



Upshift Solana ERC4626

Security Assessment

September 1st, 2025 — Prepared by OtterSec

Ajay Shankar Kunapareddy

d1r3wolf@osec.io

Xiang Yin

soreatu@osec.io

Robert Chen

r@osec.io

Table of Contents

Executive Summary	2
Overview	2
Key Findings	2
Scope	3
Findings	4
Vulnerabilities	5
OS-SSE-ADV-00 Share Inflation via Token Donation	6
OS-SSE-ADV-01 Withdrawal Fee Truncation	7
OS-SSE-ADV-02 Bypassing AUM Update Constraints	8
OS-SSE-ADV-03 Improper Handling of Transfer Fees	9
OS-SSE-ADV-04 Incorrect Vault State Size Calculation	10
General Findings	11
OS-SSE-SUG-00 Code Refactoring	12
OS-SSE-SUG-01 Code Maturity	13
Appendices	
Vulnerability Rating Scale	14
Procedure	15

01 — Executive Summary

Overview

Defiborg engaged OtterSec to assess the `solana-erc4626` program. This assessment was conducted between August 20th and August 29th, 2025. For more information on our auditing methodology, refer to [Appendix B](#).

Key Findings

We produced 7 findings throughout this audit engagement.

In particular, the vault is vulnerable to the ERC-4626 inflation/donation attack, allowing attackers to skew share calculations with unsolicited token transfers and drain victim deposits ([OS-SSE-ADV-00](#)).

Additionally, small withdrawals may bypass the vault's withdrawal fee due to integer truncation, allowing attackers to avoid fees; rounding up prevents this ([OS-SSE-ADV-01](#)). Furthermore, strict inequality with small deployed AUM values may block valid updates due to integer truncation ([OS-SSE-ADV-02](#)).

We also suggested the need to adhere to coding best practices ([OS-SSE-SUG-01](#)), and advised to incorporate certain refactors to improve functionality and mitigate potential security issues ([OS-SSE-SUG-00](#)).

02 — Scope

The source code was delivered to us in a Git repository at <https://github.com/defiborg/solana-erc4626>. This audit was performed against commit [592be1c](#).

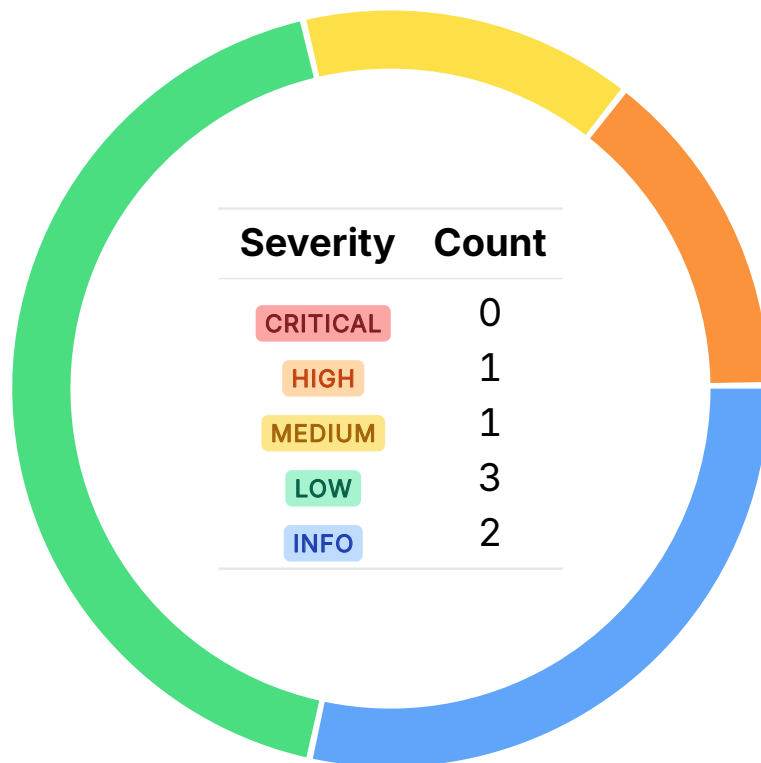
A brief description of the program is as follows:

Name	Description
solana-erc4626	A share-based vault allows users to deposit SPL tokens and receive shares, while an operator can withdraw and redeposit funds for external strategies (e.g., yield farming). The admin manages fees and operator roles, and can pause or unpause the vault in emergencies.

03 — Findings

Overall, we reported 7 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in [Appendix A](#).

