MixBytes()

Mantle mETH x Aave Integration Security Audit Report

NOVEMBER 13, 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

mETH combines Ethereum staking with an Aave-backed liquidity buffer and an oracle-driven accounting model. Therefore, in this audit, we paid special attention to exchange-rate integrity, liquidity routing, oracle pitfalls, access control, and operational safety across the scope.

We also went through our detailed checklist covering business logic, ERC-20 patterns, interactions with external contracts, integer over/underflows, reentrancy, access control, typecasts, rounding, and other common pitfalls.

We found two notable issues: share-price calculations lack oracle freshness checks, and the current unstake design skews the distribution of losses and rewards.

Other notes and recommendations:

- `Staking.ethToMETH()` applies `exchangeAdjustmentRate`, favoring early depositors and enabling deposit front-running with immediate redemption profit in an empty pool. The problem is not applicable as mETH pool is not empty at the time of writing.
- Sudden changes to `exchangeAdjustmentRate` can be sandwiched (e.g., 0→positive or vice versa), reordering other users' deposits to induce slippage; for non-empty pools and low rates the profit is negligible; not applicable to the current pool.
- The admin can change `exchangeAdjustmentRate` at any time, which may be unexpected for users. We recommend pre-announcing upcoming changes in the UI.
- There are minor typos and unused declarations (no security impact) — typos in comments ("Inititalizes", "undesireable", "exisiting"), role seed uses "ALLOCATER_SERVICE_ROLE"; unused items: `LiquidityBuffer.onlyStakingContract` modifier, `PositionManager` `Borrow`/`Repay` events, and unused imports (`DataTypes`, `IERC20`, `UnstakeRequest`, non-upgradeable `AccessControlEnumerable`).

1.3 Project Overview

Summary

Title	Description
Client Name	Mantle
Project Name	mETH
Type	Solidity
Platform	EVM
Timeline	22.10.2025–11.11.2025

Scope of Audit

File	Link
src/liquidityBuffer/LiquidityBuffer.sol	LiquidityBuffer.sol
src/liquidityBuffer/PositionManager.sol	PositionManager.sol
src/Pauser.sol	Pauser.sol
src/Staking.sol	Staking.sol

Versions Log

Date	Commit Hash	Note
22.10.2025	6210e907b0f790ee9e11fe8ccb4d4baf12de6609	Initial Commit
06.11.2025	93280383c0858c559270ceaec6f5d04f9be0a8e7	Commit for re-audit

Mainnet Deployments

LiquidityBuffer.sol (0x38F319...d858aFdC) was deployed from commit 6210e907b0f790ee9e11fe8ccb4d4baf12de6609, in which M-2 wasn't fixed.

File	Address	Blockchain
TransparentUpgradeableProxy.sol	0x006FaD...83Dad409	Ethereum Mainnet
LiquidityBuffer.sol	0x38F319...d858aFdC	Ethereum Mainnet
TransparentUpgradeableProxy.sol	0xb48420...adbc11BC	Ethereum Mainnet
PositionManager.sol	0x7a65b2...e49dbb98	Ethereum Mainnet

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	6
Low	3

Findings Statuses

ID	Finding	Severity	Status
M-1	Default manager deactivation in LiquidityBuffer can break auto-allocation	Medium	Acknowledged
M-2	Inactive managers may under-report funds	Medium	Fixed
M-3	Front-running Staking.unstakeRequestWithPermit() can invalidate user transaction	Medium	Acknowledged
M-4	Emergency admin can transfer aWETH from the PositionManager	Medium	Acknowledged
M-5	Missing freshness check on oracle data in Staking.totalControlled() enables stale-rate arbitrage	Medium	Acknowledged
M-6	Fixed exchange rate at unstaking fails to socialize slashing and distorts rewards	Medium	Acknowledged
L-1	Linear manager check may cause out-of-gas	Low	Acknowledged
L-2	Last-withdrawal inflation attack	Low	Acknowledged
L-3	No upper bound on setNumberOfBlocksToFinalize() may delay finalization	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

M-1	Default manager deactivation in LiquidityBuffer can break auto-allocation		
Severity	Medium	**Status**	Acknowledged

Description

Staking.allocateETH() forwards ETH to LiquidityBuffer.depositETH(), which auto-allocates to defaultManagerId when shouldExecuteAllocation is true. If the current default manager is deactivated via LiquidityBuffer.updatePositionManager()/togglePositionManagerStatus() without first switching defaultManagerId or disabling auto-allocation, _allocateETHToManager(defaultManagerId, …) reverts with LiquidityBuffer__ManagerInactive(). This can unexpectedly block allocation flows.

```
// LiquidityBuffer auto-allocates
// to defaultManagerId if enabled
function depositETH() external payable
onlyRole(LIQUIDITY_MANAGER_ROLE) {
  _receiveETHFromStaking(msg.value);
  if (shouldExecuteAllocation) {
    _allocateETHToManager(
        defaultManagerId,
        msg.value);
      // reverts if default manager inactive
    }
}
```

LiquidityBuffer.sol#L334-L340

Recommendation

We recommend preventing deactivation of the current defaultManagerId while auto-allocation is enabled: require shouldExecuteAllocation == false or switch defaultManagerId to an active manager before deactivation.

