MixBytes()

tramplin.io Security Audit Report

DECEMBER 10, 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

Tramplin is a solana validator that conducts random drawings for stake-based participants using Merkle tree proofs and commit-reveal scheme. The system features two types of draws (regular and big) with role-based access controls where operators manage participant snapshots, dealers execute drawings using VRF randomness, and winners claim prizes. Prize distribution is time-bounded with automatic withdrawal mechanisms for unclaimed rewards.

The codebase was audited in 10 days by 3 auditors using a combination of manual review and automated tooling.

During the audit, the following attack vectors were thoroughly checked and found to be properly implemented:

Role separation enforcement is properly implemented with all operations correctly validating admin/operator/dealer roles through strict key comparisons. Signer verification is consistently applied across all privileged operations with proper signature checks.

ORAO VRF integration is securely implemented with proper program ID validation preventing substitution attacks. Seed derivation properly combines merkle root with dealer secret. Final randomness combines VRF output with dealer secret via XOR operation, though only 32 out of 64 available bytes of ORAO randomness are utilized.

Stage transition validation strictly enforces Draw → Reveal progression with proper enum checks. Double claiming prevention is implemented through empty account validation before claim processing.

Merkle proof validation: stake range validation ensures correct stake ownership through proper range checks. Leaf construction uses secure hashing with prefix and withdrawer validation preventing manipulation.

Account ownership validation is comprehensive with all account access functions verifying program ownership. PDA derivation uses secure address generation with proper seeds ensuring deterministic creation. Bump seed consistency is maintained across all PDA operations.

The Drawing program demonstrates solid security foundations with proper implementation of core cryptographic and access control mechanisms.

However, during the audit, several system features were identified that do not impact security but nevertheless require attention:

Accounting Invariant Violation

The accounting logic has a flaw in the withdraw operation. The `draw_account.withdrawn` field includes not only the prize fund but also rent lamports that were initially on the account and any additional funds that could be manually sent to the pool after draw creation. This means that after calling `withdraw()`, the invariant `draw_account.withdrawn + draw_account.claimed >= draw_account.prize_fund` may hold true, breaking the expected accounting where withdrawn + claimed should equal the original prize fund.

Incomplete Entropy Usage

The randomness generation process only utilizes 32 out of 64 available bytes from ORAO VRF, effectively discarding 50% of available entropy. While this doesn't compromise security, it represents suboptimal use of VRF randomness and reduces the entropy in redistribution number generation. Consider combining both halves of ORAO randomness (e.g., XOR first 32 bytes with last 32 bytes, then XOR with dealer secret) to utilize full entropy while maintaining compatibility with KISS PRNG's 16-byte seed requirement.

Winner Count Transparency

The current architecture lacks transparency mechanisms for redistribution participants. Winner count is decided at draw time rather than predetermined. Users participate without knowing potential prize distribution, allowing dealers to strategically set winner numbers after seeing pool size and participants, which impacts redistribution fairness.

The ORAO program implementation was considered outside the scope of this security audit. The review focused specifically on examining the integration patterns and conducting a brief assessment of the ORAO program usage within the codebase, rather than performing a comprehensive analysis of the underlying ORAO protocol itself.

While the codebase generally follows Rust and Solana development practices, addressing the identified issues will strengthen the security posture of the redistribution drawing system.

1.3 Project Overview

Summary

Title	Description
Client Name	Tramplin
Project Name	Tramplin
Type	Rust
Platform	SVM
Timeline	03.09.2025 - 10.11.2025

Scope of Audit

File	Link
drawing/src/processor.rs	processor.rs
drawing/src/orao.rs	orao.rs
drawing/src/entrypoint.rs	entrypoint.rs
drawing/src/cpi.rs	cpi.rs
drawing/src/error.rs	error.rs
drawing/src/lib.rs	lib.rs
drawing/src/state.rs	state.rs
drawing/src/constant.rs	constant.rs
drawing/src/epoch_schedule.rs	epoch_schedule.rs
drawing/src/instruction.rs	instruction.rs
drawing/src/processor/operator.rs	operator.rs
drawing/src/processor/withdraw.rs	withdraw.rs
drawing/src/processor/draw.rs	draw.rs

