MixBytes()

Aave v3.5 Security Audit Report

JULY 18, 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

Aave is a decentralized, non-custodial liquidity protocol that enables users to supply assets to earn interest or to borrow assets by providing collateral.

As part of the v3.5 upgrade, the protocol introduced refinements to the rounding logic applied in mathematical operations. The previous approach used a general half-up rounding method. The updated version adopts a more consistent, operation-specific strategy that explicitly defines whether to round up or down depending on the context.

The audit was carried out by 3 auditors over 8 days in total: 5 days for the interim audit and 3 days for the re-audit, involving a thorough manual code review of the scope and analysis via automated tools.

During the audit, in addition to verifying well-known attack vectors and items from our internal checklist, we thoroughly investigated the following areas:

- **Off-by-One Underflow Risk.** Since some code changes lead to certain values now being rounded differently (i.e., one unit more or less), there was a risk that, due to potentially unsynchronized updates in different parts of the code, a shortage of one unit could occur somewhere, causing unexpected behavior—primarily reverts that block legitimate transactions because of coding oversights. We inspected all code sections that could be affected by this.

- **Rounding Rules for Debt and Collateral.** When implementing changes, the developers followed the principle that debt should always be overestimated (i.e., rounded up) and collateral should always be underestimated (i.e., rounded down). We evaluated each rounding instance individually—considering its intent for protocol functionality—and verified whether this principle applies in every case or if there are situations where debt is more appropriately rounded down, or collateral needs to be rounded up.

- **Implicit Integer Division Rounding.** The developers focused on rounding related to the rayMul and rayDiv operations. However, standard integer division also implicitly rounds down. We reviewed these implicit roundings as well to ensure they align with the safest implementation logic.

- **Withdraw/Burn Underflow Check.** Special attention was paid to the withdraw/burn operation, since it can automatically determine how many funds to withdraw. We audited this code path

for potential underflows.

- **Interest Rate Index Consistency.** The project uses the common "index" pattern for dynamically accounting interest rates. We confirmed that the latest code changes did not break the correct updating logic of these indexes.

- **Math Library Verification.** The math libraries are a critically important component of the code. We audited them to ensure that all operations behave as expected.

- **Impact of Code Changes on HF Calculation.** We paid special attention to the code sections related to HF calculation to ensure that the developer's changes do not lead to unplanned liquidations or make it impossible to borrow or withdraw funds.

- **Dust Post-Liquidation.** The code contains a mechanism ensuring complete liquidation so that no "dust" remains for users. We confirmed that these mechanisms continue to function correctly and also verified that users with bad debt are still fully liquidatable under the updated logic.

Protocol-Favoring Rounding Recommendations

Below are two specific recommendations to ensure that all rounding behaviors consistently benefit the protocol and prevent any user from gaining an unintended advantage:

- We recommend rounding up user debt LiquidationLogic.sol#L604-L605 to ensure all rounding favors the protocol.

- We also recommend rounding down accrued debt during reward calculation for the treasury ReserveLogic.sol#L191-L193, maintaining consistent behavior even when the DAO is treated as a special user type.

1.3 Project Overview

Summary

Title	Description
Client Name	BGD Labs
Project Name	Aave v3.5
Type	Solidity
Platform	EVM
Timeline	30.05.2025 - 17.07.2025

Scope of Audit

File	Link
src/contracts/protocol/libraries/logic/GenericLogic.sol	GenericLogic.sol
src/contracts/protocol/libraries/logic/ReserveLogic.sol	ReserveLogic.sol
src/contracts/protocol/libraries/logic/PoolLogic.sol	PoolLogic.sol
src/contracts/protocol/libraries/logic/EModeLogic.sol	EModeLogic.sol
src/contracts/protocol/libraries/logic/LiquidationLogic.sol	LiquidationLogic.sol
src/contracts/protocol/libraries/logic/SupplyLogic.sol	SupplyLogic.sol
src/contracts/protocol/libraries/logic/ValidationLogic.sol	ValidationLogic.sol
src/contracts/protocol/libraries/logic/FlashLoanLogic.sol	FlashLoanLogic.sol
src/contracts/protocol/libraries/logic/BorrowLogic.sol	BorrowLogic.sol
src/contracts/protocol/libraries/math/PercentageMath.sol	PercentageMath.sol
src/contracts/protocol/libraries/math/WadRayMath.sol	WadRayMath.sol
src/contracts/protocol/libraries/math/MathUtils.sol	MathUtils.sol
src/contracts/protocol/tokenization/ATokenWithDelegation.sol	ATokenWithDelegation.sol
src/contracts/protocol/tokenization/AToken.sol	AToken.sol
src/contracts/protocol/tokenization/VariableDebtToken.sol	VariableDebtToken.sol