Client's Commentary:

Not fixing; operators should disable auto-allocation or switch the default manager before deactivating the default manager. We have established operational guidelines and monitoring systems to ensure correct procedures. Without additional checks, the system can remain more flexible in emergency situations. The revert will not affect funds.

M-2	Inactive managers may under-report funds		
Severity	Medium	**Status**	Fixed in 93280383

Description

`LiquidityBuffer.getControlledBalance()` sums balances of managers with `isActive == true` only. If an admin deactivates a manager via `updatePositionManager()`/`togglePositionManagerStatus()` before evacuating funds, any residual underlying on that manager is excluded from the total, causing under-reporting. Moreover, withdrawals from inactive managers are blocked (`_withdrawETHFromManager` and `onlyPositionManagerContract` both require the manager to be active), so the funds remain inaccessible until the manager is reactivated.

```
function getControlledBalance() public view returns (uint256) {
    ...
    for (uint256 i = 0; i < positionManagerCount; i++) {
        ...
        if (config.isActive) {
            ...
            uint256 managerBalance = manager.getUnderlyingBalance();
            totalBalance += managerBalance;
            ...
        }
    }
    ...
}
```

LiquidityBuffer.sol#L183-L199

Recommendation

We recommend checking underlying balance before deactivation and using a dedicated `forceDeactivate()` if it's not zero.

Client's Commentary:

Fixed in 93280383c0858c559270ceaec6f5d04f9be0a8e7

M-3	Front-running Staking.unstakeRequestWithPermit() can invalidate user transaction		
Severity	Medium	**Status**	Acknowledged

Description

An attacker can read the v, r, s from the mempool and call mETH.permit() first, consuming the user's signature and updating the nonce. The subsequent user transaction Staking.unstakeRequestWithPermit() then reverts on the permit() call due to an invalidated signature/nonce, preventing the unstake flow from executing, even though allowance is already set.

```
function unstakeRequestWithPermit
    ...
    SafeERC20Upgradeable.safePermit(
        mETH,
        msg.sender,
        address(this),
        methAmount,
        deadline,
        v,
        r,
        s);
```

Staking.sol#L362-L372

Recommendation

We recommend wrapping the permit() invocation in a try-catch clause and proceeding with the unstake if allowance is already sufficient, avoiding a hard revert when signatures are pre-consumed.

Client's Commentary:

We acknowledge the issue. It will not be fixed in this version. The `unstakeRequestManager` contract will not be upgraded separately at this time; the fix will be included in a future major contract upgrade.

M-4	Emergency admin can transfer aWETH from the PositionManager		
Severity	Medium	**Status**	Acknowledged

Description

EMERGENCY_ROLE in PositionManager can transfer aWETH from the contract to any address without restrictions, which may be unintended and creates additional risks in case this role is compromised.

Recommendation

We recommend disallowing aWETH emergency transfers or approvals from the PositionManager contract, since withdrawals are already handled via the more transparent withdraw() flow.

Client's Commentary:

This is an emergency function intended for rapid fund withdrawal in critical situations. The EMERGENCY_ROLE will not be configured at launch. It will be assigned only to the highest level of multisig authority.

M-5	Missing freshness check on oracle data in `Staking.totalControlled()` enables stale-rate arbitrage		
Severity	Medium	**Status**	Acknowledged

Description

`Staking.totalControlled()` derives the mETH/ETH exchange rate inputs from `oracle.latestRecord()` without validating the record timestamp.

If the oracle lags significant state changes (e.g., validator rewards or slashing), the resulting rate becomes stale. An attacker can exploit this by timing mint/burn operations against outdated totals: redeeming mETH for excess ETH when a slashing is not yet reflected (overstated `totalControlled()`), or depositing ETH to mint excess mETH when recent rewards are not yet reflected (understated `totalControlled()`), extracting value from other users.

`Staking.sol#L596-L611`

Recommendation

We recommend enforcing freshness validation for oracle records when minting or burning mETH.

