// Security Assessment

Invalid date - 02.14.2025

Mining Pool
Nous Research

HALBORN

Mining Pool - Nous Research

Prepared by: HALBORN

Last Updated 03/06/2025

Date of Engagement by: Invalid date - February 14th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	0	0	1	4

TABLE OF CONTENTS

1. Introduction

Nous Research engaged Halborn to conduct a security assessment on their Mining Pool **Solana program** beginning on **February 12th, 2025**, and ending on **February, 18th, 2025**. The security assessment was scoped to the Solana program provided in NousResearch/psyche GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

The solana-mining-pool program implements a basic "mining" or lending pool mechanism where users (lenders) can deposit collateral into a pool, and eventually claim redeemable tokens proportionate to their share of the total deposited collateral. An authority (pool owner) can extract collateral from the pool, update the pool metadata, and enable claim functionality. The program also tracks each lender's deposited and claimed amounts.

2. Assessment Summary

Halborn was provided **6 days** for the engagement and assigned one full-time security engineer to review the security of the Solana Program in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the Co-Staking Solana Program.
- Ensure that the program's functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the Nous Research team. The main ones were the following:

- Use multi-signature wallets for these authorities accounts, in order to avoid potential unwarranted and unilateral actions and document administrative actions.
- If pool.total_deposited_collateral_amount == 0, return an error from claim logic rather than panicking.
- Add or modify an instruction to toggle or disable claims.
- Implement an instruction that subtracts from both lender.deposited_collateral_amount and pool.total_deposited_collateral_amount, and transfers the tokens back to the lender.
- Provide an instruction that checks the account’s final state and closes it, returning the SOL to the owner.

3. Test Approach And 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`).

4. 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.

4.1 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:

$$E = \prod m_e$$

4.2 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) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

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

4.3 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.

METRICS:

SEVERITY 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: psyche

(b) Assessed Commit ID: 7441712

(c) Items in scope:

- src/lib.rs
- src/logic/pool_update.rs
- src/logic/pool_create.rs
- src/logic/lender_claim.rs
- src/logic/pool_extract.rs
- src/logic/pool_claimable.rs
- src/logic/lender_create.rs
- src/logic/lender_deposit.rs
- src/state/lender.rs
- src/state/pool.rs

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- https://github.com/NousResearch/psyche/pull/230/commits/8859c76cbbab6f8806801071504b7100fe219226
- https://github.com/NousResearch/psyche/commits/8859c76cbbab6f8806801071504b7100fe219226

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	4

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
AUTHORITY CAN EXTRACT ANY AMOUNT OF COLLATERAL	LOW	SOLVED - 02/21/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NO LENDER WITHDRAWAL MECHANISM	INFORMATIONAL	ACKNOWLEDGED - 02/24/2025
DIVISION BY ZERO	INFORMATIONAL	SOLVED - 02/21/2025
CLAIMING CANNOT BE DISABLED	INFORMATIONAL	SOLVED - 03/21/2025
NO CLOSE INSTRUCTIONS	INFORMATIONAL	ACKNOWLEDGED - 02/24/2025

7. FINDINGS & TECH DETAILS

7.1 AUTHORITY CAN EXTRACT ANY AMOUNT OF COLLATERAL

// LOW

Description

The `pool_extract` instruction enables authority to transfer arbitrary amounts of collateral from the `pool_collateral` account to their own account. While this may be the intended behavior, it is important to document and advice users thoroughly.

`psyche/architectures/decentralized/solana-mining-pool/programs/solana-mining-pool/src/logic/pool_extract.rs`

```
pub fn pool_extract_processor(
    context: Context<PoolExtractAccounts>,
    params: PoolExtractParams,
```

BVSS

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

Recommendation

It is recommended to document this behavior extensively. As the design is intentionally centralized, make sure depositors understand this trust requirement. Additionally, it is recommended to use multi-signature wallets for these authorities accounts, in order to avoid potential unwarranted and unilateral actions.

Remediation Comment

SOLVED: The **Nous Research team** has solved this issue, as recommended, by implementing a multi-signature wallet. The commit hash for reference is `8859c76`.