File	Link
src/contracts/protocol/tokenization/base/ScaledBalanceTokenBase.sol	ScaledBalanceTokenBase.sol
src/contracts/protocol/tokenization/base/IncentivizedERC20.sol	IncentivizedERC20.sol
src/contracts/protocol/libraries/types/DataTypes.sol	DataTypes.sol
src/contracts/protocol/pool/Pool.sol	Pool.sol
src/contracts/helpers/LiquidationDataProvider.sol	LiquidationDataProvider.sol
src/contracts/protocol/libraries/helpers/TokenMath.sol	TokenMath.sol

Versions Log

Date	Commit Hash	Note
30.05.2025	bea2689facc94d8787cbc36751d42d9248892207	Initial Commit
30.06.2025	c3e88e161b44de14f7cbbfd86749ffb8bb392260	Re-audit Commit
15.07.2025	10ddd1a59067425fd02a6269c6e2c3f91a9227ec	Second Re-audit Commit
17.07.2025	f76f00773016f35dc1a134f090afc5923c6bcbe1	Third Re-audit Commit

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	12

Findings Statuses

ID	Finding	Severity	Status
L-1	Incorrect User Reference in aToken-Based Repay Path	Low	Fixed
L-2	Imprecise Rounding in Treasury Premium Accrual	Low	Fixed
L-3	Unnecessary Parentheses in Collateral Liquidation Formula	Low	Fixed
L-4	Typo in Transfer Amount Comment	Low	Fixed
L-5	Rounding Logic May Allow Transfers Beyond Approved Amount	Low	Fixed
L-6	Incorrect Supply and Flashloan Check	Low	Acknowledged
L-7	Overly Strict Overflow Check	Low	Fixed
L-8	Suboptimal Rounding in Treasury Fee Calculation	Low	Fixed
L-9	Supply Cap Validation Sums Token Balances Instead of Shares	Low	Fixed
L-10	Liquidation Rounding May Favor Liquidators Over Protocol	Low	Fixed
L-11	Potential Underflow in Liquidation Logic	Low	Fixed

L-12	Misleading Comments in WadRayMath	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	Incorrect User Reference in aToken-Based Repay Path		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within the executeRepay function:

- BorrowLogic.sol#L152-L154

When calculating the paybackAmount using aToken balance (in the useATokens branch), the logic references params.onBehalfOf instead of params.user. While in current usage these two addresses are equal when useATokens == true, relying on onBehalfOf introduces unnecessary coupling and reduces clarity. This may lead to confusion or potential errors in future protocol upgrades.

The similar issue is present here:

- BorrowLogic.sol#L195-L197

The issue is classified as **Low** severity because it does not currently impact protocol correctness but may hinder maintainability.

Recommendation

We recommend replacing params.onBehalfOf with params.user in the aToken.balanceOf logic to reflect the true source of the funds and improve clarity.

Client's Commentary:

Addressed in 3774ca96359ee68fc9a99135b0aec48f5401a2dd

L-2	Imprecise Rounding in Treasury Premium Accrual		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within the `_handleFlashLoanRepayment` function:

- FlashLoanLogic.sol#L226-L229

The expression used to calculate the share of the premium allocated to the protocol treasury performs a `rayDiv` operation:

```
params.totalPremium.rayDiv(reserveCache.nextLiquidityIndex)
```

However, the rounding behavior of this division is not explicitly specified, which may result in **upward rounding**. This can cause the treasury to receive 1 wei more than expected. This issue is classified as **Low** severity because the over-minting is minimal (1 wei), but it represents an unfair distribution of funds and a potential source of precision-based inconsistencies.

Recommendation

We recommend using a utility function `rayDivFloor` to ensure predictable and fair distribution of value.

Client's Commentary:

Addressed in 62c8e5bfe45298da58d1d36b9b99a4fa18ed8530

L-3	Unnecessary Parentheses in Collateral Liquidation Formula		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within the `_calculateAvailableCollateralToLiquidate` function of the protocol:

- LiquidationLogic.sol#L597

The formula used to compute the base collateral amount includes excessive parentheses, as seen below:

```
((debtAssetPrice * debtToCover * collateralAssetUnit)) /
(vars.collateralAssetPrice * debtAssetUnit);
```

While functionally correct, the use of unnecessary brackets may reduce code readability.

