Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Saros Liquidity Book**

Version: 1.2 (final report)
Date: 23rd June 2025
Commit/Hash: 05d3349a0c63d261b237b96649540ac5230e89cf
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

1.1 Project Overview

`Saros Liquidity Book` is an advanced Automated Market Maker (AMM) protocol designed to enhance decentralized trading on the Solana blockchain. By utilizing a unique bin-based architecture, it allows liquidity providers to concentrate their assets within specific price ranges, optimizing capital efficiency and reducing slippage for traders. The protocol also incorporates dynamic surge pricing, enabling liquidity providers to earn additional fees during periods of high market volatility. With its innovative approach, `Saros Liquidity Book` offers a flexible and efficient trading experience, catering to both seasoned traders and newcomers in the DeFi space.

1.2 Objective

This audit was conducted to assess the robustness, reliability, and security of the codebase. The goal was to identify vulnerabilities, ensure compliance with best security practices, and provide mitigation measures. The initial audit report was provided by `CYBRING` on May 7, 2025.

After the initial audit report, a reassessment was conducted on May 14, 2025, to verify the status of the reported issues. During this reassessment, two new critical issues were discovered. Subsequently, on June 23, 2025, these two newly identified issues were fixed, and `CYBRING` performed another reassessment.

1.3 Disclaimer

This security audit is not produced to replace any other type of assessment and does not aim to guarantee the discovery of all security issues within the scope of the assessment. While this audit was carried out with good faith and technical proficiency, it's crucial to understand that no single audit can guarantee the absolute security of a smart contract. To minimize risks, `CYBRING` recommends a multi-faceted approach involving multiple independent assessments. Please note that this report is not intended to provide financial advice.

2 Scope of Audit

The audit was conducted on commit 05d3349a0c63d261b237b96649540ac5230e89cf from git repository `https://github.com/saros-xyz/saros-sol-liquidity-book`. Further details regarding The `Saros Liquidity Book` audit scope are provided below:

- **Smart contracts audited:** See Appendix.
- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: 3ff57480a7baac6b53247645e6dadca2e66439b1
  - 1st Reassessment's commit hash: 7ec89da21077e5bc1abb299e14c453f4e24fc339

  - 2nd Reassessment's commit hash: 05d3349a0c63d261b237b96649540ac5230e89cf

3 Audit Summary

Initial assessment report is summarized in Table 1. Details of findings can be found in Section 5.

Severity	Count
Critical	2
Medium	3
Low	5

Table 1: Initial assessment summary

After code updates and design clarifications from the developers, we re-audited the latest patched build:

Issue	Severity	Status	Remediation commit
5.1	Critical	Fixed	b963445
5.2	Critical	Fixed	aef06fa
5.3	Medium	Fixed	ccf8ab9
5.4.1	Medium	Fixed	ecf7d50
5.4.2	Medium	Fixed	ecf7d50
5.4.3	Low	Fixed	ecf7d50
5.5	Low	Acknowledged	
5.6	Low	Acknowledged	
5.7	Low	Fixed	c0589af
5.8	Low	Fixed	7ec89da

Table 2: Final assessment summary

All critical and medium issues are fixed; two low-severity issues remain acknowledged.

4 Methodology

`CYBRING` conducts the following procedures to enhance security level of our client's smart contracts:

- **Pre-auditing:** Understanding the business logic of the smart contracts, investigating the deployment states of samples, and preparing for the audit.
- **Auditing:** Examining the smart contract by evaluating on multiple perspectives:
  - **Manual code review:** `CYBRING` auditors evaluate the static code analysis report to filter out false positive reports. Besides, inspect the smart contract logic, access controls, and data flows ensure the contract behaves as intended and is free from risky logic.

  - **Static code analysis:** By using advanced static analysis techniques combined with our customized detectors, we set out to identify potential vulnerabilities, optimize gas usage, and ensure adherence to best practices in smart contract development.
  - **Fuzz testing:** `CYBRING` leverages fuzzing tools to stress-test the contract, ensuring it performs securely under unpredictable conditions.

- **First deliverable and consulting:** Presenting an initial report on the findings with recommendations for remediation and offering consultation services.
- **Reassessment:** Verifying the status of the issues and identifying any additional complications in the implemented fixes.
- **Final deliverable:** Delivering a comprehensive report detailing the status of each issue.

4.1 Risk Rating

The OWASP Risk Rating Methodology was applied to assess the severity of each issue based on the following criteria, arranged from the perspective of smart contract security.

