Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Saros Point Token Hook**

Version: 1.0 (Final Report)
Date: 3rd July 2025
Commit/Hash: 595d626762a4da782d90bf44fdfc818bd34771c6
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of a security assessment of the Saros Token Hook program. The engagement was commissioned by Saros to obtain an independent evaluation of the on-chain logic that manages address whitelisting and blacklisting during token transfers.

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 4th 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 595d626762a4da782d90bf44fdfc818bd34771c6 from git repository `https://github.com/saros-xyz/point-token-hook-sol`. Further details regarding The `Saros Point Token Hook` audit scope are provided below:

- **Smart contracts audited:** See Appendix.
- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: 568ba07797abc115663c0bdf535e45d73d763f01
  - Reassessment's commit hash: 595d626762a4da782d90bf44fdfc818bd34771c6
- **Deploying state:** Table 1 shows the current address of the smart contract deployed on Solana.

Deploying Smart Contract	Address
Saros Point Token Hook	TKNsE42yoLb4sE6uyr2c4ixjsk6ufSPFR94K8kRHzUp

Table 1: Deploying smart contract address

3 Audit Summary

Our first pass identified 4 distinct issues across the `Saros Point Token Hook` program:

Severity	Count
Medium	1
Low	1
Informational	2

Table 2: Initial assessment summary

After code updates and design clarifications from the developers, we re-audited the patched build (c.f. Table 3).

Issue	Severity	Status	Remediation commit
5.1	Medium	Fixed	27b9d59
5.2	Low	Fixed	88e28f3
5.3	Informational	Fixed	e7f7b7a
5.4	Informational	Fixed	3b64296

Table 3: Final assessment summary

All issues discovered during the assessment have been resolved

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 4 shows the details of the security severity assessment for each issue.

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

Table 4: 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 5 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 5: Details of examined items

5 Detailed Findings

5.1 Using unwrap() Insecurely

- **Description:** In the `Saros Point Token Hook` smart contract, numerous procedures (e.g., *checked_add()*, *try_borrow_data()*, ...) use *unwrap()* insecurely. This issue could lead to a panic
- **Risk:** Medium (Impact: High, Likelihood: Low)
- **CWE:** CWE-248: Uncaught Exception
- **Affected code:**
  - *programs/point-token-hook/src/util.rs:61*
  - *programs/point-token-hook/src/util.rs:101*
  - *programs/point-token-hook/src/util.rs:166*
- **Mitigation:** Handle the returned error instead of using unwrap().
- **Status [04/07/2025]:** `Saros Point Token Hook` team has fixed this issue.

5.2 Missing Status Verification Before PDA Closure

- **Description:** The *close_pda()* function in *programs/point-token-hook/src/utils.rs* lacks a crucial check to determine if the Program Derived Address (PDA) has been initialized. This omission can lead to a program error (panic or unintended behavior) if an attempt is made to close a PDA that is not in an initialized state.
- **Risk:** Low (Impact: Medium, Likelihood: Low)
- **CWE:** CWE-754: Improper Check for Unusual or Exceptional Conditions
- **Affected code:** *programs/point-token-hook/src/util.rs:close_pda()*
- **Mitigation:** Add access account check before closing PDA
- **Status [04/07/2025]:** `Saros Point Token Hook` team has fixed this issue.

5.3 Missing Remove Pending Address after Accept Ownership

- **Description:** The *accept_ownership* function fails to clear or reset the *pending_address* after a new owner successfully claims ownership. This means the *pending_address* field retains the value of the newly accepted owner, rather than being set to a default or null state.
- **Risk:** Informational (Impact: Low, Likelihood: Low)
- **Mitigation:** After a successful *accept_ownership* operation, the *pending_address* field should be explicitly reset to *Pubkey::default()*
- **Status [03/07/2025]:** `Saros Point Token Hook` team has applied this suggestion.

5.4 Optimizing - Using Pubkey for Storing Authority Addresses

- **Description:** The `Saros Point Token Hook` 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 Point Token Hook` team has applied this suggestion.

Appendix

```
programs/
└── point-token-hook
    └── src
        ├── constants.rs
        ├── context.rs
        ├── error.rs
        ├── lib.rs
        ├── state.rs
        └── utils.rs
```