Recommendation

We recommend simplifying the formula by removing redundant parentheses to improve readability:

```
(debtAssetPrice * debtToCover * collateralAssetUnit) /
(vars.collateralAssetPrice * debtAssetUnit);
```

Client's Commentary:

Addressed in 547589ed5fac7a5f13fa8625d3579b64054d867d

L-4	Typo in Transfer Amount Comment		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within the executeFinalizeTransfer function of the protocol:

- SupplyLogic.sol#L199

A comment above the logic describing the transfer amount contains a typo:

```
// to ensure the receiver, receives at least the intended **amoutn** of shares."
```

Recommendation

Correct the typo in the comment:

```
// to ensure the receiver receives at least the intended **amount** of shares."
```

Client's Commentary:

Addressed in 891341086441606bb194cc1da5580e74841cf351

L-5	Rounding Logic May Allow Transfers Beyond Approved Amount		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within the transfer logic of ATokens.

- SupplyLogic.sol#L200
- AToken.sol#L206-L208
- ATokenWithDelegation.sol#L97

Currently, the transfer amount is calculated using ceiling rounding via rayDivCeil, which can lead to unintuitive behavior when using approve. Specifically, a user who approved 1000 tokens might inadvertently allow the transfer of up to 1050 tokens due to rounding, which violates expectations of approval boundaries.
The issue is classified as **Low** severity because it affects UX and correctness of token transfer approvals but does not lead to critical protocol failure.

Recommendation

We recommend changing the rounding logic from **ceiling** to **floor** when calculating scaledAmount during transfer operations. This ensures that users cannot transfer more than they approved.

Client's Commentary:

Addressed in b7beed9ee6722cc8208131d61aae697ab4d711b3, by optimizing the overall logic to properly track allowance usage.

L-6	Incorrect Supply and Flashloan Check		
Severity	Low	**Status**	Acknowledged

Description

This issue has been identified within the validateBorrow and validateFlashloanSimple functions of the protocol:

- ValidationLogic.sol#L156-L159
- ValidationLogic.sol#L338

The presented checks incorrectly assume that the total supply of aTokens directly reflects the amount of underlying tokens available for borrowing. However, since borrowings reduce the underlying token balance without affecting aToken supply, this check may incorrectly pass depending on borrow activity. The logic assumes a 1:1 backing which does not account for loans already issued.

Recommendation

We recommend removing these checks or adjusting them to account for outstanding borrows when verifying available liquidity.

Client's Commentary:

We consider this issue as invalid. We are well aware that the available liquidity is not capped via totalSupply, but virtual balance. That being said, having both validations (once via explicit check, once via underflow on the vb), just adds another layer of security.

L-7	Overly Strict Overflow Check		
Severity	Low	**Status**	Fixed in c3e88e16

Description

This issue has been identified within several functions of the protocol:

- PercentageMath.sol#L50
- WadRayMath.sol#L89
- WadRayMath.sol#L100
- WadRayMath.sol#L134
- WadRayMath.sol#L145

The current overflow check:

```
gt(value, div(sub(not(0), HALF_PERCENTAGE_FACTOR), percentage))
```

is overly restrictive and may reject valid inputs unnecessarily. A more appropriate condition would be:

```
gt(value, div(not(0), percentage))
```

This simpler condition sufficiently ensures no overflow in `mul(value, percentage)` without the unnecessary margin added by subtracting `HALF_PERCENTAGE_FACTOR`.

Recommendation

We recommend replacing the existing overflow check with a more relaxed version:

```
gt(value, div(not(0), percentage))
```

This ensures safety while expanding the range of valid inputs.

Client's Commentary:

Addressed in 8405f6bcac6af438a808951f53ce5d6ec3a91b84

L-8	Suboptimal Rounding in Treasury Fee Calculation		
Severity	Low	**Status**	Fixed in c3e88e16

Description

The issue has been identified in the _accrueToTreasury function in ReserveLogic:

```
uint256 totalDebtAccrued = reserveCache.currScaledVariableDebt.rayMulCeil(
  reserveCache.nextVariableBorrowIndex - reserveCache.currVariableBorrowIndex
);

uint256 amountToMint = totalDebtAccrued.percentMul(reserveCache.reserveFactor);

if (amountToMint != 0) {
  reserve.accruedToTreasury += amountToMint
    .rayDivFloor(reserveCache.nextLiquidityIndex)
    .toUint128();
}
```

• ReserveLogic.sol#L191-L193

The current implementation uses ceiling rounding (rayMulCeil) when calculating the total debt accrued, which leads to overestimation of the debt. This results in the treasury receiving more fees than it should.
The issue is classified as **Low** severity because it does not pose any direct security risks to user funds.

Recommendation

Consider using rayMulFloor instead of rayMulCeil when calculating totalDebtAccrued to ensure consistent underestimation of fees.

