Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Saros Staking**

Version: 1.0 (final report)
Date: 3rd July 2025
Commit/Hash: 1baa5ddf8cb70edb3051fcd631b5f33ffd4bdfdc
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of a security assessment of the Saros Staking program. The engagement was commissioned by Saros to obtain an independent evaluation of the on-chain logic that locks tokens to help secure the network and distributes passive rewards.

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 10th June 2025.

After the initial audit report, a reassessment was conducted on 3rd July 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 1baa5ddf8cb70edb3051fcd631b5f33ffd4bdfdc from git repository `https://github.com/saros-xyz/saros-stake-sol`. Further details regarding The `Saros Staking` audit scope are provided below:

- **Smart contracts audited:** See Appendix.
- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: 14c216424b0f76544f3141ed522702cfbfebcf23
  - Reassessment's commit hash: 1baa5ddf8cb70edb3051fcd631b5f33ffd4bdfdc
- **Deploying state:** Table 1 shows the current address of the smart contract deployed on Solana.

3 Audit Summary

Our first pass identified 8 distinct issues across the `Saros Staking` program:

Severity	Count
High	1
Medium	5
Informational	2

Table 1: Initial assessment summary

After code updates and design clarifications from the developers, we re-audited the patched build (c.f. Table 2).

Issue	Severity	Status	Remediation commit
5.1	High	Fixed	dd9ca9a
5.2	Medium	Acknowledged	
5.3	Medium	Fixed	4e444e2
5.4	Medium	Fixed	45c51cd
5.5	Medium	Fixed	c63e36a
5.6	Medium	Fixed	5464c5d
5.7	Informational	Fixed	edb16be
5.8	Informational	Fixed	1b6b3f8

Table 2: Final assessment summary

Most risks have been eliminated; the code base now carries only a medium issue. However, the issue has been confirmed to be unlikely to be exploited due to the current configuration.

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 Incorrect total_shares Calculation

- **Description:** The *unstake_pool_reward()* instruction incorrectly increments *total_shares* of a *pool_reward* instead of decrementing it. This happens when a user unstakes, which should reduce the total_shares in the pool.
- **Risk:** High (Impact: Medium, Likelihood: High)
- **CWE:** CWE-440: Expected Behavior Violation
- **Affected code:**

```
242     if user_pool_reward.amount > 0 {
243       let amount = user_pool_reward.amount;
244       pool_reward.total_shares = pool_reward.total_shares.
    checked_add(amount).unwrap();
245       user_pool.total_staked = user_pool.total_staked.checked_sub(
    amount).unwrap();
246
```

programs/saros-stake/src/lib.rs:244

- **Mitigation:** Review the *unstake_pool_reward()* instruction's implementation. Focus on the logic that updates *pool_reward.total_shares* and ensure that it correctly subtracts the unstaked amount instead of adding it.
- **Status [03/07/2025]:** `Saros Staking` team has fixed this issue.

5.2 Potential for Double Staking Vulnerability

- **Description:** The *stake_pool_reward()* instruction does not reset *user_pool.amount* to zero. This allows a user to continue staking in multiple reward pools simultaneously if two or more are active.
- **Exploit scenario:** If a configuration error ever allows multiple staking reward pools to be active at once, an attacker can flash-borrow a large amount of the staking token, stake the borrowed balance in each pool within the same block, immediately unstake to repay the loan, yet leave their recorded stake (user_pool.amount) unchanged in both pools. Because that value is never zeroed out after rewards are claimed, the attacker keeps a phantom share in every live pool and can call the claim function repeatedly to siphon future rewards without restaking—diluting legitimate stakers and, over time, potentially draining the reward pot.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-754: Improper Check for Unusual or Exceptional Conditions
- **Affected code:** *programs/saros-stake/src/lib.rs:stake_pool_reward()*

- **Mitigation:** Thoroughly review the execution flow and the logic for calculating user information within the *stake_pool_reward()* instruction
- **Status [03/07/2025]:** `Saros Staking` team has acknowledged this issue and confirmed the multiple pool reward will not exist as it was not intended by the business.

