// Security Assessment

02.18.2026 - 02.25.2026

Solana Vaults

TruFin

HALBORN

Solana Vaults - TruFin

Prepared by: HALBORN Last Updated 03/13/2026 Date of Engagement: February 18th, 2026 - February 25th, 2026

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
2	0	0	0	1	1

1. Introduction

`TruFin` engaged Halborn to conduct a security assessment on their `Raydium Vault` program beginning on February 18th, 2026 and ending on February 25th, 2026. The security assessment was scoped to the smart contracts provided in the GitHub repository `solana-vaults`, commit hashes, and further details can be found in the Scope section of this report.

The `Trufin team` is releasing a new version of Raydium Vault Solana program. The program implements a vault that integrates with Raydium's Concentrated Liquidity Market Maker (CLMM) and TruFin's TruSOL liquid staking token. Users deposit SOL, which is split between WSOL and staking for TruSOL, then deployed as liquidity into a Raydium CLMM position. The vault mints proportional share tokens to depositors and manages withdrawals, fee collection, and position management through a set of permissioned and permissionless instructions.

TABLE OF CONTENTS

2. Assessment Summary

Halborn was provided 6 business 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 Program.
- Ensure that smart contract functionality operates as intended.

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

- `Move the vault idle balance calculation to after the Raydium CPI in the withdraw instruction to ensure fair non-treasury fee distribution between consecutive withdrawers`
- `Include unclaimed Raydium CLMM trading fees in the total_assets calculation during deposit share pricing to prevent dilution of existing shareholders`

3. Test Approach And Methodology

Halborn performed a combination of manual review and security testing based on scripts to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Differences analysis using GitLens to have a proper view of the differences between the mentioned commits
- Graphing out functionality and programs logic/connectivity/functions along with state changes

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 (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C: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

REPOSITORY

(a) Repository: solana-vaults-public

(b) Assessed Commit ID: 80cbcc1

(c) Items in scope:

- raydium-vault/programs/raydium-vault/Cargo.toml
- raydium-vault/programs/raydium-vault/src/lib.rs
- raydium-vault/programs/raydium-vault/src/constants.rs
- raydium-vault/programs/raydium-vault/src/error.rs
- raydium-vault/programs/raydium-vault/src/state/events.rs
- raydium-vault/programs/raydium-vault/src/state/math.rs
- raydium-vault/programs/raydium-vault/src/state/mod.rs
- raydium-vault/programs/raydium-vault/src/state/types.rs
- raydium-vault/programs/raydium-vault/src/state/verify.rs
- raydium-vault/programs/raydium-vault/src/instructions/withdraw.rs
- raydium-vault/programs/raydium-vault/src/instructions/staking.rs
- raydium-vault/programs/raydium-vault/src/instructions/setters.rs
- raydium-vault/programs/raydium-vault/src/instructions/mod.rs
- raydium-vault/programs/raydium-vault/src/instructions/initialize.rs
- raydium-vault/programs/raydium-vault/src/instructions/deposit.rs
- raydium-vault/programs/raydium-vault/src/instructions/collect_fees.rs
- raydium-vault/programs/raydium-vault/src/instructions/clmm/close_position.rs
- raydium-vault/programs/raydium-vault/src/instructions/clmm/decrease_liquidity.rs
- raydium-vault/programs/raydium-vault/src/instructions/clmm/increase_liquidity.rs
- raydium-vault/programs/raydium-vault/src/instructions/clmm/mod.rs
- raydium-vault/programs/raydium-vault/src/instructions/clmm/open_position.rs

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

FILE

(a) Submitted File: solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb.zip

(b) Items in scope:

- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/clmm/close_position.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/clmm/decrease_liquidity.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/clmm/increase_liquidity.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/clmm/mod.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/clmm/open_position.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/collect_fees.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/deposit.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/initialize.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/mod.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/setters.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/staking.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/instructions/withdraw.rs

- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/state/events.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/state/math.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/state/mod.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/state/types.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/state/verify.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/constants.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/error.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/src/lib.rs
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/Cargo.toml
- /solana-vaults-public-80cbcc1d8408e430547eebb0040c2358e26de8cb/raydium-vault/programs/raydium-vault/Xargo.toml

REMEDIATION COMMIT ID:

- https://github.com/TruFin-io/solana-vaults-public/pull/3/changes/fef1093dd1c44bea81fa8df5182355decd02452a
- 697bcf9

