MixBytes()

Aave v3.7 Security Audit Report

MARCH 30, 2026

Table of Contents

1. Introduction	**2**
1.1 Disclaimer	2
1.2 Executive Summary	2
1.3 Project Overview	4
1.4 Security Assessment Methodology	7
1.5 Risk Classification	9
1.6 Summary of Findings	10
2. Findings Report	**11**
2.1 Critical	11
2.2 High	11
2.3 Medium	11
2.4 Low	11
L-1 Unused Imports and using Directives Across Protocol Contracts	11
3. About MixBytes	**13**

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

Aave is a decentralized liquidity protocol that allows users to supply assets to earn yield or borrow against posted collateral.

Aave v3.7 removes the legacy Isolation Mode and Siloed Borrowing features from the core protocol. Reserve-level debt ceilings, borrow-in-isolation flags, siloed borrowing flags, and isolated debt accounting are deprecated and no longer participate in runtime validation. Their intended risk-management role is replaced by the more flexible eMode framework introduced in v3.6. In particular, v3.7 adds an `isolated` flag to eMode categories: when enabled, only assets explicitly included in the category's `collateralBitmap` contribute borrowing power, while all other supplied assets are treated as `LTV = 0` for borrowing purposes, without affecting their liquidation threshold contribution.

The release also removes the unused `dropReserve()` flow from `Pool`, `PoolConfigurator`, and the associated validation paths. This avoids unsafe reserve ID reuse and eliminates dead code that was not realistically usable in production. For backwards compatibility, legacy data-provider surfaces remain available and return disabled values for removed features such as `debtCeiling`, `siloedBorrowing`, and `isolationModeTotalDebt`.

Liquidation math was updated to use explicit floor/ceil rounding in favor of deterministic outcomes. The total collateral seized is rounded down, the protocol-fee base is derived with floor rounding, and the fee itself is rounded up. This removes the previous half-up rounding behavior that could be gamed by liquidators through carefully chosen repayment amounts.

The `PriceOracleSentinel` and `SequencerOracle` contracts, previously used on L2 deployments to gate borrows and liquidations behind sequencer uptime checks, are removed entirely. This represents a shift from a fail-safe design toward a fail-open model for L2 deployments. Previously, borrows were blocked while the sequencer was down or within a post-recovery grace period, and liquidations were similarly gated with a special bypass allowing liquidation when a position's health factor fell below 0.95 regardless of sentinel status. Both checks are now eliminated – borrows and liquidations proceed without sequencer-awareness gating.

The stated rationale is the elimination of false-positive sequencer-down signals, which varied in reliability and behavior across L2 networks due to the absence of a standardized uptime reporting mechanism. False positives that blocked liquidations were particularly damaging to protocol health: they allowed unhealthy positions to deteriorate during the

exact window when liquidation was most needed, increasing exposure to bad debt accumulation. The 0.95 HF bypass was an acknowledgment of this danger, but it only covered severely undercollateralized positions and could be problematic in high-liquidation-threshold scenarios.

However, the removal reintroduces exposure to stale oracle conditions during sequencer downtime and recovery phases. On L2 rollups, oracle price feed updates are sequencer-dependent — when the sequencer is unavailable, oracle providers such as Chainlink cannot push new price data on-chain, causing feeds to silently serve the last known price rather than reverting. While Chainlink feeds expose an `updatedAt` timestamp that permits staleness detection, Aave's oracle layer does not enforce a maximum staleness threshold at the protocol level, meaning stale prices are consumed without revert. During such windows, users can borrow against outdated (potentially inflated) collateral valuations, effectively obtaining undercollateralized loans. Simultaneously, asymmetric liquidation opportunities arise for sophisticated actors with access to L1 forced-inclusion mechanisms (available on optimistic rollups), who can submit transactions during sequencer downtime while regular users cannot, exploiting stale pricing ahead of the broader market.

