MixBytes()

Yelay ERC4626 Security Audit Report

JANUARY 14, 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

The Yelay Lite protocol is a multi-strategy yield aggregation vault system, where the FundsFacet serves as the core vault implementation managing user deposits, withdrawals, and multi-strategy fund allocation across various DeFi protocols using an ERC1155 token standard for representing shares across different projects. The ERC4626Plugin and ERC4626PluginFactory provide a standardized ERC4626 vault interface wrapper around the underlying YelayLiteVault, allowing seamless integration with existing DeFi protocols.

The codebase was audited in 3 days by 3 auditors using a combination of manual review and automated tooling.

During the audit, the following attack vectors were checked:

Fee Rounding: We examined scenarios where the `accrueFee` function could be called with small interest amounts that result in `feeShares` rounding down to zero due to integer division in the share conversion formula. In cases where `totalInterest` multiplied by `totalSupply` is less than `lastTotalAssets`, the calculated fee shares round to zero. This attack vector is particularly relevant when fee accrual happens too frequently with minimal yield accumulation between calls, potentially allowing malicious actors to trigger fee accrual at inopportune times.

Beacon Proxy Initialization and Implementation Upgrade Vectors: We analyzed the beacon proxy pattern used in `ERC4626PluginFactory` to identify potential vulnerabilities in the initialization flow and upgrade mechanism, verifying that all necessary initializers are properly called during deployment and that the implementation contract cannot be left uninitialized. We verified that the beacon upgrade mechanism does not introduce vectors for implementation replacement attacks that could compromise existing plugin instances.

Withdrawal Margin Calculation and Asset Recovery Logic: We examined the `WITHDRAW_MARGIN` constant and its usage in the redeem function's asset calculation formula, specifically analyzing the line `assets = lack > WITHDRAW_MARGIN ? assets : assets - lack` to verify correct handling of partial withdrawals from strategies. The investigation focused on scenarios where strategies cannot provide the full requested withdrawal amount, and the margin is used to determine whether to accept the shortfall or use the originally calculated asset amount. We verified that the logic correctly handles cases where the lack of withdrawn funds is within acceptable tolerance (≤ `WITHDRAW_MARGIN`), adjusting the final asset amount to match what was actually withdrawn.

Internal Accounting Integrity and Direct Deposit Prevention: We conducted a thorough analysis of the FundsFacet's internal accounting mechanism, examining how totalAssets depends on both sF.underlyingBalance (vault's direct holdings) and LibManagement._strategyAssets(i) (funds deployed to strategies), and investigating whether attackers could manipulate these values through direct token transfers or deposits. The examination confirmed that while totalAssets calculation relies on strategy-reported balances (which could be manipulated by hacked or misbehaving strategies leading to incorrect vault valuation), there is no pathway for attackers to directly inflate sF.underlyingBalance through external deposits or transfers, as this value is only modified through controlled deposit and withdrawal flows within FundsFacet.

The primary focus of this audit was the ERC4626Plugin and ERC4626PluginFactory contracts. Additionally, changes to the FundsFacet and YieldExtractor contracts were reviewed as part of the scope, particularly the modifications introduced in pull request #27 (PR-27).

The codebase demonstrates good overall quality with well-structured architecture and proper use of OpenZeppelin's libraries. However, several serious issues were identified that require immediate attention, particularly inflation attack on ERC4626Plugin and lack of handling of fee-on-transfer tokens through balance-checking patterns with reentrancy protection. We also recommend implementing defensive mechanisms against hacked or misbehaving strategies.