ID	Severity	Status	Description
OS-SSE-ADV-00	HIGH	RESOLVED ✓	The vault's share conversion can be manipulated via unsolicited donations (assets or shares), inflating the exchange rate and allowing attackers to mint an outsized number of shares for a small deposit or to redeem more assets than contributed.
OS-SSE-ADV-01	MEDIUM	RESOLVED ✓	Small withdrawals may bypass the vault's withdrawal fee due to integer truncation, allowing attackers to avoid fees; rounding up prevents this.
OS-SSE-ADV-02	LOW	RESOLVED ✓	Strict inequality with small <code>deployed_aum</code> values may block valid updates due to integer truncation.
OS-SSE-ADV-03	LOW	RESOLVED ✓	If the vault accepts a <code>deposit_mint</code> with a <code>transfer_fee</code> Token22 extension, deposits may result in a shortfall of received tokens, while users are still credited with full shares.
OS-SSE-ADV-04	LOW	RESOLVED ✓	The <code>VAULT_STATE_SIZE</code> is overestimated and placed in the wrong file.

Share Inflation via Token Donation HIGH

OS-SSE-ADV-00

Description

This issue embodies the well-known ERC-4626 *inflation/donation attack*: an attacker first deposits a minimal amount to mint an initial share, then donates a sizable amount of assets to the vault. This artificially inflates the vault's asset-to-share ratio, distorting the exchange rate. When a subsequent user deposits, rounding may yield zero shares for their deposit, effectively transferring value to the attacker.

Additionally, the vault currently utilizes the raw `vault_token_ata.amount / vault_deposit_ata.amount` to calculate total assets in `Deposit / Redeem`. Such reliance on external balances permits an attacker to further manipulate share calculations simply by transferring assets directly into the vault's token account. This can result in inflated vault balances that cause share-minting to round down to zero for rightful depositors.

```
>_ august-vault/src/instructions/deposit.rs RUST  
  
pub fn handler(ctx: Context<Deposit>, amount: u64) -> Result<> {  
    [...]  
    let supply = ctx.accounts.share_mint.supply;  
    let total_assets =  
        ctx.accounts.vault_token_ata.amount + ctx.accounts.vault_state.deployed_aum;  
    [...]  
}
```

Remediation

1. Enforce slippage-style invariants at the boundary: `deposit` must ensure `shares > 0`; `redeem` must ensure `assets > 0`. These checks prevent zero-share mints / zero-asset redemptions caused by donation-skewed pricing.
2. Introduce *virtual shares/assets* (a fixed non-zero offset added to both the share and asset sides in conversion formulas) to stabilize the exchange rate near an empty vault and remove the attacker's ability to set an arbitrary initial price.
3. Introduce explicit accounting in `VaultState` to track net deposits and withdrawals. Base pricing and share calculations solely on this internal state rather than the raw token account balance, ensuring that direct token transfers into the vault ATA cannot manipulate asset valuation.

Patch

Resolved in [PR#3](#), [PR#4](#), and [PR#5](#).

Withdrawal Fee Truncation MEDIUM

OS-SSE-ADV-01

Description

The withdrawal fee in `Redeem` is calculated via integer division:

$\text{fees} = \text{assets} * \text{withdrawal_fee} / \text{FEE_RATE_DENOMINATOR_VALUE}$. For small withdrawals, this may truncate to zero if $\text{assets} * \text{withdrawal_fee} < \text{FEE_RATE_DENOMINATOR_VALUE}$, effectively bypassing the fee. This reduces protocol revenue and undermines the intended fee mechanism.

```
>_ august-vault/src/instructions/redeem.rs
```

RUST

```
pub fn handler(ctx: Context<Redeem>, shares: u64) -> Result<()> {  
    [...]  
    let supply = ctx.accounts.share_mint.supply;  
    let total_assets = ctx.accounts.vault_deposit_ata.amount +  
        ↪ ctx.accounts.vault_state.deployed_aum;  
    let assets = (shares as u128 * total_assets as u128 / supply as u128) as u64;  
    let fees = (assets as u128 * ctx.accounts.vault_state.withdrawal_fee as u128  
        / FEE_RATE_DENOMINATOR_VALUE as u128) as u64;  
    let final_amount = assets.checked_sub(fees).unwrap();  
    [...]  
}
```

Remediation

Round the fee up to favor the protocol, ensuring even small withdrawals contribute at least the minimum fee.

Patch

Resolved in [PR#3](#).

