// Security Assessment

04.30.2025 - 05.05.2025

Treasurer Solana program

Nous Research

HALBORN

Treasurer Solana program - Nous Research

Prepared by: HALBORN

Last Updated 07/01/2025

Date of Engagement: April 30th, 2025 - May 5th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	0	0	3

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
   - 7.1 The instruction parameter collateral_amount_per_earned_point is not validated
   - 7.2 The program does not allow to transfer the authority of a run
   - 7.3 Risk of losing control over the run account
8. Automated Testing

1. Introduction

`Nous Research team` engaged `Halborn` to conduct a security assessment on their **Treasurer Solana program** beginning on May 13, 2025, and ending on May 15, 2025. The security assessment was scoped to the Solana Programs provided in psyche GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

The `Treasurer` program provides an incentive layer on top of the Psyche's coordinator program. Psyche is a system that enables distributed training of transformer-based AI models over the internet. The `Treasurer` program allows to create a training `Run` owned by the `Treasurer` program and provides instructions to manage the `Run`, deposit reward tokens into the treasury of each `Run` and distribute the tokens to eligible participants based on their earned computation points.

2. Assessment Summary

`Halborn` was provided 3 days for the engagement and assigned one full-time security engineer to review the security of the Solana Programs 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 Solana Programs.
- Ensure that smart contract functionality operates as intended.

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

- `Implement secure Run authority transfer.`
- `Make sure the Run authority signs the Run initialization instruction.`
- `Validate the collateral_amount_per_earned_point instruction parameter.`

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-program-test`)

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

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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
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: 7a7516b

(c) Items in scope:

- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/participant_claim.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/run_create.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/state/run.rs

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

FILES AND REPOSITORY

(a) Repository: psyche

(b) Assessed Commit ID: 6f684bf

(c) Items in scope:

- ./architectures/decentralized/solana-treasurer/Cargo.toml
- ./architectures/decentralized/solana-treasurer/rustfmt.toml
- ./architectures/decentralized/solana-treasurer/Cargo.lock
- ./architectures/decentralized/solana-treasurer/Anchor.toml
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/Cargo.toml
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/Xargo.toml
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/lib.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/mod.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/run_update.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/participant_claim.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/run_create.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/participant_create.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/logic/run_top_up.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/state/run.rs

- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/state/mod.rs
- ./architectures/decentralized/solana-treasurer/programs/solana-treasurer/src/state/participant.rs

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

REMEDIATION COMMIT ID:

- 7a7516b

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

INFORMATIONAL
3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
THE INSTRUCTION PARAMETER COLLATERAL_AMOUNT_PER_EARNED_POINT IS NOT VALIDATED	INFORMATIONAL	SOLVED - 05/21/2025
THE PROGRAM DOES NOT ALLOW TO TRANSFER THE AUTHORITY OF A RUN	INFORMATIONAL	FUTURE RELEASE
RISK OF LOSING CONTROL OVER THE RUN ACCOUNT	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 THE INSTRUCTION PARAMETER COLLATERAL_AMOUNT_PER_EARNED_POINT IS NOT VALIDATED

// INFORMATIONAL

Description

The `run_create` instruction allows users to create new training `Run`s and configure them using various parameters. However, it does not validate the `collateral_amount_per_earned_point` parameter. If this value is mistakenly set to 0, users will be unable to claim any earned collateral. On the other hand, setting it too high could result in incorrect reward distribution and potentially drain the `Run`'s treasury.

run_create.rs

```
72  pub fn run_create_processor(
73      context: Context<RunCreateAccounts>,
74      params: RunCreateParams,
75  ) -> Result<()> {
76      let run = &mut context.accounts.run;
77      run.bump = context.bumps.run;
78      run.index = params.index;
79
80      run.main_authority = params.main_authority;
81      run.join_authority = params.join_authority;
82
83      run.coordinator_instance = context.accounts.coordinator_instance.key();
84      run.coordinator_account = context.accounts.coordinator_account.key();
85
86      run.collateral_mint = context.accounts.collateral_mint.key();
87      run.collateral_amount_per_earned_point =
88          params.collateral_amount_per_earned_point;
```

BVSS

AO:S/AC:L/AX:L/R:P/S:U/C:N/A:C/I:N/D:M/Y:N (1.1)

Recommendation

To address this issue, it is recommended to validate the `collateral_amount_per_earned_point` parameter and ensure it falls within an appropriate and expected range.