File	Link
drawing/src/processor/create_pool.rs	create_pool.rs
drawing/src/processor/reveal.rs	reveal.rs
drawing/src/processor/admin.rs	admin.rs
drawing/src/processor/claim.rs	claim.rs

Versions Log

Date	Commit Hash	Note
03.09.2025	cb8c5b3d59fde49b4d0778b5d67ed0a818ce4ff8	Initial Commit
14.10.2025	8345118ecdd7079873beda1612000e458abecbac	Commit for Re-audit
29.10.2025	a74bb3a18dd938a4e0deda1ed251b45070cf2461	Commit for Re-audit
10.11.2025	dabb7b3f7ce8c7b15223cb8c3921e4883e097b70	Commit for Re-audit

Mainnet Deployments

Program	Address	Blockchain
Drawing	3NJyzGWjSHP4hZvsqakodi7jAtbufwd52vn1ek6EzQ35	Solana Mainnet

The deployed program hash matches the locally built executable from the audited commit. Hash: f0a7b3868f4e7c1cca7ed2741929eb8b13bfb285334d91ea7c4122ef48639218.

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 standard Rust tools 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 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	1
Medium	5
Low	9

Findings Statuses

ID	Finding	Severity	Status
C-1	Incorrect comparison in claim validation allows non-winners to claim	Critical	Fixed
H-1	Missing pool account address check in `process_claim()`	High	Fixed
M-1	Incorrect Account List in CPI Withdraw Function	Medium	Fixed
M-2	Duplicate Winner ID Claims Prevention	Medium	Fixed
M-3	Unbounded Claim Deadline Allows Indefinite Fund Locking	Medium	Fixed
M-4	No withdrawal mechanism for pool funds if a draw is not executed	Medium	Fixed
M-5	Missing Reveal Deadline Check for Regular Draw	Medium	Fixed
L-1	Missing Error Validation in Instruction Unpacking	Low	Fixed
L-2	Incorrect Account Mutability in CPI Functions	Low	Fixed
L-3	Inconsistent Argument Ordering in CPI Functions	Low	Acknowledged

L-4	Code Quality Issues	Low	Fixed
L-5	Insufficient Prize Fund Validation	Low	Fixed
L-6	Missing Amount Update for Big Draw Claims	Low	Fixed
L-7	Missing zero `merkle_root` check in `MakeSnapshot` validation	Low	Fixed
L-8	Front-running risk when requesting ORAO VRF in `process_draw()`	Low	Acknowledged
L-9	Account Order Mismatch in DrawingInstruction Definitions	Low	Fixed

2. Findings Report

2.1 Critical

C-1	Incorrect comparison in claim validation allows non-winners to claim		
Severity	Critical	**Status**	Fixed in a74bb3a1

Description
The `ClaimArgs::validate` function uses an inverted comparison (`self.stake_id >= self.winner_id`) when checking whether the `winner_id` falls within the eligible stake range. As a result, a non-winner can pass validation and claim the prize, while the actual winner may be incorrectly rejected.
The issue is classified as Critical severity because it enables unauthorized prize claims and can prevent rightful winners from claiming, directly impacting funds and protocol integrity.
Recommendation
We recommend correcting the condition to validate that `winner_id` is within the range `[stake_id, stake_id + stake)`, i.e., `self.winner_id >= self.stake_id && self.winner_id < self.stake_id + self.stake`.