Client's Commentary:

Addressed in 98148ea154379f592da6ae6672b450f5e57da758

L-9	Supply Cap Validation Sums Token Balances Instead of Shares		
Severity	Low	**Status**	Fixed in c3e88e16

Description

The issue has been identified in the validateSupply function in ValidationLogic:

```
require(
    supplyCap == 0
        || (
            (
                IAToken(reserveCache.aTokenAddress).scaledTotalSupply()
                    + uint256(reserve.accruedToTreasury)
            ).rayMulFloor(reserveCache.nextLiquidityIndex) + amount
        ) <= supplyCap * (10 ** reserveCache.reserveConfiguration.getDecimals()),
    Errors.SupplyCapExceeded()
);
```

• ValidationLogic.sol#L80-L87

Here, token balances are summed, whereas actually, after rounding down shares on mint, a user will receive fewer ATokens than the amount.
This issue is classified as **Low** severity because it results in only an insignificant incorrect enforcement of the supply cap constraint and does not pose a real risk to the protocol or user funds.

Recommendation

We recommend using more accurate calculations:

```
uint256 sharesAfterSupply = amount
    .rayDivFloor(reserveCache.nextLiquidityIndex) +
    IAToken(reserveCache.aTokenAddress).scaledTotalSupply() +
    uint256(reserve.accruedToTreasury);

if (
    sharesAfterSupply.rayMulFloor(reserveCache.nextLiquidityIndex) >
    supplyCap * (10 ** reserveCache.reserveConfiguration.getDecimals())
) {
    revert Errors.SupplyCapExceeded();
}
```

Client's Commentary:

Addressed in 070cd23d949d828ad78d098ef481c2049f5eabb6

L-10	Liquidation Rounding May Favor Liquidators Over Protocol		
Severity	Low	**Status**	Fixed in c3e88e16

Description

In the LiquidationLogic._calculateAvailableCollateralToLiquidate function, when the maximum collateral to liquidate exceeds the borrower's available collateral balance, the debtAmountNeeded is recalculated based on the actual collateral amount. However, the current calculation uses standard division, which may result in rounding down — potentially allowing liquidators to acquire collateral at a slightly discounted rate.

```
if (vars.maxCollateralToLiquidate > borrowerCollateralBalance) {
    vars.collateralAmount = borrowerCollateralBalance;
    vars.debtAmountNeeded = (
        (vars.collateralAssetPrice * vars.collateralAmount * debtAssetUnit)
            / (debtAssetPrice * collateralAssetUnit)
        ).percentDiv(liquidationBonus);
```

· LiquidationLogic.sol#L602-L605

This approach is inconsistent with the protocol's standard of rounding in its own favor.

Recommendation

We recommend rounding up the debtAmountNeeded calculation to ensure consistency with the protocol's rounding strategy.

Client's Commentary:

Addressed in 5177902bd8472174e841c38982ab4d0a183771a2

L-11	Potential Underflow in Liquidation Logic		
Severity	Low	**Status**	Fixed in c3e88e16

Description

In the LiquidationLogic.executeLiquidationCall function, liquidationProtocolFeeAmount is transferred to the treasury if it is non-zero. The system scales this amount (rounding down) and then compares it with the borrower's remaining scaled collateral balance to ensure sufficient funds for the fee transfer.

· LiquidationLogic.sol#L369-L387

A mismatch in rounding methods during fee calculation may lead to an edge case. While the comparison uses rayDivFloor to compute the required scaled amount, the subsequent transfer operation applies rayDivCeil. This discrepancy may result in an underflow error when attempting to transfer more fees than the borrower actually holds.

Recommendation

We recommend rounding down the scaledAmount — rather than rounding up — during AToken transfer operations to avoid underflow.

Client's Commentary:

Fix does not perfectly apply any more as the code was changed, but it was addressed here: 070cd23d949d828ad78d098ef481c2049f5eabb6

L-12	Misleading Comments in WadRayMath		
Severity	Low	**Status**	Fixed in c3e88e16

Description

Some comments in the WadRayMath library are misleading.

In the rayMul function, it is checked that a <= (type(uint256).max - HALF_RAY) / b, however, the following comment is present:

```
// Overflow check: Ensure a * b does not exceed uint256 max
```

- WadRayMath.sol#L73

The rayDiv function checks that a <= (type(uint256).max - b / 2) / RAY, however the comment states:

```
// Overflow check: Ensure a * RAY does not exceed uint256 max
```

- WadRayMath.sol#L118

The current documentation comment for the library may need to be updated, as additional functionality has been added that allows rounding behavior to be specified.

```
// Operations are rounded. If a value is >= 0.5,
// it will be rounded up; otherwise, it will be rounded down.
```

- WadRayMath.sol#L10

Recommendation

We recommend updating comments in the library to reflect the current logic and rounding behavior more accurately.

Client's Commentary:

Addressed in 8e9fae5a9f64b99f2ff64a581471d6fff873ec2e

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