Bypassing AUM Update Constraints LOW

OS-SSE-ADV-02

Description

The strict inequality in the `require` statements excludes equality, which creates problems if `deployed_aum < 20`, where integer division truncation may result in no valid increases or decreases of `new_aum`, effectively blocking updates.

```
>_ august-vault/src/instructions/operator_update_aum.rs
```

RUST

```
pub fn handler(ctx: Context<OperatorUpdateAum>, new_aum: u64) -> Result<> {  
    let state = &mut ctx.accounts.vault_state;  
    require!(new_aum > 90 * state.deployed_aum / 100, ErrorCode::AumDecreaseTooBig);  
    require!(new_aum < 110 * state.deployed_aum / 100, ErrorCode::AumIncreaseTooBig);  
  
    state.deployed_aum = new_aum;  
    Ok(())  
}
```

Remediation

Allow equality in the two `require` statements.

Patch

Resolved in [PR#3](#).

Improper Handling of Transfer Fees LOW

OS-SSE-ADV-03

Description

The vault currently assumes that the `deposit_mint` transfers the full amount requested in the deposit instruction. However, if the mint includes a `transfer_fee` Token22 extension, the actual tokens received by the vault token account will be less than the requested amount. This discrepancy allows users to be over-credited with shares, effectively diluting the value of existing depositors.

Remediation

Enforce a whitelist of permitted `Token22` extensions for `deposit_mint` during vault initialization, or explicitly handle `transfer_fee` logic when processing deposits to ensure that share calculations are based on net tokens received.

Patch

The developers acknowledged the issue by mentioning this is an anti miss-click feature and the operator role can do whatever it wants with the `deployed_aum`.

Incorrect Vault State Size Calculation LOW

OS-SSE-ADV-04

Description

The current `VAULT_STATE_SIZE` constant is incorrectly calculated, utilizing 64 for a `u64` field and resulting in an inflated size (234 instead of 182). Also, keeping the size definition in `initialize` instruction is brittle, since any change to `VaultState` fields risks mismatched values. A better approach is to define the size alongside the `VaultState` structure in `vault`.

```
>_ august - vault/src/instructions/initialize.rs
```

RUST

```
const VAULT_STATE_SIZE: usize = 8 + 32 + 32 + 32 + 32 + 32 + 1 + 64 + 1;
```

Remediation

Ensure proper size allocation and move the size definition to the `vault.rs` file.

Patch

Resolved in [PR#3](#).

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-SSE-SUG-00	Recommendation for refactoring the logic to improve functionality and mitigate potential security issues.
OS-SSE-SUG-01	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.

Code Refactoring

OS-SSE-SUG-00

Description

1. Directly casting a `u128` to `u64` via `as u64` may silently truncate values if they exceed `u64::MAX`, resulting in incorrect results. Utilize `u64::try_from` to ensure the conversion is checked, returning an error in case of an overflow.
2. Utilize a single unified context for all vault update instructions (`SetWithdrawalFee`, `SetOperator`, `SetFeeRecipient`, `Pause`, `Unpause`) instead of separate ones. This reduces code duplication and ensures consistent access control.
3. Ensure the vault's `admin` explicitly signs the `initialize` instruction instead of just passing it as a parameter. This prevents accidental misconfiguration where the wrong `admin` key is assigned.

Remediation

Incorporate the above refactors.

Patch

Resolved in [PR#3](#).

Code Maturity

OS-SSE-SUG-01

Description

1. `VaultState::seed` should be renamed to `seeds` since it returns multiple PDA seeds, not a single one. This improves clarity and readability.

```
>_ august-vault/src/state/vault.rs RUST  
  
/// Get the seed for the vault state PDA  
pub fn seed(&self) -> [&[u8]; 2] {  
    [b"vault_state", &self.pda_bump]  
}
```

2. Define the seed string `vault_state` as a constant similar to how `NOMINATED_ADMIN_PDA_SEED` is defined, for better clarity and maintainability.
3. It is not required to mark the `operator` and `admin` accounts as mutable (`mut`) in all operator-related and admin-related instructions, respectively, as they are only read from.

Remediation

Implement the above-mentioned suggestions.

Patch

Resolved in [PR#3](#).

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the [General Findings](#).

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
 - Improperly designed economic incentives leading to loss of funds.
-

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
 - Exploitation involving high capital requirement with respect to payout.
-

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
 - Forced exceptions in the normal user flow.
-

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.
-

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
 - Improved input validation.
-

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.