Remediation Hash

https://github.com/NousResearch/psyche/pull/230/commits/8859c76cbbab6f88068010715O4b7100fe219226

7.2 NO LENDER WITHDRAWAL MECHANISM

// INFORMATIONAL

Description

Once a lender deposits collateral, there is no path for them to withdraw or redeem it. Only the pool authority can call the `pool_extract` instruction. This behavior is unusual in typical DeFi pools, where depositors often have a route to withdraw. From a user perspective, depositors have no on-chain recourse to recover their collateral unless the authority cooperates.

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:N/I:N/D:L/Y:N (1.3)

Recommendation

It is recommended to implement an instruction that subtracts from both `lender.deposited_collateral_amount` and `pool.total_deposited_collateral_amount`, and transfers the tokens back to the lender.
If the design intentionally disallows lender withdrawal, ensure it is well documented to avoid user confusion.

Remediation Comment

ACKNOWLEDGED: The Nous **Research team** acknowledged this finding.

7.3 DIVISION BY ZERO

// INFORMATIONAL

Description

In the `lender_claim_processor` function, the code divides by `pool.total_deposited_collateral_amount`. If this value is 0, the program panics because of division by zero.

`psyche/architectures/decentralized/solana-mining-pool/programs/solana-mining-pool/src/logic/lender_claim.rs`

```
    let claimable_redeemable_amount = u64::try_from(
        u128::from(total_repayed_redeemable_amount)
            * u128::from(lender.deposited_collateral_amount)
            / u128::from(pool.total_deposited_collateral_amount),
    )
    .unwrap();
```

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:L/I:N/D:N/Y:N (1.3)

Recommendation

If `pool.total_deposited_collateral_amount == 0`, return an error from claim logic rather than panicking.

Remediation Comment

SOLVED: The **Nous Research team** has solved this issue as recommended. The commit hash for reference is `8859c76`.

Remediation Hash

https://github.com/NousResearch/psyche/pull/230/commits/8859c76cbbab6f8806801071504b7100fe219226

7.4 CLAIMING CANNOT BE DISABLED

// INFORMATIONAL

Description

The `pool_claimable_processor` function sets `pool.claiming_enabled = true` and fails if it is already set. There is no instruction to revert it to false.

`psyche/architectures/decentralized/solana-mining-pool/programs/solana-mining-pool/src/logic/pool_claimable.rs`

```
pub fn pool_claimable_processor(
    context: Context<PoolClaimableAccounts>,
    _params: PoolClaimableParams,
) -> Result<()> {
    let pool = &mut context.accounts.pool;

    if pool.claiming_enabled {
        return err!(ProgramError::PoolClaimingEnabledIsTrue);
    }

    pool.claiming_enabled = true;
    pool.redeemable_mint = context.accounts.redeemable_mint.key();
    pool.total_claimed_redeemable_amount = 0;

    Ok(())
}
```

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:N/I:L/D:N/Y:N (1.3)

Recommendation

If it is intentional that once claims are enabled, there is no going back, clarify it in the specification or documentation. Otherwise, add a second instruction to toggle or disable claims.

Remediation Comment

SOLVED: The **Nous Research team** has solved this issue as recommended. The commit hash for reference is `8859c76`.

Remediation Hash

https://github.com/NousResearch/psyche/commits/8859c76cbbab6f8806801071504b7100fe219226

7.5 NO CLOSE INSTRUCTIONS

// INFORMATIONAL

Description

The program does not provide a way to close accounts (Lender or Pool) when they are no longer needed, which would let users or the authority reclaim the rent-exempt SOL used to create them.

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:N/I:L/D:N/Y:N (1.3)

Recommendation

Provide an instruction that checks the account's final state and closes it, returning the SOL to the owner.

Remediation Comment

ACKNOWLEDGED: The **Nous Research team** has acknowledged this finding.

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	CRATE	DESCCRIPTION
RUSTSEC-2024-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed255109-dalek`
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s Scalar29::sub/Scalar52::sub

---

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.