Client's Commentary:

***Client**: Most sanity checks are in the Oracle contract, ensuring consistency.*

***MixBytes**: The `Oracle.latestRecord()` function has no sanity checks—it simply returns `_records[_records.length - 1]`. This is correct, as the function is not supposed to perform any checks. The validations should be implemented on the caller side, specifically in the `Staking.totalControlled()` function.*

***Client**: the issue is known; won't fix;*

***MixBytes**: we accept the decision since changing a high-TVL contract is risky and the oracle's failure risk is low.*

M-6	Fixed exchange rate at unstaking fails to socialize slashing and distorts rewards		
Severity	Medium	**Status**	Acknowledged

Description

When `Staking.unstakeRequest()` is called, the mETH/ETH rate is fixed and does not reflect slashing or rewards that may occur by the time `Staking.claimUnstakeRequest()` is executed. If two users create requests concurrently and losses arrive afterward, those losses are not socialized across them. One request may be fully paid while the other may revert on claim due to insufficient allocated funds.

This can be exacerbated by frontrunning updates to `LiquidityBuffer.cumulativeDrawdown()`, enabling informed actors to anticipate loss application.

Rewards are also misallocated: requests lock mETH but do not burn it until `UnstakeRequestsManager.claim()`, so these shares continue participating in reward distribution, diluting rewards for users who have not requested to unstake.

Example: Alice and Bob each hold 100 mETH and the pool controls 200 ETH. Alice submits `Staking.unstakeRequest()` for 100 mETH; the shares are locked but not burned. A subsequent 100 ETH reward is then distributed across 200 mETH, so only ~50 ETH is attributed to Bob instead of ~100 ETH. This dilution persists until Alice calls `UnstakeRequestsManager.claim()` and the 100 mETH are burned.

Recommendation

We recommend aligning withdrawal settlement with the latest protocol state to socialize slashing and adjusting reward accounting so pending unstakes do not accrue or dilute rewards.

Client's Commentary:

***Client**: Not fixing: this is a protocol design trade-off.*

- *Users can predict their returns when submitting unstake requests*
- *unstakeRequestManager also supports the request cancellation and refund mechanism.*
- *This design has proven effective under normal market conditions*
- *This issue only occurs under extreme market conditions (e.g., significant slashing events)*
  - *The protocol has robust monitoring and alert systems to detect such scenarios*
  - *The affected scope is limited to users who submit unstake requests during the same period*
- *Fixing this issue would require restructuring core unstaking mechanisms, resulting in high development costs*

***MixBytes**: The probability of a significant slashing event leading to net losses not offset by rewards is low, and no such event has occurred to date.*

We acknowledge the design choice to fix the rate at unstake request time: it improves predictability but can, in rare cases, create timing advantages and uneven distribution of losses/rewards.

We recommend clearly documenting this trade-off for users; an explicit compensation or `topUp()` policy would further reduce residual risk.

2.4 Low

L-1	Linear manager check may cause out-of-gas		
Severity	Low	**Status**	Acknowledged

Description

`LiquidityBuffer.onlyPositionManagerContract` performs a cycle scan over all registered managers on every call. As the number of managers increases, this check may cause out-of-gas error.

```
modifier onlyPositionManagerContract() {
    ...
    for (uint256 i = 0; i < positionManagerCount; i++) {
        ...
    }
    ...
}
```

LiquidityBuffer.sol#L545-L563

Recommendation

We recommend enforcing a bounded number of managers.

Client's Commentary:

Not fixing; we'll have a low number of positionManagers, and currently just the one AAVE positionManager

L-2	Last-withdrawal inflation attack		
Severity	Low	**Status**	Acknowledged

Description

If the protocol is left with a single deposit and its funds reside on a validator, the depositor can request to withdraw all but 1 wei of shares. Because rewards accrue on the validator during the withdrawal process and settlement uses a prior share price, the protocol may end up with 1 wei of shares plus accrued ETH after the claim. This can drive the share price to an extreme value (e.g., ~$1,000 per 1 wei with ~5% APR and a one-week delay), enabling an inflation attack against subsequent deposits.

Recommendation

We recommend keeping a small protocol-owned deposit to prevent any external depositor from becoming the last holder; alternatively, enforce withdrawal rules ensuring the contract ends with either zero shares or a minimum-share threshold controlled by an admin.

Client's Commentary:

Not fixing; this is a theoretical attack with minimal actual risk. The protocol remains sufficiently liquid at all times that a single depositor situation is unlikely

L-3	No upper bound on setNumberOfBlocksToFinalize() may delay finalization		
Severity	Low	**Status**	Acknowledged

Description

`setNumberOfBlocksToFinalize()` has no upper bound; misconfiguration can delay request finalization excessively.

UnstakeRequestsManager.sol#L366-L375

Recommendation

We recommend adding a reasonable limit to prevent excessive delays.

Client's Commentary:

Not fixing; we have a strict governance procedures to regulate such changes, and we have established a monitoring system to detect abnormal configurations

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