- **Likelihood:** a measure of how likely this vulnerability is to be discovered and exploited by an attacker.
- **Impact:** a measure of the potential consequences or severity of a vulnerability if it is exploited by an attacker, including the extent of damage, data loss, or disruption of operations.

Table 3 shows the details of the security severity assessment for each issue.

Impact				
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Informational	Low	Medium
		Low	*Medium*	*High*
		Likelihood		

Table 3: Overall Risk Severity

4.2 Audit Categories

- **Common vulnerabilities:** Smart contracts are analyzed following OWASP smart contract top 10 and Smart Contract Weakness Classification (SWC).
- **Advanced vulnerabilities:** `CYBRING` simulates a certain types of attack scenarios to exploit the smart contracts. These scenarios were prioritized from high to low severity.
- **Security best practices:** The source code of the smart contract is analyzed from the development perspective, providing suggestions for improving the overall code quality.

4.3 Audit Items

Table 4 shows the details of the issues our auditors will conduct the audit upon.

Category	**Item**
Missing PDA validation	Missing verify user-supplied PDA
	Incorrect seeds used in derivation
	Missing bump seed validation
	Reusing same seed for multiple roles
Data Validation and Integrity	Integer overflow/underflow
	Misuse of *unwrap()* or *expect()*
	Unvalidated input (especially in CPI)
Exceeding compute budget and Denial of service (DOS)	Unbounded loops
	Exceeding compute unit limit with loops
	Exceeding compute unit limit with nested CPI
Access control & authority	Authority change without proper checks
	Upgrade authority mishandled
	Missing lock upgrade authority post-deployment
Private information and randomness	Store sensitive information on smart contracts
	Insecure randomness source
Timestamp dependence	Timestamp dependence on critical function
Best practices	Use of *checked_add*, *checked_sub*, etc.
	Separate validation logic from execution logic
	Follow standard style guide (naming conversion, program modularity, order of layout, etc.)

Table 4: Details of examined items

5 Detailed Findings

5.1 Absence of Expected Liquidity Caused by Inconsistent Protocol-Fee Calculation

- **Description:** Protocol fees for swaps are inconsistently calculated. The fee is computed in token X but added in token Y (and vice versa for the opposite swap direction). This leads to missing liquidity if token X's value is less than token Y's, or underpayment of fees to the pair owner if token X's value is greater.

  Attack scenario: A compromised pair owner could exploit this vulnerability by creating malicious swap instructions (e.g., swapping a low-value memecoin for a high-value stablecoin). This manipulation would allow the compromised pair owner to gain an unfair advantage in protocol fees, effectively siphoning liquidity from the protocol.

- **Risk:** Critical

- **CWE:** CWE-430: Deployment of Wrong Handler

- **Affected code:**

```
    if swap_for_y {
        pair.protocol_fees_y = pair
            .protocol_fees_y
            .checked_add(total_protocol_fee)
            .ok_or(LBError::AmountOverflow)?;
    } else {
        pair.protocol_fees_x = pair
            .protocol_fees_x
            .checked_add(total_protocol_fee)
            .ok_or(LBError::AmountOverflow)?;
    }

```

programs/liquidity-book/src/manager/swap_manager.rs:142-152

- **Mitigation:** Ensure that the protocol fee is consistently calculated and applied in the correct token throughout the swap process.

- **Status [23/06/2025]:** `Saros Liquidity Book` team has acknowledged and fixed this issue.

5.2 Repeated Protocol Fee Withdrawal Vulnerability

- **Description:** A critical vulnerability exists at *withdraw_protocol_fees* instruction, where the *protocol_fee* account is not properly reset to zero after a withdrawal due to a missing mutable property. This oversight allows a compromised pair owner to repeatedly withdraw the protocol fee, effectively draining the protocol's vault until one of its two program vaults is exhausted.

- **Risk:** Critical
- **CWE:** CWE-863: Incorrect Authorization
- **Affected code:**

```
    #[account(
        has_one = liquidity_book_config @ LBError::InvalidConfig,
        has_one = token_mint_x @ LBError::PairTokenMismatch,
        has_one = token_mint_y @ LBError::PairTokenMismatch,
    )]
    pub pair: Account<'info, Pair>,

```

programs/liquidity-book/src/instructions/admin/withdraw_protocol_fees.rs:20-25

- **Mitigation:** Add the *mut* (mutable) property to the *pair* account to ensure its state can be modified.
- **Status [23/06/2025]:** `Saros Liquidity Book` team has acknowledged and fixed this issue.

