MixBytes()

Jupiter Lend Vaults on Solana Security Audit Report

SEPTEMBER 16, 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

The Jupiter Lend Vaults program enables collateralized borrowing on Solana through a sophisticated debt position management system. Users can create positions by supplying collateral tokens and borrowing against them, managed through a tick-based system for precise position tracking. The program includes liquidation mechanisms that can be triggered when positions exceed risk thresholds, configurable parameters like collateral factors and liquidation penalties, and integrates with the liquidity and oracle programs for seamless borrowing and real-time price feeds.

The audit was conducted over 26 days by 3 auditors, using a combination of manual review and automated tooling.

During the audit, the following attack vectors were checked:

1. Solana Accounts Model Security: The codebase demonstrates robust account validation that effectively prevents unauthorized actions through crafted accounts. The implementation correctly leverages Anchor's constraint system combined with comprehensive runtime validation:

- Position Authority Validation: The verify_operate function at lines 147-150 in validate.rs properly validates that position_token_account.mint == position.position_mint and position_token_account.amount == 1, preventing cross-position attacks where an attacker could operate on others' positions using their own NFT tokens.
- Account Derivation Consistency: PDAs are consistently derived using deterministic seeds throughout the codebase, ensuring account relationships cannot be spoofed. The SDK correctly implements these derivations client-side, but the on-chain validation provides the critical security boundary.
- Multi-layer Validation: The combination of Anchor constraints, custom validation functions like verify_position_authority_interface, and runtime checks creates defense-in-depth against account substitution attacks.

2. Liquidation Process Correctness: The liquidation mechanism exhibits sophisticated correctness with proper handling of complex liquidation scenarios:

- Branch and Tick Management: The liquidation logic correctly manages the hierarchical relationship between branches, ticks, and partials. The debt_factor calculations in branch merging operations maintain mathematical consistency across liquidation events.
- Debt Tracking Integrity: The system properly tracks debt through debt_liquidity variables and connection factors, ensuring no debt is lost during liquidation cascades. The absorb functionality correctly handles bad debt above maximum liquidation limits.

- Partial Liquidation Handling: The implementation correctly calculates partial liquidations using the get_current_partials_ratio function, maintaining proper ratio calculations even when positions are partially liquidated across multiple transactions.
- State Consistency: Liquidation state updates are atomic and maintain consistency between vault state, branch states, and individual position states throughout the liquidation process.

3. DoS Resistance and Compute Optimization: The codebase shows good awareness of Solana's compute limitations with several optimization strategies:

- Batch Processing: The use of remaining accounts for branches, ticks, and oracle sources allows processing multiple related accounts without hitting instruction size limits.
- Efficient Data Structures: The tick and branch data structures are optimized for minimal compute usage during traversal and updates.
- Bounded Operations: Critical loops in liquidation have proper termination conditions, preventing runaway compute consumption from malicious position configurations.
- Account Limiting: The system limits the number of accounts that can be processed in a single transaction through remaining_accounts_indices validation, preventing DoS through excessive account requirements.

4. Oracle Price Range Limitations: The implementation reveals a significant constraint in price range handling compared to the EVM version:

- Restricted Price Range: The Solana implementation constrains exchange rates to the range [1e6, 1e26] (20 orders of magnitude), which is substantially narrower than the EVM version's range of [1e9, 1e54] (45 orders of magnitude).

- Token Compatibility Impact: This reduced range may create compatibility issues with certain token pairs, particularly those with extreme decimal differences or highly volatile price relationships. For example, tokens with very high decimal precision paired with low-value tokens could exceed these bounds.

- Risk Assessment: While the current range covers most mainstream token pairs, the limitation could become problematic for:
  - Micro-cap tokens with extreme price variations
  - Cross-chain wrapped tokens with unusual value relationships
  - Future token innovations that operate outside conventional price ranges

The audit scope was specifically limited to the vaults module of the Jupiter Lend Protocol, focusing exclusively on the collateralized debt position management system.

During the audit, we conducted a comprehensive comparison between the EVM implementation and the current Solana scope, thoroughly going through main components of the original EVM codebase to ensure functional parity and identify potential migration-related vulnerabilities.

The codebase demonstrates high-quality engineering with strong security practices, comprehensive validation, and thoughtful optimization for Solana's constraints. The account model security is particularly robust, and the liquidation logic is mathematically sound.