1.3 Project Overview

Summary

Title	Description
Client Name	Yelay
Project Name	Yelay ERC4626
Type	Solidity
Platform	EVM
Timeline	23.10.2025 - 06.11.2025

Scope of Audit

File	Link
src/facets/FundsFacet.sol	FundsFacet.sol
src/YieldExtractor.sol	YieldExtractor.sol

File	Link
src/plugins/ERC4626Plugin.sol	ERC4626Plugin.sol
src/plugins/ERC4626PluginFactory.sol	ERC4626PluginFactory.sol

Versions Log

Date	Commit Hash	Note
23.10.2025	4ca9ca866fa21b4820820828f469985b35fa2e9b	Initial Commit
03.11.2025	4ae1252be72721500012eb130b3d995b173420f9	Commit for Re-audit
06.11.2025	ec5332121399f81ee988d1d3497179d551c7154e	Commit with Updates

Mainnet Deployments

File	Address	Blockchain
FundsFacet.sol	0xFE8aA2d7835a5022F853F01fDc042154ca4eD022	Base
YieldExtractor.sol	0xE8671aF5617b520dEB8Cc630A9D4a7745817D62D	Base
ERC4626PluginFactory.sol	0x0aADeF25ac515c6f9cCAD2afD23fCf90209F7346	Base
ERC4626Plugin.sol	0xE3d46c1e4123879944C407a118064c375Bc74E11	Base
FundsFacet.sol	0x1798b36C340DBA0A2bd7BD379dF100af07396e57	Arbitrum
YieldExtractor.sol	0xF4a23fC0f9beB15Df3d4e1a44425Ec559f746D4F	Arbitrum
ERC4626PluginFactory.sol	0x0aADeF25ac515c6f9cCAD2afD23fCf90209F7346	Arbitrum
ERC4626Plugin.sol	0x655f41973caF96b0aF68ABE732b50022c95374e7	Arbitrum
FundsFacet.sol	0x0A09027643d21F35df24f156694A97776d211907	Avalanche
YieldExtractor.sol	0x7Cdd48a57AD24BcE6195234B153b0b8a0ee3974f	Avalanche
ERC4626PluginFactory.sol	0x0aADeF25ac515c6f9cCAD2afD23fCf90209F7346	Avalanche

File	Address	Blockchain
ERC4626Plugin.sol	0x22623938 4EB7d78Cdf279BA6Fb458E2A4945E275	Avalanche
FundsFacet.sol	0x0F648F6EAF4AE90Dc8b1Edf54D9aAA9dBe87B9F8	Ethereum
YieldExtractor.sol	0x55A6CbAeFF11414e3819B4C7Fc8dF5eE244eD315	Ethereum
ERC4626PluginFactory.sol	0x0aADeF25ac515c6f9cCAD2afD23fCf90209F7346	Ethereum
ERC4626Plugin.sol	0x315f5f9EEaD29C3f370233AfebeE99F0adA5297d	Ethereum
FundsFacet.sol	0x680eb0997BB85Ac6f89a2Aeec9bb9157cFDF0ED1	Sonic
YieldExtractor.sol	0xdd4D9766B931d165E44890D0288A6f2Dd565AdEC	Sonic
ERC4626PluginFactory.sol	0x0aADeF25ac515c6f9cCAD2afD23fCf90209F7346	Sonic
ERC4626Plugin.sol	0x0F648F6EAF4AE90Dc8b1Edf54D9aAA9dBe87B9F8	Sonic

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	1
High	0
Medium	2
Low	4

Findings Statuses

ID	Finding	Severity	Status
C-1	ERC4626Plugin empty pool inflation attack	Critical	Fixed
M-1	Fee-on-Transfer Tokens Incompatibility in Deposit Flow	Medium	Acknowledged
M-2	Division by Zero Risk with Zero Total Assets	Medium	Acknowledged
L-1	Missing Zero Address and Input Validation Checks	Low	Acknowledged
L-2	Code Quality Issues and Potential Silent Failures	Low	Fixed
L-3	Incorrect Zero Floor Subtraction in Decimal Offset Calculation	Low	Acknowledged
L-4	Incorrect Slippage Check in ERC4626Plugin _withdraw Function	Low	Fixed

2. Findings Report

2.1 Critical

C-1	ERC4626Plugin empty pool inflation attack		
Severity	Critical	Status	Fixed in 4ae1252b

Description
The deposit path relies on OpenZeppelin's ERC4626Upgradeable.deposit() and ERC4626Upgradeable.mint() with virtual (dead) shares, which protect against the empty-pool inflation attack. However, ERC4626Plugin.redeem() and ERC4626Plugin.withdraw() are custom implementations that do not account for virtual shares. As a result, the first depositor can be exploited by combining a direct donation that increases totalAssets without increasing totalSupply with a rounding error.