5.3 Potential for Out-of-Bounds Vulnerability

- **Description:** The code at *programs/liquidity-book/src/state/position.rs:44* is susceptible to an out-of-bounds access by missing the following constraint when creating a position:

  $upper\_bin\_id - lower\_bin\_id + 1 <= MAX\_BIN\_PER\_POSITION.$
- **Risk:** Medium
- **CWE:** CWE-125: Out-of-bounds Read
- **Affected code:**

```
    pub fn get_share_mut(&mut self, bin_id: u32) -> Result<&mut u128> {
        if bin_id < self.lower_bin_id || bin_id > self.upper_bin_id {
            return Err(LBError::BinNotFound.into());
        }

        Ok(&mut self.liquidity_shares[(bin_id - self.lower_bin_id) as usize])
    }

```

programs/liquidity-book/src/state/position.rs:39-45

- **Mitigation:** The above constraint should be implemented in creating position handler.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

5.4 Using unwrap() Insecurely

5.4.1 Potential Panic when Token Reserve Exceeds Max Liquidity per Bin

- **Description:** When the reserve of one of the two tokens in a bin exceeds the max liquidity constant, the require condition at *programs/liquidity-book/src/state/bin.rs:81* or *programs/liquidity-book/src/state/bin.rs:292* or *programs/liquidity-book/src/state/bin.rs:375* will not be met. The *get_shares_and_effective_amounts_in* function is expected to return a *LiquidityOverflow* error in this scenario.

  However, some references are currently using *unwrap()* to handle the return value. Failing to check and handle this *LiquidityOverflow* error will lead to a panic (runtime error) when the token reserve overflow condition occurs.

- **Risk:** Medium

- **CWE:** CWE-248: Uncaught Exception

- **Affected code:**

```
136     let (mut shares, mut effective_amount_x, mut effective_amount_y) = bin
137         .get_shares_and_effective_amounts_in(amount_x, amount_y, price)
138         .unwrap();
139
```

programs/liquidity-book/src/manager/liquidity_manager.rs:136-138

```
36     let (amount_in_with_fees, amount_out_of_bin, fee_amount, protocol_fee) = bin
37         .swap_exact_in(
38             pair.bin_step,
39             pair.active_id,
40             amount_in_left,
41             fee,
42             pair.get_protocol_share(),
43             swap_for_y,
44     )
45     .unwrap();
46
```

programs/liquidity-book/src/manager/swap_manager.rs:36-45

```
103     let (amount_in_with_fees, amount_out_of_bin, fee_amount, protocol_fee) = bin
104         .swap_exact_out(
105             pair.bin_step,
106             pair.active_id,
107             amount_out_left,
108             fee,
                pair.get_protocol_share(),
                swap_for_y,
            )
                .unwrap();

```

programs/liquidity-book/src/manager/swap_manager.rs:103-112

- **Mitigation:** Handle the returned error instead of using *unwrap()*.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

5.4.2 Potential Panic when Retrieving Shares of a Bin Outside Position

- **Description:** If a Liquidity Provider attempts to add liquidity and their specified liquidity distribution contains a bin ID that falls outside the *lower_bin_id* and *upper_bin_id* of the target position, the transaction will trigger a panic.
- **Risk:** Medium
- **CWE:** CWE-248: Uncaught Exception
- **Affected code:**

```
    let user_share = position.get_share_mut(bin_id).unwrap();
```

programs/liquidity-book/src/manager/liquidity_manager.rs:83

- **Mitigation:** Handle the returned error instead of using *unwrap()*.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

5.4.3 Potential Panic when the mul_div function returns None

- **Description:** The *mul_div()* function can return None if the result of an internal *div_ceil()* operation exceeds the maximum value of **u128** (often due to a very small denominator). At *programs/liquidity-book/src/state/bin.rs:46*, the code unwraps this potential **None** value, which will cause a panic if *mul_div()* indeed returned None.
- **Risk:** Low
- **CWE:** CWE-248: Uncaught Exception
- **Mitigation:** Handle the returned error instead of using *unwrap()*.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

5.5 Potential Revert When Crossing Extensive Empty-Bin Gaps

- **Description:** During *swap()* the program iterates through consecutive bins until it finds liquidity. If a trade must traverse a set of many empty bins, the loop can exceed the transaction's compute-unit budget and the instruction reverts. The accompanying fee escalation for each empty bin (volatility accumulator update) is intentional and affects only the caller's transaction, similar to high slippage.
- **Risk:** Low
- **CWE:** CWE-754: Improper Check for Unusual or Exceptional Conditions
- **Mitigation:** Pre-check path liquidity and split orders to avoid large gaps. Add a hard cap on the number of bins traversed per swap (e.g., $MAX\_BINS\_CROSSED$) to fail fast with a clearer error. Alert pool operators when prolonged empty stretches appear, signalling a stale pool.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has acknowledged this issue.

