0xCommit

Security Audit Report

SOEX - Solana Programs

Staking HVT

Version: Final

Date: 4th Jan 2025

Table of Contents

License

Disclaimer

Introduction

Purpose of this report

0xCommit has been engaged by **SOEX - Solana Programs** to perform a security audit of several Solana Programs components.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine solana program bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Version	List of programs	Source
1	Staking HVT	https://gitlab.dcircle.io/dcchain/soex_solana_x/-/tree/master/programs

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Summary of Findings

Sr. No.	Program	Description	Severity	Status
1	Staking HVT	Unprotected init function	Medium	Ackno...
2	Staking HVT	Missing Signature verification	Medium	Ackno...
3	Staking HVT	Insufficient timestamp validation	High	Resolv...

Detailed Findings

1. Unprotected init function

Severity: **Medium** **Program - Staking HVT**

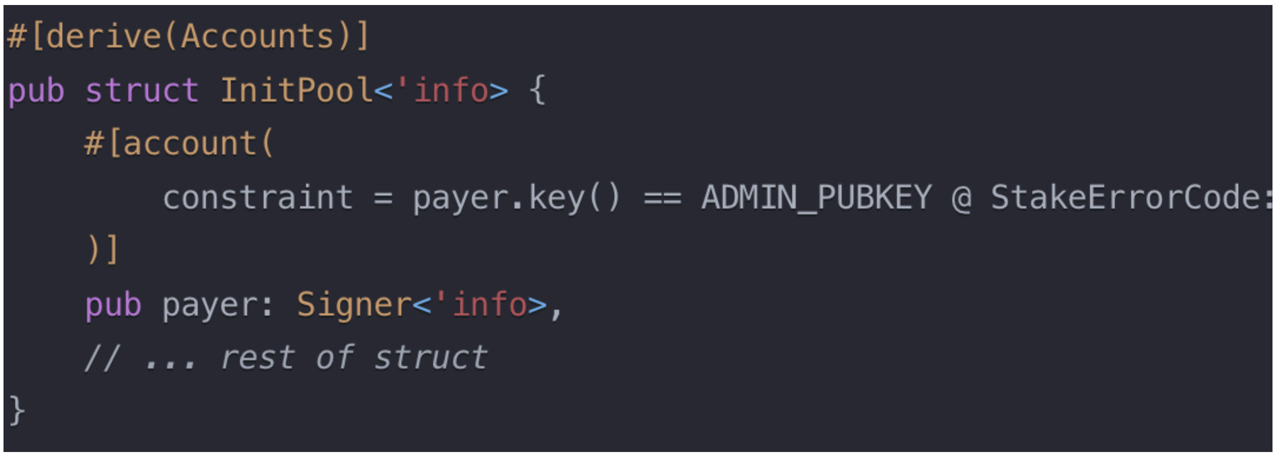
Description

A critical vulnerability has been identified in the SOEX staking contract where initialization functions lack proper access controls. This vulnerability could allow unauthorized parties to initialize the staking pool with malicious parameters.

Remediation

Following are the remediation for the issue

- Implement admin public key which can initialise the program.



```
#[derive(Accounts)]
pub struct InitPool<'info> {
    #[account(
        constraint = payer.key() == ADMIN_PUBKEY @ StakeErrorCode:
    )]
    pub payer: Signer<'info>,
    // ... rest of struct
}
```

Status

Acknowledged

2. Missing Signature verification

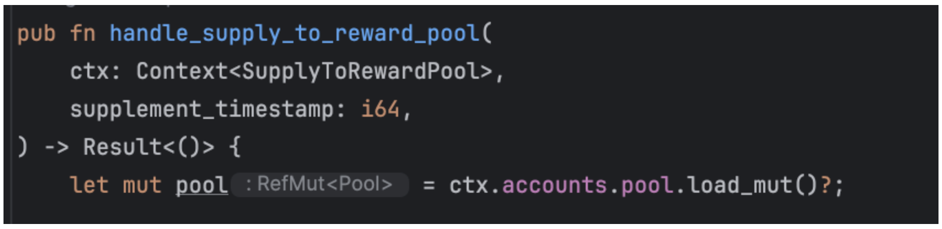
Severity: **Medium** **Program - Staking HVT**

Description

Multiple endpoints in the staking protocol lack proper signer verification checks, potentially allowing unauthorized users to execute privileged operations.