Client's Commentary:
Fixed in 9bd8af5640ede7685b23ee47a47e450a6c761e54

2.2 High

H-1	Missing pool account address check in process_claim()		
Severity	High	**Status**	Fixed in 8345118e

Description
This issue has been identified within the process_claim function. There is no validation that the pool token account is associated with the specific draw being claimed. A claimant can supply any WSOL token account owned by the authority, enabling abuse. For example, if two draws (big and regular) run in the same epoch, a winner of one draw can specify the pool from the other draw. This can drain funds from the incorrect pool, causing shortfalls for legitimate winners of the other draw and requiring manual replenishment, with funds locked until claim_deadline.
The issue is classified as **High** severity because it enables denial of service through cross-pool fund misallocation, leading to operational overhead, delayed payouts, and temporarily locked funds.
Recommendation
We recommend adding explicit pool address validation in the process_claim function by deriving the expected pool address using derive_pool(args.kind, draw_account.epoch) and comparing it against the provided pool account. This ensures that only the correct pool for the specific draw can be used for claiming prizes.

Client's Commentary:
Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

2.3 Medium

M-1	Incorrect Account List in CPI Withdraw Function		
Severity	Medium	**Status**	Fixed in 8345118e

Description

The withdraw function in the cpi.rs module builds an incorrect account list that omits required accounts for the process_withdraw function in withdraw.rs. This problem causes construction of an instruction where some accounts are missing while other accounts remain at their expected positions, which makes the resulting call using this instruction data fail.

The issue is classified as **Medium** severity because it causes instruction construction failures that prevent successful CPI calls due to account misalignment in the instruction data.

Recommendation

We recommend updating the withdraw function in cpi.rs to include all required accounts: add AccountMeta::new_readonly(derive_authority(program_id).0, false) as the second account and AccountMeta::new(derive_metrics(program_id).0, false) as the seventh account to match the expected account layout in process_withdraw.

Client's Commentary:

Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

M-2	Duplicate Winner ID Claims Prevention		
Severity	Medium	**Status**	Fixed in 8345118e

Description

The current implementation in the process_claim function in claim.rs allows the possibility of duplicate winner_id values being generated during the drawing process in process_reveal. While there are no plans to have duplicate winner_ids in the same draw, the random generation process could theoretically produce the same winner_id multiple times, which would prevent subsequent winners from claiming their prizes due to the Claimed account derivation using only winner_id, epoch, and kind parameters.

The issue is classified as **Medium** severity because the possibility of duplicate winner_ids compromises the fairness of the drawing process and could prevent legitimate winners from claiming their rightful prizes.

Recommendation

We recommend implementing duplicate detection in the winner selection process during process_reveal and regenerating random values when duplicate winner_ids occur. This ensures all selected winners have unique identifiers and can successfully claim their prizes.

Client's Commentary:

Fixed in 6f6bbf826d3564941ef618801e3b96545d5d544e

M-3	Unbounded Claim Deadline Allows Indefinite Fund Locking		
Severity	Medium	**Status**	Fixed in 8345118e

Description

The `Config.claim_deadline` parameter in the configuration lacks an upper bound validation, allowing administrators to set extremely large values. When a claim deadline is set too high, unclaimed prize funds may remain locked indefinitely, as the program only validates that the deadline is greater than zero but does not enforce a maximum limit. This could result in significant amounts of tokens being permanently locked, preventing legitimate fund recovery mechanisms from functioning properly.

The issue is classified as **Medium** severity because it requires an administrative error to occur for the problem to manifest, but could still lead to permanent loss of user funds through improper configuration.

Recommendation

We recommend implementing a maximum claim deadline validation in the `ConfigArgs::validate()` function to prevent administrators from setting unreasonably large claim deadlines.

Client's Commentary:

Fixed in 8345118ecdd7079873beda1612000e458abecbac

M-4	No withdrawal mechanism for pool funds if a draw is not executed		
Severity	Medium	**Status**	Fixed in a74bb3a1

Description

The `CreatePool` flow initializes a PDA token account for the prize pool, but there is no mechanism to withdraw or recover funds if the draw is not executed (e.g., the dealer misses the deadline). In such cases, the funds remain locked in the pool account with no path to reclaim them.

The issue is classified as Medium severity because it can lead to permanent loss of access to funds in case of a privileged role mistake.

Recommendation

We recommend adding a controlled withdrawal function that allows authorized recovery of pool funds after some deadline if the draw wasn't executed or the secret wasn't revealed.