5.3 Insufficient Authority Management

- **Description:** The `Saros Staking` smart contract lacks proper Ownership Management. Hard-coding a list of specific addresses could lead to a loss of control if those addresses become compromised.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-654: Reliance on a Single Factor in a Security Decision
- **Mitigation:** Review the authority management mechanism and complete its implementation.
- **Status [03/07/2025]:** `Saros Staking` team has fixed this issue.

5.4 User-Supplied PDA Bump Allows Misdirected Authority

- **Description:** The `Saros Staking` implements a feature to store PDA (Program Derived Address) bumps to optimize computation costs. However, the bump value is provided by the user rather than being derived and validated internally by the program. This approach introduces a risk where the user-provided bump value might differ from the actual bump calculated by the Anchor framework. If these values mismatch, it could lead to the loss of control over the PDA.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-348: Use of Less Trusted Source
- **Affected code:**
  - *programs/saros-stake/src/lib.rs:82*
  - *programs/saros-stake/src/lib.rs:155*
- **Mitigation:** Instead of relying on user input, the stored bump should be gathered from Anchor framework.
- **Status [03/07/2025]:** `Saros Staking` team has fixed this issue.

5.5 Missing ownership checking when working with token account

- **Description:** The `Saros Staking` declares token account as an **AccountInfo**. This implementation, by lacking ownership checking for accounts, could lead to unintended system behavior.
- **Risk:** Medium (Impact: Medium, Likelihood: Medium)

- **CWE:** CWE-1173: Improper Use of Validation Framework
- **Affected code:**
  - *programs/saros-stake/src/context.rs:209*
  - *programs/saros-stake/src/context.rs:401*
- **Mitigation:** Implement ownership verification for the provided token accounts.
- **Status [03/07/2025]:** `Saros Staking` team has fixed this issue.

5.6 Using unwrap() Insecurely

- **Description:** In the `Saros Staking` smart contract, numerous computing functions (e.g., *checked_add()*, *checked_sub()*, ...) use unwrap() insecurely. This issue could lead to a panic.
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-248: Uncaught Exception
- **Affected code:**
  - *programs/saros-stake/src/lib.rs:201*
  - *programs/saros-stake/src/lib.rs:219*
  - *programs/saros-stake/src/lib.rs:220*
  - *programs/saros-stake/src/lib.rs:221*
  - *programs/saros-stake/src/lib.rs:244*
  - *programs/saros-stake/src/lib.rs:245*
  - *programs/saros-stake/src/lib.rs:372*
  - *programs/saros-stake/src/state.rs:44*
  - *programs/saros-stake/src/state.rs:45*
  - *programs/saros-stake/src/state.rs:47*
  - *programs/saros-stake/src/state.rs:63*
  - *programs/saros-stake/src/state.rs:66*
  - *programs/saros-stake/src/state.rs:70*
- **Mitigation:** Handle the returned error instead of using unwrap().
- **Status [03/07/2025]:** `Saros Staking` team has fixed this issue.

5.7 Optimizing - Implementing Anchor Framework for Computing ATA

- **Description:** The `Saros Staking` calculates an Associated Token Account (ATA) address manually. Current implementation could be improved to be cleaner and more efficient by using Anchor framework.
- **Risk:** Informational (Impact: Low, Likelihood: Low)
- **Mitigation:** Using Anchor Framework instead of computing manually.
- **Status [03/07/2025]:** `Saros Staking` team has applied this suggestion.

5.8 Optimizing - Using Pubkey for Storing Authority Addresses

- **Description:** The `Saros Staking` uses **str** type for declaring authority addresses. It could be more memory effective by using **Pubkey**.
- **Risk:** Informational (Impact: Low, Likelihood: Low)
- **Mitigation:** Using **Pubkey** for Storing Authority Addresses instead of **str**
- **Status [03/07/2025]:** `Saros Staking` team has applied this suggestion.

Appendix

```
programs/
└── saros-stake
    └── src
        ├── constants.rs
        ├── context.rs
        ├── error.rs
        ├── event.rs
        ├── lib.rs
        ├── state.rs
        └── utils.rs
```