposit()`, `mint()`, and `claim()` check whether `msg.sender` or the receiver address is blacklisted before allowing operations. Gearbox should be aware that if the Gateway address is blacklisted by the vault owner through `addToBlacklist()`, users will not be able to use the Gateway for `deposit`, `mint`, or `claim` operations.

The Upshift integration is implemented correctly and follows best security practices. The codebase demonstrates proper separation of concerns with the adapter, gateway, and phantom token contracts working together cohesively.

## 1.3 Project Overview

### Summary

Title	Description
Client	Gearbox
Category	Lending
Project	Upshift Integration
Type	Solidity
Platform	EVM
Timeline	12.12.2025 – 17.12.2025

### Scope of Audit

File	Link
<code>contracts/adapters/upshift/ UpshiftVaultAdapter.sol</code>	<a href="#">UpshiftVaultAdapter.sol</a>
<code>contracts/helpers/upshift/ UpshiftVaultGateway.sol</code>	<a href="#">UpshiftVaultGateway.sol</a>
<code>contracts/helpers/upshift/ UpshiftVaultWithdrawalPhantomToken.sol</code>	<a href="#">UpshiftVaultWithdrawalPhantomToken.sol</a>

## Versions Log

Date	Commit Hash	Note
12.12.2025	7a286681d96c61999e24d37067ca25bd92332c89	Initial Commit
16.12.2025	bdc69ec0de108b66f32109c6bab19a89cf7062b1	Re-audit Commit

## Mainnet Deployments

Deployment verification will be conducted via  
<https://permissionless.gearbox.foundation/bytocode/>

## 1.4 Security Assessment Methodology

### Project Flow

Stage	Scope of Work
Interim audit	<p><b>Project Architecture Review:</b></p> <ul style="list-style-type: none"><li>• Review project documentation</li><li>• Conduct a general code review</li><li>• Perform reverse engineering to analyze the project's architecture based solely on the source code</li><li>• Develop an independent perspective on the project's architecture</li><li>• Identify any logical flaws in the design</li></ul> <p><b>OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.</b></p>
	<p><b>Code Review with a Hacker Mindset:</b></p> <ul style="list-style-type: none"><li>• Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities.</li><li>• Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead.</li><li>• Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws.</li><li>• Review test cases and in-code comments to identify potential weaknesses.</li></ul> <p><b>OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.</b></p>
	<p><b>Code Review with a Nerd Mindset:</b></p> <ul style="list-style-type: none"><li>• Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits.</li><li>• Utilize static analysis tools (e.g., Slither, Mytril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors.</li></ul> <p><b>OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.</b></p>

Stage	Scope of Work
	<p><b>Consolidation of Auditors' Reports:</b></p> <ul style="list-style-type: none"> <li>• Cross-check findings among auditors</li> <li>• Discuss identified issues</li> <li>• Issue an interim audit report for client review</li> </ul> <p>OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.</p>
Re-audit	<p><b>Bug Fixing &amp; Re-Audit:</b></p> <ul style="list-style-type: none"> <li>• The client addresses the identified issues and provides feedback</li> <li>• Auditors verify the fixes and update their statuses with supporting evidence</li> <li>• A re-audit report is generated and shared with the client</li> </ul> <p>OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.</p>
Final audit	<p><b>Final Code Verification &amp; Public Audit Report:</b></p> <ul style="list-style-type: none"> <li>• Verify the final code version against recommendations and their statuses</li> <li>• Check deployed contracts for correct initialization parameters</li> <li>• Confirm that the deployed code matches the audited version</li> <li>• Issue a public audit report, published on our official GitHub repository</li> <li>• Announce the successful audit on our official X account</li> </ul> <p>OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.</p>

# 1.5 Risk Classification

## Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: 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	1

### Findings Statuses

ID	Finding	Severity	Status
L-1	Gateway Incorrectly Delays Claims Even When Vault Processes Instantly	Low	Fixed

# 2. Findings Report

## 2.1 Critical

Not Found

## 2.2 High

Not Found

## 2.3 Medium

Not Found

## 2.4 Low

L-1	Gateway Incorrectly Delays Claims Even When Vault Processes Instantly		
Severity	Low	Status	Fixed in bdc69ec0

### Description

When `lagDuration == 0`, `UpshiftVault.requestRedeem()` immediately calls `_claim()` and transfers assets to the gateway. However, `UpshiftVaultGateway.requestRedeem()` still stores `claimableTimestamp` from `UpshiftVault.getWithdrawalEpoch()`, which always adds the 5-minute `_TIMESTAMP_MANIPULATION_WINDOW` and rounds to midnight of the calculated day.

In most cases, this results in `claimableTimestamp` being midnight of the current day (in the past), so `UpshiftVaultGateway.claim()` can be called immediately. However, there is an edge case: when requests are made in the last 5 minutes before midnight (after 11:55 PM and up to 11:59:59 PM), `block.timestamp + 5 minutes` rolls over to the next day, making `claimableTimestamp` midnight of the next day (in the future). During this period, users cannot claim their funds and must wait until midnight, with the actual wait time varying from a few seconds to up to 5 minutes depending on when exactly the request is made, even though funds are already at the gateway.

### Recommendation

We recommend reading the current `UpshiftVault.lagDuration()` value and, if `lagDuration == 0` (i.e., claims are processed immediately by the vault), setting `claimableTimestamp = block.timestamp` in `UpshiftVaultGateway.requestRedeem()` for that request. Otherwise, keep using `UpshiftVault.getWithdrawalEpoch()` to populate `claimableTimestamp`.

# 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](https://github.com/mixbytes/audits_public)
-  [hello@mixbytes.io](mailto:hello@mixbytes.io)
-  <https://x.com/mixbytes>