- **Exploit steps**:
  1. The attacker mints 1 share in an empty plugin via ERC4626Plugin.mint(1, hacker).
  2. The attacker frontruns the victim's transaction and inflates the share price by calling FundsFacet.deposit(victim_deposit_amount * (1 + 10 ** DECIMALS_OFFSET), projectId, address(ERC4626Plugin)), increasing totalAssets without increasing totalSupply.
  3. The victim deposits victim_deposit_amount via ERC4626Plugin.deposit() and receives 0 shares at the inflated price (due to rounding).
  4. The attacker calls ERC4626Plugin.redeem(1, hacker, hacker) and withdraws the entire pool balance (their donation + the victim's deposit + 1).

The cost for the attacker grows exponentially with DECIMALS_OFFSET — the difference between ERC4626Plugin.decimals() and the underlying token's decimals. The most realistic setting for this attack is when the underlying token has 18 decimals (e.g., WETH, BNB, DAI).
This vulnerability has **Critical** severity because it allows an attacker to steal the first depositor's funds.

Recommendation
We recommend adopting the virtual-share accounting used by ERC4626Upgradeable._convertToShares() when calculating shares to burn on redeem()/withdraw(). This makes the described inflation attack economically unattractive, limiting it to expensive griefing. Additionally, disallow minting/burning of 0 shares in deposit(), mint(), redeem(), and withdraw().

Client's Commentary:
Acknowledged and fixed in PR-29

2.2 High

Not Found

2.3 Medium

M-1	Fee-on-Transfer Tokens Incompatibility in Deposit Flow		
Severity	Medium	**Status**	Acknowledged

Description
The deposit function in FundsFacet performs two separate operations: first transferring tokens from the user to the contract, and then depositing the same amount into strategies. This pattern assumes that the amount transferred equals the amount received, which is not true for fee-on-transfer tokens. When a user deposits fee-on-transfer tokens, the contract receives less than the stated amount due to the transfer fee, but then attempts to deposit the full original amount into strategies. This discrepancy will cause the strategy deposit to fail or result in incorrect accounting, as the contract doesn't have sufficient balance to cover the full deposit amount.

Recommendation
We recommend modifying the deposit flow to measure the actual balance change before and after the token transfer, and use the actual received amount for subsequent operations. Alternatively, explicitly document that fee-on-transfer tokens are not supported and add validation to reject such tokens.

Client's Commentary:
Acknowledged, will be not fixed. Yelay protocol doesn't support fee on transfer tokens.

M-2	Division by Zero Risk with Zero Total Assets		
Severity	Medium	**Status**	Acknowledged

Description
The _convertToShares function in FundsFacet can encounter a division by zero error when newTotalAssets equals zero while newTotalSupply is non-zero. This scenario can occur when a strategy is compromised, exploited, or contains a bug that causes it to report zero assets when shares still exist in the system. The function performs assets.mulDiv(newTotalSupply, newTotalAssets) without checking if newTotalAssets is zero, leading to a revert. This would effectively freeze the entire vault, preventing all deposits, withdrawals, and other operations that depend on share conversion calculations, resulting in a complete denial of service.

Recommendation
We recommend adding a require statement to check that newTotalAssets is greater than zero before performing the division operation. Additionally, consider implementing emergency pause mechanisms that automatically trigger when critical invariants like totalAssets becoming zero are detected, and add comprehensive strategy health monitoring to prevent such scenarios.

Client's Commentary:
Acknowledged, will be not fixed. We have a pausing functionality on all critical write functions and are using circle breaker.

2.4 Low

L-1	Missing Zero Address and Input Validation Checks		
Severity	Low	**Status**	Acknowledged

Description
The deposit and redeem functions in FundsFacet lack validation to ensure the receiver address is not the zero address. Similarly, the ERC4626PluginFactory's deploy and deployDeterministically functions do not validate input parameters such as the vault address and project ID, potentially allowing deployment with invalid configurations. These missing validations could lead to plugins being deployed in unusable states, causing user funds to become permanently locked or inaccessible.

