// HALBORN

Solana - Stake Pool

Solana Program Security Assessment

Prepared by: **Halborn**

Date of Engagement: **December 11th, 2023 - December 31st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	12/31/2023
0.2	Draft Version	01/08/2024
0.3	Draft Review	01/08/2024
0.4	Draft Review	01/09/2024
1.0	Remediation Plan	01/17/2024
1.1	Remediation Plan Review	01/17/2024
1.2	Remediation Plan Review	01/18/2024
1.3	Remediation Plan Updates	01/22/2024
1.4	Remediation Plan Updates Review	01/22/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

The Solana stake-pool program allows for the creation of stake pools, which introduce the ability for multiple stakeholders to aggregate their stake into a single pool. This benefits smaller stakeholders, who can participate in the staking process without needing to run their own validator node, increasing network security and health. The stake-pool program allows liquid-staking, where pool tokens are given to the staker in exchange for their SOL, depending on the proportion of stake they have supplied to the pool. This improves liquidity for stakers since these tokens can be used throughout the ecosystem, whereas traditional staking would require the users' tokens to be locked.

Solana engaged Halborn to conduct a security assessment on their Solana programs, beginning on December 11th, 2023 and ending on December 31st, 2023 . The security assessment was scoped to a few pull requests to the stake-pool program provided in the solana-program-library GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided two weeks and a half for the engagement and assigned one full-time security engineer to review the security of the programs in scope. The security engineer is a blockchain and Solana program security expert with advanced penetration testing and Solana program hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Identify potential security issues within the programs

In summary, Halborn identified one security risk that will be addressed

by the Solana team in a future release.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture, purpose, and use of the platform.
- Manual program source code review to identify business logic issues.
- Mapping out possible attack vectors
- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.
- Scanning dependencies for known vulnerabilities (cargo audit).
- Local runtime testing (solana-test-framework)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. Solana stake-pool program

- Repository: solana-program-library
- Pull Requests in scope:
  - #5285
  - #5288
  - #5322
- Programs in scope:
  1. stake-pool (/stake-pool/program)

Out-of-scope:
- third-party libraries and dependencies
- financial-related attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	0

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) USERS COULD AVOID FEES BY DEPOSITING SMALL AMOUNTS OF SOL	Low (2.5)	SOLVED - 1/22/2024

FINDINGS & TECH DETAILS

4.1 (HAL-01) USERS COULD AVOID FEES BY DEPOSITING SMALL AMOUNTS OF SOL - LOW (2.5)

Description:

In the process_deposit_sol function, there are multiple fees which are deducted from the pool_tokens which will be minted to the user and the sol deposited to the stake account. In the case the Sol deposited is small, it is possible the fee calculations round down to zero due to loss of precision, preventing fee collection even though the user has deposited SOL and been minted pool_tokens in return. However, this scenario would not be economically viable, and the cost of fee avoidance would outweigh the benefit of depositing such a small amount to the stake pool.

Code Location:

Listing 1: stake-pool/program/src/processor.rs

```
3051 let new_pool_tokens = stake_pool
3052     .calc_pool_tokens_for_deposit(deposit_lamports)
3053     .ok_or(StakePoolError::CalculationFailure)?;
3054
3055 // @test - possible rounding down, preventing fee from being
  ↳ collected
3056 let pool_tokens_sol_deposit_fee = stake_pool
3057     .calc_pool_tokens_sol_deposit_fee(new_pool_tokens)
3058     .ok_or(StakePoolError::CalculationFailure)?;
3059
3060 let pool_tokens_user = new_pool_tokens
3061     .checked_sub(pool_tokens_sol_deposit_fee)
3062     .ok_or(StakePoolError::CalculationFailure)?;
3063 // @audit - referral_fee will == 0 in case
  ↳ pool_tokens_sol_deposit_fee == 0
3064 //
3065 let pool_tokens_referral_fee = stake_pool
3066     .calc_pool_tokens_sol_referral_fee(pool_tokens_sol_deposit_fee
  ↳ )
3067     .ok_or(StakePoolError::CalculationFailure)?;
```

```
3068 let pool_tokens_manager_deposit_fee = pool_tokens_sol_deposit_fee
3069     .checked_sub(pool_tokens_referral_fee)
3070     .ok_or(StakePoolError::CalculationFailure)?;
```

Listing 2: stake-pool/src/state.rs

```rust
pub fn apply(&self, amt: u64) -> Option<u128> {
    if self.denominator == 0 {
        return Some(0);
    }
    (amt as u128)
        .checked_mul(self.numerator as u128)?
        .checked_div(self.denominator as u128)
}
```

Proof Of Concept:

Listing 3

```rust
fn test_fee() {
    let fee = Fee {
        denominator: 100,
        numerator: 2
    };

    let minted_pool_tokens: u64 = 2;
    let fee_applied = fee.apply(minted_pool_tokens).unwrap();
    assert!(fee_applied != 0);
}
```

The following test will fail even though the minted_pool_tokens != 0. Since the apply function is used for multiple fee calculations in the process_deposit_sol function, the same behavior will occur for multiple fee calculations.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:L/R:N/S:U (2.5)

Recommendation:

Include check that ensures fee != 0 for fees collected in process_deposit_sol

Remediation Plan:

SOLVED: This finding was remediated in commit 79243a3b874, which introduces fee rounding in favor of the pool, preventing the fees from rounding down to zero.

MANUAL TESTING

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

Scenario	Expectation	Result
can call initialize with arbitrary address, who isnt signer	tx fails	pass
possible to pass in stake-pool not owned by stake-pool program	tx fails	pass
possible to pass in stake-pool that is already initialized	tx fails	pass
possible to use validator list not owned by stake-pool program	tx fails	pass
possible to set max validators == 0	tx fails	pass
possible to implement fee > 100possible to add duplicate validator to validator list	tx fails	pass
fee rounds in favor of stake-pool, not user	fee cannot round down to zero	fail
possible to pass in invalid pool mint, != legitimate mint	tx fails	pass
possible to initialize pool mint with supply != 0	tx fails	pass
deposit_authority can collide with authority for a different pool	pda collision not possible	pass

THANK YOU FOR CHOOSING

// HALBORN