MixBytes()

Gearbox Midas Integration 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 Protocol's Midas integration enables credit account users to deposit assets into Midas vaults and receive mTokens as collateral. The system supports both instant (redeeming mTokens for output tokens immediately) and delayed redemptions (requesting redemption, waiting for admin approval, then withdrawing tokens). The Gateway contract manages pending redemptions and tracks user balances, while phantom tokens enable Gearbox to value pending redemption requests as collateral.

The audit was conducted over 3 days by 3 auditors, involving an in-depth manual code review and automated analysis within the scope.

During the audit, we covered the following attack vectors and security considerations:

- We verified that during the `MidasRedemptionVaultGateway.requestRedeem()` call, the requestId is correctly retrieved from the Midas vault and properly recorded in the `pendingRedemptions` mapping, ensuring that the requestIds match with no discrepancies.
- During request approval, we confirmed that the `mTokenRate` of the request is updated by the Midas vault admin, and the Gateway contract correctly fetches the new rate when the `MidasRedemptionVaultGateway.withdraw()` function is called.
- We confirmed that users can withdraw only up to the `availableAmount` calculated by the Gateway in `MidasRedemptionVaultGateway._calculateTokenOutAmount()`. Once the full amount is withdrawn, users cannot claim more from their pending redemption request, and the `pendingRedemptions` mapping is correctly cleared.
- We verified that users can have only 1 active pending redemption at a time; a second call to `MidasRedemptionVaultAdapter.requestRedeem()` will fail, preventing data from being overwritten in the `pendingRedemptions` mapping until the `availableAmount` is fully claimed.
- We verified that the adapters correctly enforce the Gearbox adapter architecture constraints, including `onlyCreditFacade` and `onlyConfigurator` restrictions.
- We confirmed that the partial-withdrawal mechanism cannot lead to funds being locked on the Gateway; remaining claimable amounts are tracked and can be withdrawn subsequently until fully settled.
- We verified that user redemption/withdrawal requests are isolated: claims are bound to the originating credit account and cannot be withdrawn by other users.

It should be noted that the Midas vault functions used by the Gearbox adapters: `IMidasIssuanceVault.depositInstant()`, `IMidasRedemptionVault.redeemInstant()`, `IMidasRedemptionVault.redeemRequest()` are protected by access control modifiers:

- `whenFnNotPaused` - allows pausing specific function selectors

- `onlyGreenlisted` - allows removing accounts from the greenlist
- `onlyNotBlacklisted` - allows blacklisting accounts
- `onlyNotSanctioned` - allows sanctioning accounts

These access controls may prevent deposit or redemption operations from executing, causing user actions to revert.

The important consideration is that the Midas Redemption Vault adapter may experience delayed liquidations due to pending redemption requests. This is not considered a security issue as the mTokens and corresponding phantom tokens are going to be restricted to stable asset pools only, where price volatility is minimal, significantly reducing the risk of bad debt accumulation from delayed liquidations.

Overall, the code is of high quality, well-written and documented. The issues we identified during the audit are documented in the **Findings Report** section.

1.3 Project Overview

Summary

Title	Description
Client Name	Gearbox
Project Name	Midas Integration
Type	Solidity
Platform	EVM
Timeline	23.10.2025 - 30.10.2025

Scope of Audit

File	Link
contracts/adapters/midas/MidasRedemptionVaultAdapter.sol	MidasRedemptionVaultAdapter.sol
contracts/adapters/midas/MidasIssuanceVaultAdapter.sol	MidasIssuanceVaultAdapter.sol
contracts/helpers/midas/MidasRedemptionVaultGateway.sol	MidasRedemptionVaultGateway.sol

File	Link
contracts/helpers/midas/ MidasRedemptionVaultPhantomToken.sol	MidasRedemptionVaultPhantomToken.sol

Versions Log

Date	Commit Hash	Note
23.10.2025	e53ff92202fe2a16d12ec28617d4356666ef95dc	Initial commit
30.10.2025	cd8f68dd72ccb27eb0251c14759f6c8dc75358fc	Re-audit commit