Recommendation
We recommend adding require statements to validate that receiver addresses are not zero in deposit and redeem functions, and adding comprehensive input validation in factory functions to ensure all parameters are valid before deployment.

Client's Commentary:
Acknowledged, will be not fixed since it doesn't add any value.

L-2	Code Quality Issues and Potential Silent Failures		
Severity	Low	**Status**	Fixed in 4ae1252b

Description

Several code quality issues exist throughout the codebase that reduce maintainability. "Merkle" is incorrectly spelled as "merkl" in multiple locations including variable names and comments. There is an unused import of `SafeTransferLib` and `ERC20` contracts in `ERC4626Plugin` that adds unnecessary bytecode. The `maxWithdraw` function in `ERC4626Plugin` may revert silently when `super.maxWithdraw(owner)` returns a value smaller than `WITHDRAW_MARGIN`, resulting in a confusing underflow revert instead of a clear error message. These issues, while not directly exploitable, reduce code clarity and could lead to debugging difficulties or unexpected behavior in edge cases.

Recommendation

We recommend correcting the spelling of "Merkle" throughout the codebase, removing unused imports to reduce contract size and improve clarity, and adding explicit bounds checking in `maxWithdraw` to revert with a clear error message when the calculated result would underflow, rather than allowing silent arithmetic failures.

Client's Commentary:

Acknowledged, unnecessary SafeTransferLib is removed and maxWithdraw is adjusted in PR-29.

Merkl will be not fixed since semantically we are referring to the service https://merkl.xyz/.

L-3	Incorrect Zero Floor Subtraction in Decimal Offset Calculation		
Severity	Low	**Status**	Acknowledged

Description

The ERC4626Plugin initializer uses zeroFloorSub for calculating DECIMALS_OFFSET when computing the difference between 18 decimals and the underlying asset's decimals. The zeroFloorSub function returns zero when the subtraction would result in a negative number, which masks an error condition. If the underlying asset has more than 18 decimals, the offset calculation would silently return zero instead of reverting or handling the case appropriately. This incorrect offset would be stored in the contract and used in shares to assets ratio calculation. While not affecting the calculations itself, this case should be handled explicitly.

Recommendation

We recommend replacing zeroFloorSub with a standard subtraction operation and adding a require statement to explicitly check that the underlying asset has 18 or fewer decimals before performing the calculation. This ensures that unsupported token configurations are caught during initialization rather than causing silent failures.

Client's Commentary:

Client: Not acknowledged. Honestly, don't understand the issue at all. Even for 18 decimals tokens offset is 0. For any potential "exotic" token with decimals > 18 it is safe to use offset of 0 as well.

MixBytes: In this finding, we refer to the semantic meaning of the DECIMALS_OFFSET variable. In the case where an ERC20 token with more than 18 decimals is used, this variable won't actually store the offset between 18 (used as a constant in the code) and the configured token decimals. However, it doesn't affect the calculations where DECIMALS_OFFSET is used, so this is more of a code style improvement recommendation.

L-4	Incorrect Slippage Check in ERC4626Plugin _withdraw Function		
Severity	Low	**Status**	Fixed in ec533212

Description
The slippage check in ERC4626Plugin is incorrect because assetsBalance already includes the assetsReceived amount that was just withdrawn from YelayLiteVault. The current check assetsReceived + assetsBalance >= assets essentially verifies that assetsBalance >= assets since assetsReceived is already part of assetsBalance, which doesn't provide meaningful slippage protection and double-counts the received assets.

Recommendation
We recommend capturing the asset balance before calling yelayLiteVault.redeem(), then comparing it with the balance after the redemption to calculate the actual assets received. The slippage check should verify that the difference between the balances (the actual amount received) is greater than or equal to the requested assets amount.

Client's Commentary:
Acknowledged, fixed in ec5332121399f81ee988d1d3497179d551c7154e. We want to make sure there are enough assets on the plugin itself, since redemption from Yelay Lite Vault slightly overcharges. Please check two edge cases - test_WithdrawSlippageExceeded and test_last_withdraw.

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