The removal of the sentinel's post-recovery grace period further increases sensitivity to race conditions immediately after sequencer restart. Without this buffer, the protocol is immediately operational while oracle feeds may still be propagating updates, creating a transient window where partial or delayed price refreshes can lead to inconsistent protocol state and opportunistic liquidations against positions that would not be underwater under current market pricing. The per-reserve `liquidationGracePeriodUntil` remains in place but serves a different purpose (protecting against liquidations during reserve configuration changes) and unrelated to sequencer-recovery scenarios. The `PoolAddressesProvider` retains its sentinel getter and setter for ABI compatibility across deployments, but the stored address is no longer read by the Pool.

Given this shift, we recommend that Aave governance establishes off-chain monitoring of sequencer uptime for each L2 deployment, with predefined operational runbooks to manually pause borrowing via the emergency admin role (`setReservePause`, `setReserveFreeze`, `setPoolPause`, `setReserveBorrowing`) when prolonged sequencer downtime is detected. This would preserve the fail-open benefit for short-lived or spurious outages while providing a manual safety net against sustained downtime scenarios where oracle staleness becomes material.

The protocol's L2 security model now relies entirely on oracle-provider-level freshness guarantees and off-chain monitoring, with no on-chain sequencer-awareness fallback. This is a deliberate trade-off: the documented history of false positives causing blocked liquidations and bad debt accumulation is weighed against the narrower but real stale-price exploitation window during sequencer outages. The risk profile shifts from frequent, moderate-impact false-positive events to rare but potentially high-impact market desynchronization events during L2 sequencer failures.

The audit was conducted by 3 auditors over 2.5 days, through manual review of the scoped changes, and targeted compilation and test execution for the new isolated-eMode and liquidation-rounding paths.

During the audit, the following attack vectors and regression risks were checked:

- It was verified that removing isolation-mode and siloed-borrowing validations does not leave partially active enforcement paths in borrow, repay, supply, liquidation, or configuration flows.

- Examination confirms that `ValidationLogic.getUserReserveLtv()` correctly assigns `LTV = 0` to assets outside the `collateralBitmap` when an eMode category is marked `isolated`, while preserving existing ltvzeroBitmap behavior for assets inside the bitmap.
- The computation of `avgLtv`, `hasZeroLtvCollateral`, and `avgLiquidationThreshold` was verified under the new isolated eMode semantics.
- The removal of `dropReserve()` was reviewed and confirmed to eliminate the reserve-ID reuse path while preserving legacy gap-handling checks in reserve-iteration logic.
- The liquidation rounding changes were reviewed for internal consistency and confirmed to remove the liquidator's ability to bias rounding in their favor across multiple computation steps.
- Targeted tests were implemented for isolated eMode configuration and toggling, isolated collateral enforcement, and liquidation invariants under the updated rounding logic.

The scoped changes are coherent, materially reduce protocol complexity, and are generally aligned with the accompanying documentation. The main security considerations shift from reserve-level isolation controls to correct governance use of the new isolated-eMode configuration, and from on-chain sequencer-awareness gating to full reliance on oracle-provider-level freshness guarantees and off-chain monitoring for L2 deployments.

1.3 Project Overview

Summary

Title	Description
Client	Aave DAO
Project	Aave v3.7
Code developed by	BGD Labs
Category	Lending
Type	Solidity
Platform	EVM
Timeline	06.03.2026 - 26.03.2026

Scope of Audit

File	Link
src/contracts/protocol/libraries/configuration/EModeConfiguration.sol	src/contracts/protocol/libraries/configuration/EModeConfiguration.sol
src/contracts/protocol/libraries/configuration/ReserveConfiguration.sol	src/contracts/protocol/libraries/configuration/ReserveConfiguration.sol
src/contracts/protocol/libraries/configuration/UserConfiguration.sol	src/contracts/protocol/libraries/configuration/UserConfiguration.sol
src/contracts/protocol/libraries/logic/BorrowLogic.sol	src/contracts/protocol/libraries/logic/BorrowLogic.sol
src/contracts/protocol/libraries/logic/ConfiguratorLogic.sol	src/contracts/protocol/libraries/logic/ConfiguratorLogic.sol
src/contracts/protocol/libraries/logic/FlashLoanLogic.sol	src/contracts/protocol/libraries/logic/FlashLoanLogic.sol
src/contracts/protocol/libraries/logic/GenericLogic.sol	src/contracts/protocol/libraries/logic/GenericLogic.sol
src/contracts/protocol/libraries/logic/LiquidationLogic.sol	src/contracts/protocol/libraries/logic/LiquidationLogic.sol
src/contracts/protocol/libraries/logic/PoolLogic.sol	src/contracts/protocol/libraries/logic/PoolLogic.sol
src/contracts/protocol/libraries/logic/ReserveLogic.sol	src/contracts/protocol/libraries/logic/ReserveLogic.sol
src/contracts/protocol/libraries/logic/SupplyLogic.sol	src/contracts/protocol/libraries/logic/SupplyLogic.sol
src/contracts/protocol/libraries/logic/ValidationLogic.sol	src/contracts/protocol/libraries/logic/ValidationLogic.sol
src/contracts/protocol/libraries/types/DataTypes.sol	src/contracts/protocol/libraries/types/DataTypes.sol
src/contracts/protocol/libraries/helpers/Errors.sol	src/contracts/protocol/libraries/helpers/Errors.sol
src/contracts/protocol/libraries/math/PercentageMath.sol	src/contracts/protocol/libraries/math/PercentageMath.sol