1.3 Project Overview

Summary

Title	Description
Client Name	Jupiter Lend
Project Name	Vault
Type	Rust
Platform	SVM
Timeline	28.07.2025 - 10.09.2025

Scope of Audit

File	Link
programs/vaults/src/events.rs	events.rs
programs/vaults/src/constants.rs	constants.rs
programs/vaults/src/module/user.rs	user.rs
programs/vaults/src/module/mod.rs	mod.rs
programs/vaults/src/module/admin.rs	admin.rs
programs/vaults/src/lib.rs	lib.rs
programs/vaults/src/invokes/liquidity_layer.rs	liquidity_layer.rs
programs/vaults/src/invokes/mod.rs	mod.rs
programs/vaults/src/invokes/oracle.rs	oracle.rs
programs/vaults/src/invokes/mint.rs	mint.rs
programs/vaults/src/utils/mod.rs	mod.rs
programs/vaults/src/utils/validate.rs	validate.rs

File	Link
programs/vaults/src/utils/operate.rs	operate.rs
programs/vaults/src/utils/liquidate.rs	liquidate.rs
programs/vaults/src/state/branch.rs	branch.rs
programs/vaults/src/state/vault_state.rs	vault_state.rs
programs/vaults/src/state/tick_has_debt.rs	tick_has_debt.rs
programs/vaults/src/state/mod.rs	mod.rs
programs/vaults/src/state/state.rs	state.rs
programs/vaults/src/state/seeds.rs	seeds.rs
programs/vaults/src/state/position.rs	position.rs
programs/vaults/src/state/tick.rs	tick.rs
programs/vaults/src/state/structs.rs	structs.rs
programs/vaults/src/state/vault_config.rs	vault_config.rs
programs/vaults/src/state/context.rs	context.rs
programs/vaults/src/errors.rs	errors.rs
programs/vaults/src/module/view.rs	view.rs

Versions Log

Date	Commit Hash	Note
28.07.2025	ea55b3f63a345889228d30feb19a4e7f681f9435	Initial Commit
09.09.2025	9170ece98c09d52b437e6bc44ff48a8edbbb9d94	Commit for Re-audit

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 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	0
High	0
Medium	0
Low	3

Findings Statuses

ID	Finding	Severity	Status
L-1	Typos and Incorrect Comments	Low	Fixed
L-2	Missing `init_if_needed` for signer supply ATA causes `operate()` to revert	Low	Fixed
L-3	Absorption reverts when branch with zero id is missing	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	Typos and Incorrect Comments		
Severity	Low	**Status**	Fixed in 9170ece9

Description

Several code quality issues were identified in the codebase:

1. Misleading comment in programs/vaults/src/state/context.rs:
   - context.rs#L319

   ```
   // No init_if_needed, as the signer associated token account
   // must already exist in order to payback
   // @dev make it optional in future
   #[account(
       init_if_needed,  // <- Actually uses init_if_needed
       payer = signer,
       associated_token::mint = borrow_token,
       associated_token::authority = signer,
       associated_token::token_program = borrow_token_program
   )]
   pub signer_borrow_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
   ```

   The comment incorrectly states "No init_if_needed" but the macro actually uses init_if_needed. This is necessary because when a user pays back debt, they might not have an associated token account for the borrow token initialized yet.

2. Typo in parameter name in programs/vaults/src/invokes/oracle.rs:
   - oracle.rs#L14

   - oracle.rs#L26

   ```
   fn get_exchange_rate(&self, nonce: u16, is_iquidate: bool) -> Result<u128> {
   ```

   The parameter is_iquidate is missing the 'l' and should be is_liquidate.

3. Typo in variable name in multiple files:
   - liquidate.rs#L109
   - liquidate.rs#L121
   - operate.rs#L41
   - operate.rs#L53

   In programs/vaults/src/utils/operate.rs and programs/vaults/src/utils/liquidate.rs:

   ```
   let remainining_accounts = ctx  // <- Should be: remaining_accounts
       .remaining_accounts
       .iter()
       ...
   ```

   The variable name remainining_accounts has an extra 'in' and should be remaining_accounts.

4. Typo in comment in programs/vaults/src/constants.rs:
   - constants.rs#L35
   - constants.rs#L38

   ```
   // Minium acceptable collateral amount  // <- Should be: Minimum
   // Minium acceptable debt amount        // <- Should be: Minimum
   ```

   The comment incorrectly spells "Minimum" as "Minium" in two locations.