Client's Commentary:

Fixed in a74bb3a18dd938a4e0deda1ed251b45070cf2461

M-5	Missing Reveal Deadline Check for Regular Draw		
Severity	Medium	**Status**	Fixed in dabb7b3f

Description

The `process_reveal` function only enforces the deadline check for Big Draw but not for Regular Draw, allowing the dealer to postpone `reveal()` indefinitely after `draw()` with no upper time limit for Regular Draw. This creates a vulnerability where the dealer can manipulate redistribution outcomes by selectively choosing which draws to complete and which to abandon.

Recommendation

We recommend adding the deadline check for Regular Draw in the `process_reveal` function by removing the `if matches!(args.kind, DrawKind::Big)` condition and applying `draw_account.check_draw_deadline(clock.slot)?` to both draw types.

Client's Commentary:

Fixed in dabb7b3f7ce8c7b15223cb8c3921e4883e097b70

2.4 Low

L-1	Missing Error Validation in Instruction Unpacking		
Severity	Low	**Status**	Fixed in 8345118e

Description
The instruction unpacking logic in the `unpack` method of `DrawingInstruction` in `instruction.rs` for the Claim instruction performs byte slice operations without proper error validation. The `split_at` operations and `try_into` conversions could potentially panic and consume all compute units if the instruction data is malformed.
Recommendation
We recommend adding explicit error validation using `.ok_or(ProgramError::InvalidInstructionData)?` after each slice operation and ensure all byte array conversions are properly validated.

Client's Commentary:
Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

L-2	Incorrect Account Mutability in CPI Functions		
Severity	Low	**Status**	Fixed in 8345118e

Description
Several CPI functions in the `cpi.rs` module declare accounts with incorrect mutability flags. The dealer account in the `withdraw` function is marked as mutable when it should be read-only, and the participants account in the `draw` function should be created via `new_readonly`. These incorrect flags can lead to unnecessary rent charges and potential security issues.
Recommendation
We recommend updating the CPI functions to use correct mutability flags: change `AccountMeta::new(*dealer, true)` to `AccountMeta::new_readonly(*dealer, true)` in the withdraw function, and ensure participants account uses `AccountMeta::new_readonly(derive_participants(program_id).0, false)` in the draw function.

Client's Commentary:
Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

L-3	Inconsistent Argument Ordering in CPI Functions		
Severity	Low	**Status**	Acknowledged

Description

The argument ordering in CPI functions in the `cpi.rs` module is inconsistent across different operations. The `reveal` and `withdraw` functions have confusing parameter orders where `kind` should be the first parameter for consistency with other functions. While this doesn't lead to logic errors since named parameters are used, the inconsistent ordering may confuse readers of the program and developers may make mistakes in future development.

Recommendation

We recommend standardizing the argument ordering across all CPI functions by placing `kind` as the first parameter consistently.

Client's Commentary:

Won't fix

L-4	Code Quality Issues		
Severity	Low	**Status**	Fixed in 8345118e

Description

Multiple code quality issues exist throughout the codebase including unnecessary imports (RngCore in reveal.rs), variable naming inconsistencies (final_randmoness in process_reveal, totat_withdrawn in Metrics struct, signer instead of withdrawer in CPI instruction definition), and unclear field names (drawing_condition in state structures). These issues reduce code maintainability and can lead to confusion during development and auditing.

Recommendation

We recommend performing a comprehensive code cleanup by removing unused import, fixing typos in variable names, and improving naming conventions for better clarity.

Client's Commentary:

Contract will not compile without RngCore import

other issues fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

L-5	Insufficient Prize Fund Validation		
Severity	Low	**Status**	Fixed in 8345118e

Description
The draw initialization in the process_draw function in draw.rs doesn't validate that the prize fund is sufficient for the number of winners, which could lead to rounding down during prize distribution (prize_fund / winners_num as u64) in the process_reveal function or leave dust amounts. This could result in unfair prize distribution or accumulated dust that cannot be properly distributed to winners.
Recommendation
We recommend implementing minimum prize fund validation during draw initialization to ensure the prize fund is sufficient for meaningful prize distribution. Add checks that verify the prize fund can be evenly distributed among winners with acceptable remainder amounts.