File	Link
src/contracts/protocol/pool/Pool.sol	src/contracts/protocol/pool/Pool.sol
src/contracts/protocol/pool/PoolConfigurator.sol	src/contracts/protocol/pool/PoolConfigurator.sol
src/contracts/protocol/tokenization/AToken.sol	src/contracts/protocol/tokenization/AToken.sol
src/contracts/protocol/tokenization/ATokenWithDelegation.sol	src/contracts/protocol/tokenization/ATokenWithDelegation.sol
src/contracts/protocol/tokenization/VariableDebtToken.sol	src/contracts/protocol/tokenization/VariableDebtToken.sol
src/contracts/protocol/tokenization/base/IncentivizedERC20.sol	src/contracts/protocol/tokenization/base/IncentivizedERC20.sol
src/contracts/protocol/tokenization/base/MintableIncentivizedERC20.sol	src/contracts/protocol/tokenization/base/MintableIncentivizedERC20.sol
src/contracts/protocol/tokenization/delegation/BaseDelegation.sol	src/contracts/protocol/tokenization/delegation/BaseDelegation.sol

Versions Log

Date	Commit Hash	Note
06.03.2026	922e2258c516aa72790c9fbfa1af41d0bfa4596f	Initial Commit
19.03.2026	dd1077b902ce9b990f671af3f4a4bcbf888eecd3	Commit With Fixes And Sentinel Removal
26.03.2026	75145ad021c123a14d9190c67bd30e6867e22cb7	Commit With Liquidation Dust Rounding Fix

Mainnet Deployments

The deployment verification will be conducted later after the full deployment of the protocol into the 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	0
Low	1

Findings Statuses

ID	Finding	Severity	Status
L-1	Unused Imports and using Directives Across Protocol Contracts	Low	Acknowledged

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Unused Imports and `using` Directives Across Protocol Contracts		
Severity	Low	**Status**	Acknowledged

Description

Multiple files under `src/contracts/protocol` contain dead `import` and `using` directives — library methods that are never invoked, types never referenced, or structurally inert bindings. All were verified against the full inheritance trees where applicable.

Structurally dead `using ReserveLogic for DataTypes.ReserveCache` — every `ReserveLogic` function takes `DataTypes.ReserveData storage` as its first parameter; none accept `ReserveCache`, so the directive attaches zero methods:

- `src/contracts/protocol/libraries/logic/BorrowLogic.sol`
- `src/contracts/protocol/libraries/logic/FlashLoanLogic.sol`
- `src/contracts/protocol/libraries/logic/LiquidationLogic.sol`
- `src/contracts/protocol/libraries/logic/SupplyLogic.sol`

`ValidationLogic.sol` — import + `using` pairs with no corresponding method calls (`ReserveLogic` L13/L22):

- `src/contracts/protocol/libraries/logic/ValidationLogic.sol`, `src/contracts/protocol/libraries/logic/ValidationLogic.sol`

`BorrowLogic.sol` — dead `SafeCast` (L5/L29) and `ReserveConfiguration` (L12/L28):