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	1

INFORMATIONAL
1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
PRE-CPI IDLE BALANCE CALCULATION CAUSES UNFAIR FEE DISTRIBUTION BETWEEN WITHDRAWERS	LOW	SOLVED - 03/09/2026
SHARE PRICING EXCLUDES UNCLAIMED RAYDIUM CLMM FEES LEADING TO DEPOSITOR DILUTION OF EXISTING SHAREHOLDERS	INFORMATIONAL	SOLVED - 03/09/2026

7. FINDINGS & TECH DETAILS

7.1 PRE-CPI IDLE BALANCE CALCULATION CAUSES UNFAIR FEE DISTRIBUTION BETWEEN WITHDRAWERS

// LOW

Description

The `withdraw` instruction computes the withdrawing user's proportional share of idle vault assets (`vault_wsol_redeemed`) **before** invoking the Raydium `decrease_liquidity_v2` CPI, as shown in the code snippet below.

Raydium's `decrease_liquidity_v2` collects **all** accumulated trading fees from the entire position on every invocation, regardless of how much liquidity is removed.

These fees enter the vault ATA as part of the CPI return, but only a percentage (`vault_state.fee / FEE_PRECISION`) is tracked as treasury fees.

The remaining non-treasury portion stays in the vault ATA as untracked balance.Since the user's idle share was already frozen before the CPI, the non-treasury fees are excluded from their payout calculation.

The payout formula `wsol_to_user = wsol_received + vault_wsol_redeemed - wsol_fees_collected` effectively cancels the fees from the Raydium return, leaving the user with only their proportional liquidity plus pre-CPI idle.

The non-treasury fees remain in the ATA and inflate the idle balance for subsequent withdrawers.

programs/raydium-vault/src/instructions/withdraw.rs

```
let wsol_treasury_fee = ctx.accounts.vault_state.treasury_wsol_fees;
...
let pre_vault_wsol = ctx.accounts.wsol_ata.amount;
...
// Idle share calculated BEFORE the Raydium CPI
let vault_wsol_redeemed = U256::from(pre_vault_wsol.saturating_sub(wsol_treasury_fee))
    .checked_mul(shares_redeemed_ratio)
    ...
    .as_u64();
...
// Raydium CPI: collects ALL accumulated fees + removes proportional liquidity
cpi::decrease_liquidity_v2(decrease_liquidity_ctx, liquidity_to_remove, 0, 0)?;
...
let wsol_received = post_vault_wsol.saturating_sub(pre_vault_wsol);
...
let wsol_fees_collected = post_wsol_fees_claimed.saturating_sub(pre_wsol_fees_claimed);
...
// Fees cancel out: wsol_received includes fees, but wsol_fees_collected subtracts them
let mut wsol_to_user = wsol_received
    .saturating_add(vault_wsol_redeemed)
    .saturating_sub(wsol_fees_collected);
...
// Only treasury portion is tracked; non-treasury remains as untracked idle
let wsol_fee_to_treasury = wsol_fees_collected
    .checked_mul(vault_state.fee)
    .and_then(|v| v.checked_div(FEE_PRECISION))
    ...;
vault_state.treasury_wsol_fees = vault_state.treasury_wsol_fees
    .saturating_add(wsol_fee_to_treasury);
```

The first user to withdraw after Raydium trading fees accumulate forfeits their proportional share of non-treasury fees entirely. Subsequent withdrawers receive a disproportionately larger payout because the untracked non-treasury fees inflate their idle balance.

For example, with two equal shareholders (50/50), 100 WSOL in accumulated fees, and a 10% treasury fee:

- User A (first withdrawer) receives 0 of the 90 WSOL non-treasury fees.
- User B (second withdrawer) receives all 90 WSOL of non-treasury fees.
- Fair distribution would be 45 WSOL each — User A loses 45 WSOL.

This creates an incentive to delay withdrawals and wait for other users to trigger fee collection first. The severity scales with the size of accumulated fees relative to total vault assets and the frequency of fee sweeping operations.

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:N/Y:M (3.4)

Recommendation

It is recommended to move the idle balance calculation (`vault_wsol_redeemed` / `vault_trusol_redeemed`) to **after** the Raydium CPI, and include the non-treasury fee portion in the idle before computing the user's proportional share.

The corrected calculation would:
1. Execute the Raydium `decrease_liquidity_v2` CPI first.
2. Compute the treasury fee from the collected fees.
3. Calculate the user's idle share from the post-CPI ATA balance, excluding both old and new treasury fees, but including non-treasury fees.

Remediation Comment

SOLVED: The **TruFin team** solved the issue by implementing the suggested changes.