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	1
Medium	1
Low	5

Findings Statuses

ID	Finding	Severity	Status
H-1	`rejectRequest()` Leads to Adverse Consequences	High	Fixed
M-1	`withdrawPhantomToken` May Withdraw an Unexpected Underlying	Medium	Fixed
L-1	Unnecessary Calculation For 18-Decimal Tokens In `_convertToE18()`	Low	Fixed
L-2	Lack Of Zero Amount Validation In `withdraw()`	Low	Fixed
L-3	String-Based Revert Messages Should Use Custom Errors	Low	Fixed
L-4	Unused Imports	Low	Fixed
L-5	Ambiguous `rateMinRAY` Direction and Decimals in Redemption Adapter	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	rejectRequest() Leads to Adverse Consequences		
Severity	High	**Status**	Fixed in cd8f68dd

Description

If the Midas vault admin calls rejectRequest() on a pending redemption request, the request status is set to **Cancelled** and the user's mTokens remain locked in the Midas vault. The Midas admin can transfer these mTokens back to any address, especially to the Gateway (which created the request). However, even if mTokens are transferred back to the Gateway, there's no mechanism to recover them from the Gateway to the user's credit account.
Additionally, if mTokens are sent to the credit account or any address other than the Gateway, MidasRedemptionVaultGateway.pendingTokenOutAmount() will continue to show a non-zero result despite the request being rejected, inflating the phantom token's balance and the credit account's collateral value.
This issue is classified as **High** severity because, despite being caused by the Midas admin, it can lead to several serious consequences: permanently blocked user funds, inability to recover mTokens from the Gateway, and inflated collateral calculations.

Recommendation

We recommend implementing functionality to handle such cases, allowing the system to clear rejected requests and transfer tokens out of the Gateway.

Client's Commentary

Client: When designing the gateway, we focused on being able to gracefully resolve issues arising from `rejectRequest()` that can negatively affect borrowers and LPs. In particular, there is no way to solve Midas transferring assets to the gateway without introducing strong new trust assumptions, as the Gateway may hold processed withdrawals from other users and a generic token transfer function would allow the authorized party to take their withdrawals. At the same time, transferring to the gateway would be an error by Midas that does not affect users (or rather, does not affect them further if `rejectRequest` already was called) - this is analogous to Midas accidentally transferring to any other unrecoverable address, such as address(0).
The gateway has functionality that allows to manually process a request inside the gateway if the request was rejected on the Midas side. This allows the market curator to collaborate with Midas to recover a withdrawal without additional friction to the Credit Account owner in order to correct the mistake (as calling `rejectRequest` for a gateway withdrawal would always be accidental).
As for account value inflation - this can simply be solved by proper monitoring and adjustment of risk parameters. In case a request rejection to the gateway is detected, the withdrawal phantom token can be forbidden, effectively disallowing any further borrows or withdrawals on the account while the phantom token is enabled as collateral on it - this nullifies any possible attack vectors from the troubled account. From there, there are several ways to correct the issue - the account owner can either cooperate with Midas and the market owner to process the withdrawal properly through the gateway, or

they can close the account and recover the funds transferred from Midas, at which point the Gearbox DAO can take the account out of the Account Factory to prevent further use.
With all this in mind, we believe that currently the gateway design strikes a good balance between complexity and ability to resolve issues stemming from incorrect behavior from Midas, and see no further reason to modify it.
MixBytes: The functionality, which allows manual processing of rejected requests, was not present in the initial commit - it was added later in response to the issue report. Therefore, the issue was marked as fixed.

2.3 Medium

M-1	withdrawPhantomToken May Withdraw an Unexpected Underlying		
Severity	Medium	Status	Fixed in cd8f68dd

Description

`MidasRedemptionVaultAdapter.withdrawPhantomToken(token, amount)` only checks that a phantom is registered for `token` and does not verify that the pending redemption's `tokenOut` matches the expected underlying. Integration may receive a different token than expected.

Recommendation

We recommend additionally checking that the redemption's `tokenOut` in Midas equals `phantomTokenToOutputToken[token]`; on mismatch, revert with a custom error.