5.6 Unclaimed Protocol Fees May Cause a DoS Vulnerability

- **Description:** During every swap, the Liquidity Book siphons off a portion of the trading fee as protocol revenue. If these earnings accumulate without being claimed, the counter may eventually overflow, causing all subsequent swaps to revert.
- **Risk:** Low
- **CWE:** CWE-190: Integer Overflow or Wraparound
- **Affected code:**

```
    if protocol_fee > 0 {
        effective_amount_x -= protocol_fee;

        pair.protocol_fees_x = pair
            .protocol_fees_x
            .checked_add(protocol_fee)
            .ok_or(LBError::AmountOverflow)?;
    }

```

programs/liquidity-book/src/manager/liquidity_manager.rs:213-220

- **Mitigation:** The preset authority role should be aware that periodically claiming protocol fee distribution system is crucial to prevent potential integer overflows and subsequent transaction rollbacks.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has acknowledged this issue.

5.7 Absence of an NFT-Burn Mechanism on Position Closure

- **Description:** Currently, the Liquidity Book doesn't burn authority NFTs when positions close, which can confuse Liquidity Providers. They might mistakenly believe their exited positions are still active. Burning these NFTs would clearly indicate closure, improving clarity about their investments and overall system understanding.
- **Risk:** Low
- **CWE:** CWE-672: Operation on a Resource after Expiration or Release
- **Mitigation:** Burn authority mint token mechanism should be implemented.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

5.8 Unchecked Cast Presents an Overflow Risk

- **Description:** In *programs/liquidity-book/src/state/bin.rs:122,134*, the program lacks a crucial check on the return value of the *mul_div()* function before casting. If the result of *mul_div()* exceeds the maximum value representable by a **u64**, the subsequent type casting will lead to significant calculation errors and data corruption. This unchecked casting can introduce inaccuracies in core financial computations within the Liquidity Book.
- **Risk:** Low
- **CWE:** CWE-190: Integer Overflow or Wraparound
- **Mitigation:** Implement a check on the return value of *mul_div()* to ensure it does not exceed the maximum value of **u64** before performing the cast.
- **Status [14/05/2025]:** `Saros Liquidity Book` team has addressed this issue.

Appendix

```
programs/
└─ liquidity-book
   └─ src
      ├─ instructions
      │  ├─ admin
      │  │  ├─ accept_config_ownership.rs
      │  │  ├─ initialize_bin_step_config.rs
      │  │  ├─ initialize_config.rs
      │  │  ├─ initialize_quote_asset_badge.rs
      │  │  ├─ mod.rs
      │  │  ├─ set_hook.rs
      │  │  ├─ transfer_config_ownership.rs
      │  │  ├─ update_bin_step_config.rs
      │  │  ├─ update_pair_static_fee_parameters.rs
      │  │  ├─ update_quote_asset_badge.rs
      │  │  └─ withdraw_protocol_fees.rs
      │  ├─ close_position.rs
      │  ├─ create_position.rs
      │  ├─ decrease_position.rs
      │  ├─ increase_position.rs
      │  ├─ initialize_bin_array.rs
      │  ├─ initialize_pair.rs
      │  ├─ mod.rs
      │  └─ swap.rs
      ├─ manager
      │  ├─ liquidity_manager.rs
      │  ├─ mod.rs
      │  ├─ swap_manager.rs
      │  ├─ token_manager.rs .4 math
      │  ├─ bin_math.rs
      │  ├─ mod.rs
      │  ├─ u128x128_math.rs
      │  ├─ u64x64_math.rs
      │  └─ utils.rs
      ├─ state
      │  ├─ bin.rs
      │  ├─ bin_array.rs
      │  ├─ bin_step_config.rs
      │  ├─ config.rs
      │  ├─ fee_parameters.rs
      │  ├─ mod.rs
      │  ├─ pair.rs
      │  ├─ position.rs
      │  └─ quote_asset_badge.rs
      ├─ constants.rs
      └─ errors.rs
```

```
│
├── events.rs
└── lib.rs
```