Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Saros DLMM**

Version: 1.0 (final report)
Date: 3rd October 2025
Commit/Hash: bf65a366b1557117022a7a19c2e7a8fc5ac12e35
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of a security assessment of the reward feature of the `Saros DLMM` Smart Contract. The engagement was commissioned by `Saros DLMM` to obtain an independent evaluation of the chain logic that distributes the reward tokens to all liquidity providers on the `Saros DLMM` system.

1.1 Objective

This audit was conducted to assess the robustness, reliability, and security of the code base. 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 23rd September 2025.

After the initial audit report, a reassessment was conducted on 3rd October 2025 to verify the status of the reported issues. While some issues were addressed, others were acknowledged and will be prioritized for future improvement.

1.2 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. Further, economic modeling, business-logic desirability, and market comparisons (e.g., target price ceilings relative to other launch platforms) were explicitly out of scope.

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 bf65a366b1557117022a7a19c2e7a8fc5ac12e35 from git repository `https://github.com/saros-xyz/saros-sol-liquidity-book`. Further details regarding The `Saros DLMM` audit scope are provided below:

- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: bf65a366b1557117022a7a19c2e7a8fc5ac12e35
  - Reassessment audit's commit hash: b5a5e9afa0b548b616b4c1f9c6699716a0b19427

3 Audit Summary

Our first pass identified 4 distinct (Critical/High/Medium) issues across the `Saros DLMM` program. The overview of the initial findings is presented in Table 1.

Severity	Count
Critical	1
High	1
Medium	2
To be discussed	1

Table 1: Initial assessment summary

Following fixes reviewed on 3rd October 2025, 5.1 Critical, 5.2 High, and 5.4 Medium are Fixed; 5.3 Medium and 5.5 Informational are Acknowledged (c.f. Table 2). Overall risk is materially reduced; remaining items center on authority/allowlisting and admin controls—address before expanding integrations.

Issue	Severity	Status	Remediation Evidence
5.1	Critical	Fixed	2953f0c
5.2	High	Fixed	4c0fa795
5.3	Medium	Acknowledged	
5.4	Medium	Fixed	b5a5e9a
5.5	Informational (Discussed)	Acknowledged	

Table 2: Final assessment summary

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 compute units, 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.
- **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 Unauthorized Reward Accounting Vulnerability

- **Description:** The `Hook Rewarder` contract's accounting function does not properly validate the signer (`pair`) account. This vulnerability allows a malicious actor to manipulate the signer account and directly call the rewarder hook, enabling them to claim reward tokens without proper authorization. This can lead to the unauthorized withdrawal of rewards, resulting in a loss of funds for legitimate users.
- **Risk:** Critical (Impact: High, Likelihood: High)
- **CWE:** CWE-732: Incorrect Permission Assignment for Critical Resource
- **Mitigation:** Implement a thorough validation check to ensure that the signer account is the legitimate account for triggering accounting functions.
- **Status [03/10/2025]:** `Saros DLMM` team has fixed this issue.

5.2 Sweep Function Drains All Funds Without Accounting Check

- **Description:** The `sweep()` function, when executed during the reward duration, can cause a mismatch between the platform's actual token reserve and the amount of unclaimed rewards calculated for users. This discrepancy could lead to a situation where the platform's reserve is insufficient to cover the total amount of tokens users are entitled to, potentially resulting in a loss for users and a broken incentive system.
- **Risk:** High (Impact: High, Likelihood: Medium)
- **CWE:** CWE-840: Business Logic Errors
- **Mitigation:**
  - If the `sweep()` function is used as an incident response tool, it must reset the reward rate (`reward_per_second`) to zero. This action effectively freezes the total amount of unclaimed tokens, which can then be used in the recovery phase.
  - If the `sweep()` function is used to claim the remaining token reserve after the reward campaign, users should be notified before the action is taken.
- **Status [03/10/2025]:** `Saros DLMM` team has fixed this issue.

5.3 Insecure Hook/Token Hook Integration

- **Description:** The current integration path for hook and reward-token-hook programs does not explicitly verify the program's authority or allowlisting prior to attachment. In the absence of these checks, an attacker could attempt to attach an unauthorized program, leading to unintended execution or state changes.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-494: Download of Code Without Integrity Check

- **Mitigation:**
  - `Implement an owner check`: The contract must verify that the public key of the program being applied is owned by the *Preset Authority*. This ensures that only programs with the correct and trusted upgrade authority can be integrated.
  - `Introduce a whitelist`: Implement a whitelist of public keys that are authorized to deploy hook programs. This list should be managed exclusively by the Preset Authority, ensuring that only trusted entities can create and introduce new hook programs to the system.
  - `Verify programs before integration`: All hook and reward token hook programs must undergo a thorough review and verification process before they are integrated with the main contract. This should include a security audit or a formal code review to ensure their integrity and functionality.
- **Status [03/10/2025]:** `Saros DLMM` team has acknowledged this issue.

5.4 Potentially Cross-Pair Reward Manipulation

- **Description:** During the `initialize_position` instruction, the program fails to enforce a link between the `hook` account and the `lb_position` account. This missing constraint allows an attacker to create a malicious hook position using a `lb_position` account from a different trading pair than the one associated with the hook account. This vulnerability can lead to the manipulation of reward token distribution.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-862: Missing Authorization
- **Mitigation:** Implement a validation check during the `initialize_position` instruction to ensure that the hook account and `lb_position` account are correctly associated with the same trading pair. This will prevent malicious actors from using positions from other pairs to manipulate the reward system.
- **Status [03/10/2025]:** `Saros DLMM` team has fixed this issue.

5.5 Centralization Risk

- **Description:** At the `Saros DLMM` smart contract, the admin address holds the authority to perform critical, sensitive actions. If the admin's private key is compromised, or if the admin decides to act maliciously, the entire protocol and its user base are at risk.
- **Initial Risk:** To be discussed
  - Medium (Impact: High, Likelihood: Low) if there is no solution applied to prevent admin compromise.
- **Finalized Risk:** Informational

- **Mitigation:** Multi-sig wallet should be used for mitigating this issue as it requires multiple trusted parties to approve transactions, distributing control and making it much harder for a single point of failure to cause damage.

- **Status [03/10/2025]:** `Saros DLMM` team has acknowledged this issue. Admin authority is held by a multisig wallet. We recommend publishing the admin/multisig policy and (optionally) enforcing a timelock for sensitive actions.