Client's Commentary

The `token` field actually accepts the phantom token address, rather that the output token. In `babe15b4bd2d975b577e67e9ad917a6b7beffeb4`, we've added validation in `setTokenAllowedStatusBatch` to ensure that the phantom token's `tokenOut` matches the output token it was added alongside with.

2.4 Low

L-1	Unnecessary Calculation For 18-Decimal Tokens In `_convertToE18()`		
Severity	Low	**Status**	Fixed in `cd8f68dd`

Description

Both `MidasIssuanceVaultAdapter._convertToE18()` and `MidasRedemptionVaultAdapter._convertToE18()` perform unnecessary multiplication and division operations when the token already has 18 decimals. For tokens that use 18 decimals, the calculation `amount * WAD / tokenUnit` becomes `amount * 1e18 / 1e18 = amount`, which is redundant.
Similarly, `MidasRedemptionVaultGateway._calculateTokenOutAmount()` performs unnecessary operations when the output token has 18 decimals.

Recommendation

We recommend adding an early return for 18-decimal tokens to skip unnecessary calculations.

Client's Commentary

Fixed in `e42b11d2b5c3421bd055d04293ee86f273f02615`.

L-2	Lack Of Zero Amount Validation In withdraw()		
Severity	Low	**Status**	Fixed in cd8f68dd

Description

MidasRedemptionVaultGateway.withdraw() lacks a zero-amount check, allowing users to call the function with amount = 0 as a no-op. When called with zero, the function will still validate the pending redemption, fetch request data from Midas vault, check status, calculate availableAmount, and transfer 0 tokens, all while consuming gas unnecessarily.
Adding input validation would prevent accidental zero-amount calls and make the function's intent clearer.

Recommendation

We recommend adding a zero-amount check at the beginning of the withdraw() function:

```
 function withdraw(uint256 amount) external nonReentrant {
++  if (amount == 0) revert ZeroAmountException();
    // ... rest of the logic
}
```

Client's Commentary

Fixed in e42b11d2b5c3421bd055d04293ee86f273f02615.

L-3	String-Based Revert Messages Should Use Custom Errors		
Severity	Low	**Status**	Fixed in cd8f68dd

Description

MidasRedemptionVaultGateway contract uses string-based revert messages in multiple places. Since the contract uses Solidity ^0.8.23, which supports custom errors introduced in 0.8.4, these should be replaced with custom errors for gas efficiency and better developer experience. String revert messages consume more gas and are less type-safe compared to custom errors.

Recommendation

We recommend replacing string-based revert messages with custom errors.

Client's Commentary

Fixed in e42b11d2b5c3421bd055d04293ee86f273f02615.

L-4	Unused Imports		
Severity	Low	**Status**	Fixed in cd8f68dd

Description

Some files contain unused imports:

- contracts/adapters/midas/MidasIssuanceVaultAdapter.sol: ICreditManagerV3
- contracts/adapters/midas/MidasRedemptionVaultAdapter.sol: IMidasRedemptionVault

Recommendation

We recommend removing unused imports.

Client's Commentary

Fixed in e42b11d2b5c3421bd055d04293ee86f273f02615.

L-5	Ambiguous rateMinRAY Direction and Decimals in Redemption Adapter		
Severity	Low	**Status**	Fixed in cd8f68dd

Description

In MidasRedemptionVaultAdapter.redeemInstantDiff(), the rateMinRAY parameter is documented as *minimum rate from input token to mToken*, while for redemption the input is mToken and the output is tokenOut. The code computes minReceiveAmount as (amount * rateMinRAY) / RAY and then normalizes it to 1e18 for tokenOut, which implies:

- rateMinRAY = amountOut(tokenOut_decimals) / amountIn(mToken_decimals) * 1e27, i.e., it must account for decimal differences.

This ambiguity can lead integrators to supply a wrongly scaled rateMinRAY (e.g., without adjusting for decimals), causing excessive slippage rejections or unexpected reverts.

Recommendation

We recommend clarifying NatSpec in Midas adapters

Client's Commentary

Fixed in cd8f68dd72ccb27eb0251c14759f6c8dc75358fc.

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