- `src/contracts/protocol/libraries/logic/BorrowLogic.sol`, `src/contracts/protocol/libraries/logic/BorrowLogic.sol`, `src/contracts/protocol/libraries/logic/BorrowLogic.sol`

`FlashLoanLogic.sol` — dead `ReserveConfiguration` (L12/L30):

- `src/contracts/protocol/libraries/logic/FlashLoanLogic.sol`, `src/contracts/protocol/libraries/logic/FlashLoanLogic.sol`

`SupplyLogic.sol` — dead `PercentageMath` (L11/L29) and `ReserveConfiguration` (L14/L27):

- src/contracts/protocol/libraries/logic/SupplyLogic.sol,
  src/contracts/protocol/libraries/logic/SupplyLogic.sol,
  src/contracts/protocol/libraries/logic/SupplyLogic.sol,
  src/contracts/protocol/libraries/logic/SupplyLogic.sol

ReserveLogic.sol – dead IERC20/GPv2SafeERC20 (L4–L5/L28) and self-referential using ReserveLogic for DataTypes.ReserveData (L29) with no internal member-access calls:
- src/contracts/protocol/libraries/logic/ReserveLogic.sol,
  src/contracts/protocol/libraries/logic/ReserveLogic.sol

PoolStorage.sol – three import + using pairs (L4–L6/L16–L18) in a pure storage contract with no function bodies; verified unused through Pool → L2Pool → all leaf instances:
- src/contracts/protocol/pool/PoolStorage.sol, src/contracts/protocol/pool/PoolStorage.sol

UserConfiguration.sol – dead using ReserveConfiguration for DataTypes.ReserveConfigurationMap (L15); the import on L7 is still needed for ReserveConfiguration.MAX_RESERVES_COUNT:
- src/contracts/protocol/libraries/configuration/UserConfiguration.sol

Tokenization – dead using WadRayMath for uint256 with zero .rayMul()/.rayDiv() etc. calls across the entire 11-contract inheritance tree (IncentivizedERC20 → … → all leaf instances):
- src/contracts/protocol/tokenization/base/IncentivizedERC20.sol,
  src/contracts/protocol/tokenization/base/IncentivizedERC20.sol
- src/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol,
  src/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol
- src/contracts/protocol/tokenization/ATokenWithDelegation.sol,
  src/contracts/protocol/tokenization/ATokenWithDelegation.sol

VariableDebtToken.sol – dead using SafeCast for uint256 (L24); no .toUint*() calls in this contract or its children, and parent contracts carry their own declarations:
- src/contracts/protocol/tokenization/VariableDebtToken.sol,
  src/contracts/protocol/tokenization/VariableDebtToken.sol

Recommendation

We recommend removing all dead import and using directives listed above to reduce bytecode size and improve code clarity.

Client's Commentary:

We acknowledge the finding and confirm that these unused `import` and `using` directives have no impact on functionality or security. Given their negligible effect and to avoid unnecessary re-audit overhead, we have chosen to defer cleanup to a future refactor.

3. About MixBytes

MixBytes is a leading provider of smart contract auditing and blockchain security research, helping Web3 projects enhance their security and reliability.

Since its inception, MixBytes has been dedicated to safeguarding innovation in **DeFi** through rigorous audits and cutting-edge technical research.

Our team brings together experienced engineers, security auditors, and blockchain researchers with deep expertise in smart contract security and protocol design.

With years of proven experience in Web3, MixBytes combines in-depth technical excellence with a proactive, security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by leading blockchain protocols such as **Lido**, **Aave**, **Curve**, **1inch**, **Fluid**, **Gearbox**,**Resolv**, and others. MixBytes has successfully secured billions of dollars in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems, backed by years of hands-on experience.
- **Innovative Research:** MixBytes actively contributes to blockchain security research and open-source tools, sharing insights with the global community.

Our Services

- **Smart Contract Audits:** Comprehensive security assessments of DeFi protocols to detect and mitigate vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical studies, economic and protocol-level modeling for Web3 projects.
- **Custom Security Solutions:** Tailored frameworks and advisory for complex decentralized systems and blockchain ecosystems.

Contact Information

 https://mixbytes.io/

 https://github.com/mixbytes/audits_public

 https://x.com/MixBytes

 hello@mixbytes.io