Location :

- File:"src/instruction/handle_supply_to_reward_pool.rs" → Fn "Handle_supply_to_reward_pool"

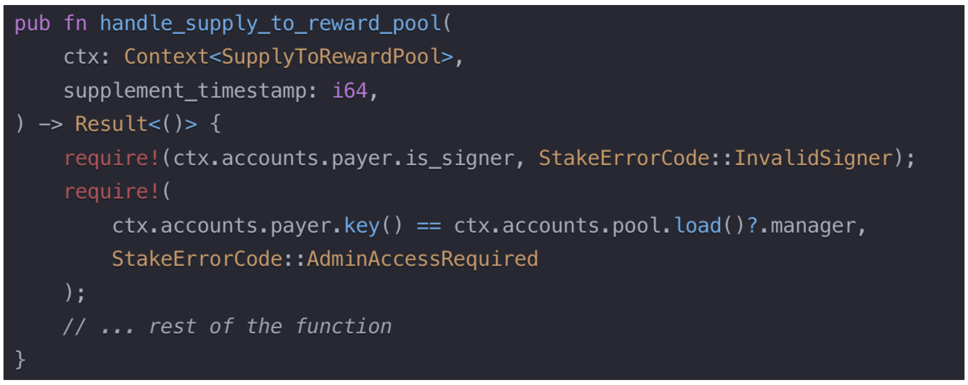


```
pub fn handle_supply_to_reward_pool(
    ctx: Context<SupplyToRewardPool>,
    supplement_timestamp: i64,
) -> Result<()> {
    let mut pool : RefMut<Pool> = ctx.accounts.pool.load_mut()?;
```

Remediation

Following are the remediation for the issue

- Implement authority checks



```
pub fn handle_supply_to_reward_pool(
    ctx: Context<SupplyToRewardPool>,
    supplement_timestamp: i64,
) -> Result<()> {
    require!(ctx.accounts.payer.is_signer, StakeErrorCode::InvalidSigner);
    require!(
        ctx.accounts.payer.key() == ctx.accounts.pool.load()?.manager,
        StakeErrorCode::AdminAccessRequired
    );
    // ... rest of the function
}
```

Status

Acknowledged

3. Insufficient timestamp validation

Severity: High **Program - Staking HVT**

Description

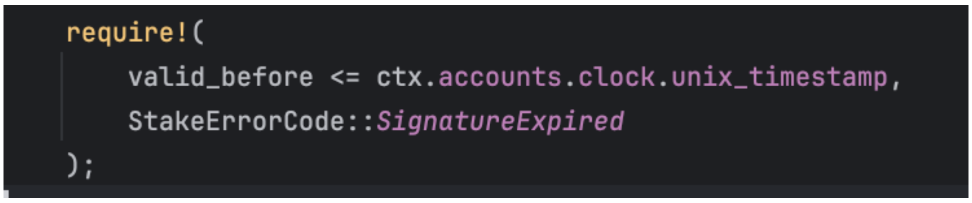
The issue stems from incorrect timestamp validation logic in the claim reward function, which could allow expired signatures to be accepted and potentially lead to replay attacks.

The vulnerability could allow attackers to:

- Replay expired signatures indefinitely
- Execute transactions outside their intended validity period
- Potentially drain funds through repeated unauthorized claims
- Manipulate market conditions through transaction timing exploitation

Location

File: src/instructions/claim_reward.
Function: handle_claim
Lines: 41-45



```
require!(
    valid_before <= ctx.accounts.clock.unix_timestamp,
    StakeErrorCode::SignatureExpired
);
```

(Note :- The logic is inverted, it functions as after not before)

Remediation

Following are the remediation

- Implement correct timestamp comparison logic

  "ctx.accounts.clock.unix_timestamp <= valid_before,"

Status

Resolved