Smart Contract Audit Report

Prepared by: **Cybring**
Prepared for: **Saros Token Proxy**

Version: 1.0 (final report)
Date: 4th July 2025
Commit/Hash: d0a0578f896e460ae3b2d4f11733c52146ea6860
Auditor: Anh Bui, Duc Cuong Nguyen

Contents

1 Introduction

This document presents the results of a security assessment of the Saros Token Proxy program. The engagement was commissioned by Saros to obtain an independent evaluation of the on-chain logic that extends the capabilities of the SPL Token standard by enabling a group of approved addresses to mint tokens, instead of relying on a single mint authority.

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 d0a0578f896e460ae3b2d4f11733c52146ea6860 from git repository `https://github.com/saros-xyz/saros-token-proxy-sol`. Further details regarding The `Saros Token Proxy` audit scope are provided below:

- **Smart contracts audited:** See Appendix.
- **Codebase details:**
  - Language: Rust
  - Frameworks: Anchor
  - Initial audit's commit hash: cd92c4bf3bf6a3d80260c8d39a0ffae733e34f01
  - Reassessment's commit hash: d0a0578f896e460ae3b2d4f11733c52146ea6860
- **Deploying state:** Table 1 shows the current address of the smart contract deployed on Solana.

Deploying Smart Contract	Address
Saros Token Proxy Program	prxtZK1VbkooonzQ1rNjW8qZBP1fm5WkdtuCzLmi87e

Table 1: Deploying smart contract address

3 Audit Summary

Our first pass identified 3 distinct issues across the `Saros Token Proxy` program:

Severity	Count
High	2
Informational	1

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	High	Fixed	d776aa
5.2	High	Fixed	5dd77e6
5.3	Informational	Fixed	90e5362

Table 3: Final assessment summary

All risks have been eliminated.

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.

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

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 Insufficient Authority Management

- **Description:** The Saros Token Proxy 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:** High (Impact: High, Likelihood: medium)
- **CWE:** CWE-654: Reliance on a Single Factor in a Security Decision
- **Mitigation:** Review the authority management mechanism and complete its implementation.
- **Status [04/07/2025]:** Saros Token Proxy team has fixed this issue.

5.2 Insufficient Size Control for authorities Vector

- **Description:** The *add_authority()* function allows new addresses to be added to the authorities vector without performing checks on the vector's current size. The account allocation fits a maximum of 10 public keys; exceeding this limit triggers a runtime error.
- **Risk:** High (Impact: High, Likelihood: Medium)
- **CWE:** CWE-770: Allocation of Resources Without Limits or Throttling
- **Mitigation:** Implement a clear size check within the *add_authority()* function.
- **Status [04/07/2025]:** Saros Token Proxy team has fixed this issue.

5.3 Optimizing - Implementing Anchor Framework for Computing ATA

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

Appendix

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