Remediation Hash

https://github.com/TruFin-io/solana-vaults-public/pull/3/changes/fef1093dd1c44bea81fa8df5182355decd02452a

7.2 SHARE PRICING EXCLUDES UNCLAIMED RAYDIUM CLMM FEES LEADING TO DEPOSITOR DILUTION OF EXISTING SHAREHOLDERS

// INFORMATIONAL

Description

The `deposit` instruction calculates the number of shares to mint for a new depositor using the `calculate_shares_minted` function, which computes `total_assets` as the sum of vault ATA balances (net of treasury fees) and position token amounts derived from `calculate_position_token_amounts`, as shown in the code snippet below.

The `calculate_position_token_amounts` function uses the position's liquidity and tick range to determine the token amounts held in the Raydium CLMM position, but it does not account for unclaimed trading fees accrued on the position.

Raydium CLMM positions accumulate trading fees over time, which are only materialized when `decrease_liquidity_v2` is called. These unclaimed fees are part of the vault's total value but are excluded from the `total_assets` calculation.

programs/raydium-vault/src/instructions/deposit.rs

Copy Code

```
// Vault ATA amounts - treasury fees (treasury fees are held in ATAs but belong to treasur
let wsol_for_shareholders =
    ctx.accounts.wsol_ata.amount.saturating_sub(ctx.accounts.vault_state.treasury_wsol_fee
let trusol_for_shareholders = vault_trusol_balance;

let shares_to_mint = calculate_shares_minted(
    amount,
    wsol_for_shareholders,
    trusol_for_shareholders,
    stake_pool_account.total_lamports,
    stake_pool_account.pool_token_supply,
    pool_held_wsol_amount,       // from calculate_position_token_amounts (liquidity only)
    pool_held_trusol_amount,     // from calculate_position_token_amounts (liquidity only)
    ctx.accounts.share_mint.supply,
)?;
```

```
...
fn calculate_shares_minted(...) -> Result<u64> {
    ...
    // total_assets = vault idle + position liquidity value (fees NOT included)
    let total_sol_amount = net_vault_wsol.checked_add(pool_wsol_amount)...;
    let total_trusol_amount = net_vault_trusol.checked_add(pool_trusol_amount)...;
    ...
    let total_assets = total_sol_amount.checked_add(total_trusol_amount_in_sol)...;

    // shares = amount * total_shares / total_assets
    let shares_to_mint = amount_u256
        .checked_mul(total_shares_u256)
        .and_then(|x| x.checked_div(total_assets_u256))...;
    ...
}

fn calculate_position_token_amounts(...) -> Result<(u64, u64)> {
    // Only computes token amounts from liquidity and tick range
    // Does NOT include unclaimed trading fees (token_fees_owed_0/1)
    let (amount_0, amount_1) = liquidity_math::get_delta_amounts_signed(
        tick_current, sqrt_price_current_x64, tick_lower, tick_upper, liquidity_delta,
    )...;
    Ok((amount_0, amount_1))
}
```

Since `total_assets` is undervalued, the share price denominator is smaller than it should be, causing new depositors to receive more shares than their proportional contribution warrants. These excess shares dilute existing shareholders.

For example, if the vault holds 1000 SOL in liquidity plus 100 SOL in unclaimed fees, `total_assets` equals 1000 instead of 1100. A depositor of 100 SOL receives shares representing ~9.1% of the vault instead of the correct ~8.3%, diluting existing holders by the difference.

The magnitude depends on the size of unclaimed fees relative to total vault assets and the frequency of operations that trigger fee collection (`decrease_liquidity_v2`).

BVSS

AO:A/AC:L/AX:M/R:N/S:U/C:N/A:N/I:N/D:N/Y:L (1.7)

Recommendation

It is recommended to include the unclaimed Raydium CLMM fees `personal_position.token_fees_owed_0` and `token_fees_owed_1`) in the `calculate_position_token_amounts` function, so that `total_assets` reflects the full value of the vault including accrued but uncollected trading fees.

Remediation Comment

SOLVED: The **TruFin team** solved the issue by implementing the suggested changes. The `calculate_position_token_amounts` function now adds `personal_position.token_fees_owed_0` and `token_fees_owed_1` to the liquidity-derived token amounts, ensuring unclaimed Raydium trading fees are included in `total_assets` for share pricing.

Remediation Hash

https://github.com/TruFin-io/solana-vaults-public/commit/697bcf981345f667fc43b51be4cbfd102ee7f0be

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated security scanners to assist with the 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 reviewers 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.

```
No vulnerabilities were found.
```

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.