Client's Commentary:
Fixed in 42cfb6a04e2c9f52a1a24cb122eb7b986b72d2f6

L-6	Missing Amount Update for Big Draw Claims		
Severity	Low	**Status**	Fixed in 8345118e

Description

In the claim processing logic for Big draws in the `process_claim` function in `claim.rs`, the `claimed.amount` field is not properly updated after a successful claim, while it is correctly set for Regular draws. This inconsistency leads to inaccurate tracking of claimed amounts for Big draw winners and could affect metrics.

Recommendation

We recommend adding the missing `claimed.amount = winner.prize;` assignment in the Big draw claim processing branch to ensure consistent tracking of claimed amounts across both draw types. This maintains proper accounting and metrics consistency.

Client's Commentary:

Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

L-7	Missing zero `merkle_root` check in `MakeSnapshot` validation		
Severity	Low	**Status**	Fixed in 8345118e

Description

This issue has been identified within `MakeSnapshotArgs.validate()`.
The function does not verify that `merkle_root` is non-zero (`[0u8; 32]`). Since the operator cannot update `merkle_root` for a given epoch, providing a zero or otherwise incorrect value would prevent the draw from being executed for that epoch, resulting in a denial of service for that round.
The issue is classified as **Low** severity because it causes availability issues without directly enabling fund loss, but it can halt draws for an entire epoch.

Recommendation

We recommend adding an explicit check to ensure `merkle_root != [0u8; 32]` during `MakeSnapshot` validation, and optionally providing a mechanism to update or override the snapshot for the affected epoch.

Client's Commentary:
Fixed in b61eae3ee4b54b9ba29a62e138a1e3e960c66616

L-8	Front-running risk when requesting ORAO VRF in `process_draw()`		
Severity	Low	**Status**	Acknowledged

Description

An attacker can front-run the `orao.request_v2()` call by observing the transaction parameters (including the `seed`) via gossip and submitting their own ORAO request with the same `seed` first. If the attacker's request is processed before the program's, the CPI call inside `process_draw()` will fail because the corresponding randomness account already exists. This creates a griefing attack vector that forces the dealer to regenerate the `secret` and `seed` and resubmit the transaction, which the attacker can frontrun again.

The issue is classified as **Low** severity because it does not compromise funds or the correctness of randomness, but it can cause denial-of-service-style friction and operational delays.

Recommendation

We recommend invoking `orao.request_v2()` only if the corresponding ORAO randomness account does not already exist. If it exists and is not yet fulfilled, use that existing request account to retrieve randomness instead of issuing a new request. If it is already fulfilled, the secrets collide; the transaction should revert.

Client's Commentary:

Won't fix. Solana doesn't have public mempool unlike EVM. Should be taken into account during backend development.

L-9	Account Order Mismatch in DrawingInstruction Definitions		
Severity	Low	**Status**	Fixed in dabb7b3f

Description

There are two documentation inconsistencies in the DrawingInstruction enum where the account annotations do not match the actual account order used in the processor implementations. In the Draw instruction, the documentation shows token_program at index 6 and system_program at index 7, but the processor implementation uses the reverse order with system_program at index 6 and token_program at index 7. In the Withdraw instruction, the documentation incorrectly lists metrics twice at indices 6 and 7, while the processor implementation expects metrics at index 6 and token_program at index 7. While these mismatches don't affect program functionality, they create confusion for developers.

Recommendation

We recommend correcting the account annotations in the DrawingInstruction enum to match the actual implementation by swapping system_program and token_program in the Draw instruction (placing system_program at index 6 and token_program at index 7), and replacing the duplicate metrics with token_program at index 7 in the Withdraw instruction to ensure the documentation accurately reflects the actual account order used in the processor implementations.

Client's Commentary:

Fixed in dabb7b3f7ce8c7b15223cb8c3921e4883e097b70

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