Remediation Comment

SOLVED: The **Nous Research team** solved this finding by removing the `collateral_amount_per_earned_point` parameter and distributing one collateral per earned point.

Remediation Hash

https://github.com/PsycheFoundation/psyche/commit/7a7516b3c107ca5ea073ff033737226490569769

7.2 THE PROGRAM DOES NOT ALLOW TO TRANSFER THE AUTHORITY OF A RUN

// INFORMATIONAL

Description

The instruction `run_create` allows users to create new training `Run` s and simultaneously set an authority who will be authorized to manage the run. However, the program does not provide an instruction to transfer the old authority to a new one. This might be useful, for example in the case of compromised private key to prevent unauthorized actions.

run_create.rs

```
72  pub fn run_create_processor(
73      context: Context<RunCreateAccounts>,
74      params: RunCreateParams,
75  ) -> Result<()> {
76      let run = &mut context.accounts.run;
77      run.bump = context.bumps.run;
78      run.index = params.index;
79
80      run.main_authority = params.main_authority;
81      run.join_authority = params.join_authority;
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To address this issue, it is recommended to implement authority transfer in one of two ways: either as a single instruction requiring signatures from both the current and new authorities, or as a two-step process where the current authority proposes the transfer in the first instruction, and the new authority confirms it with their signature in a second instruction.

Remediation Comment

FUTURE RELEASE: The **Nous Research team** plans to implement an authority transfer instruction in a future release, should the need arise.

7.3 RISK OF LOSING CONTROL OVER THE RUN ACCOUNT

// INFORMATIONAL

Description

The `run_create` instruction allows users to create new training Runs and assign an authority to manage them. However, it does not require a signature from the newly assigned authority, which is passed only as an instruction parameter. This could lead to a loss of control over the Run account if an incorrect authority is mistakenly set—especially if the corresponding private key is unavailable.

run_create.rs

```
63  #[derive(AnchorSerialize, AnchorDeserialize, Clone)]
64  pub struct RunCreateParams {
65      pub index: u64,
66      pub run_id: String,
67      pub main_authority: Pubkey,
68      pub join_authority: Pubkey,
69      pub collateral_amount_per_earned_point: u64,
70  }
71
72  pub fn run_create_processor(
73      context: Context<RunCreateAccounts>,
74      params: RunCreateParams,
75  ) -> Result<()> {
76      let run = &mut context.accounts.run;
77      run.bump = context.bumps.run;
78      run.index = params.index;
79
80      run.main_authority = params.main_authority;
81      run.join_authority = params.join_authority;
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To address this finding, it is recommended to require the signature of the run authority to initialize a new run.

Remediation Comment

ACKNOWLEDGED: The **Nous Research team** acknowledged the finding, noting that in some cases, such as when the receiver is a DAO, requiring a signature would necessitate a complex signing setup, reducing the practicality and usability of the instruction. Therefore, the original implementation will be retained.

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	DESCRIPTION
RUSTSEC-2025-0024	crossbeam-channel	crossbeam-channel: double free on Drop
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s `Scalar29::sub` / `Scalar52::sub`
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`
RUSTSEC-2025-0022	openssl	Use-After-Free in `Md::fetch` and `Cipher::fetch`
RUSTSEC-2023-0071	rsa	Marvin Attack: potential key recovery through timing sidechannels

---

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.