5. Typo in error code names in programs/vaults/src/errors.rs:
   - errors.rs#L155
   - errors.rs#L158

   ```
   VAULT_CPY_TO_LIQUIDITY_FAILED,  // <- Should be: CPI (Cross Program Invocation)
   VAULT_CPY_TO_ORACLE_FAILED,     // <- Should be: CPI (Cross Program Invocation)
   ```

   The error codes incorrectly use "CPY" instead of "CPI" (Cross Program Invocation), which is the standard Solana terminology.

   These are code quality issues that don't affect functionality but reduce code clarity and maintainability. The misleading comment could confuse developers about the expected account initialization behavior.

Recommendation

1. Update the comment in context.rs to accurately reflect that init_if_needed is used and explain why it's necessary
2. Fix the typo is_iquidate → is_liquidate in oracle.rs
3. Fix the typo remainining_accounts → remaining_accounts in operate.rs and liquidate.rs
4. Fix the typo Minium → Minimum in constants.rs comments
5. Fix the typo CPY → CPI in error code names in errors.rs

Client's Commentary:

Ack and fixed in this PR - PR-79

some were fixed previously in earlier commits.

L-2	Missing init_if_needed for signer supply ATA causes operate() to revert		
Severity	Low	Status	Fixed in 9170ece9

Description

The Operate struct context.rs#L309-L317 signer_supply_token_account to be already initialized. As a result, if the position NFT is transferred or delegated to an account that does not already have an associated token account for the supply token, the operate call will revert during account validation. This creates a denial-of-service condition for legitimate ownership/delegation flows. To overcome this problem, manual creation of an ATA is required. The issue is classified as **Low** severity because it can block position operations under common scenarios (transfer/delegation) without directly risking loss of funds.

Recommendation

We recommend adding init_if_needed to signer_supply_token_account or making it optional.

Client's Commentary:

Acknowledged. This is a known issue and can be handled by attaching a create ATA instruction to the transaction before calling operate, which will initialize the user's token account. Since the fix is straightforward and doesn't introduce risk, we believe this should be considered Low severity.

fix commit - PR-79

L-3	Absorption reverts when branch with zero id is missing		
Severity	Low	**Status**	Fixed in 9170ece9

Description

This issue has been identified within the absorb function.

The function can revert if the branch account with id == 0 is not supplied in the instruction context. When the current branch is the base branch, connected_minima_tick admin.rs#L209 i32::MIN, which causes the logic to set branch_data.id to 0 and later attempt to load the branch with id 0 from the provided accounts (user.rs#L931, user.rs#L967). If that account is not included, the call fails and the absorption flow reverts. Normally branch numbers start with 1, so to bypass the problem it will be required to create a separate dummy account.

The issue is classified as **Low** severity because it can cause a denial-of-service for the absorption process (potentially blocking liquidation progress) without directly compromising funds.

Recommendation

We recommend normalizing base-branch initialization, setting connected_minima_tick to i32::MIN in the reset_branch_data function.

Client's Commentary:

Client: This is actually working as intended, not a bug. The way we designed the system, Branch ID 0 acts like a conceptual "master branch", think of it as the foundation that other branches build on top of. When the absorption process runs through all the branches during liquidation, it eventually needs to fall back to this master level, which is exactly why the code sets branch_data.id = 0 at that point.

The audit was done on an older version of our code where we were checking connected_minima_tick != 0, but we've updated that to use connected_minima_tick != COLD_TICK instead before going live. This was an important fix because COLD_TICK (which equals i32::MIN) specifically means "no tick constraint exists," while 0 is actually a valid tick value.

fix commit - 8ecdb86a

MixBytes: We acknowledge the clarification that Branch ID 0 serves as the conceptual "master branch" and that the fix updating the condition from connected_minima_tick != 0 to connected_minima_tick != COLD_TICK effectively addresses the core issue. Based on this fix and design explanation, we have updated our recommendation to setting connected_minima_tick to i32::MIN (COLD_TICK) instead of 0 in the reset_branch_data function, which aligns with the client's approach of using COLD_TICK as the sentinel value for "no constraint" and resolves our original concern about potentially entering an incorrect execution branch in the absorb function due to ambiguous connected_minima_tick value.

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

𝕏 https://x.com/mixbytes