# Solomon Labs - Stake program

# Audit Report

Version 1.2

*Zigtur*

July 20, 2024

# Solomon Labs - Stake program - Audit Report

Zigtur

July 20, 2024

Prepared by: Zigtur

## Table of Contents

# Introduction

## Disclaimer

A smart contract security review cannot guarantee the complete absence of vulnerabilities. This effort, bound by time, resources, and expertise, aims to identify as many security issues as possible. However, there is no assurance of 100% security post-review, nor is there a guarantee that the review will uncover all potential problems in the smart contracts. It is highly recommended to conduct subsequent security reviews, implement bug bounty programs, and perform on-chain monitoring.

## About Zigtur

**Zigtur** is an independent blockchain security researcher dedicated to enhancing the security of the blockchain ecosystem. With a history of identifying numerous security vulnerabilities across various protocols in public audit contests and private audits, **Zigtur** strives to contribute to the safety and reliability of blockchain projects through meticulous security research and reviews. Explore previous work here or reach out on X @zigtur.

## About Solomon Labs

Solomon Labs is building a stablecoin-like yield protocol powered by perpetual funding payments.

# Security Assessment Summary

***Review commit hash -*** 2819c01d0fc59d14b3c495f8d64254ec35adea93

***Fixes review commit hash -*** 8ddbdee6b4186b23715dcd7b9f627040e53121ce

## Deployment chains

- Solana

## Scope

The following files are in scope of the review:

- stake/src/lib.rs
- stake/src/context.rs

### Risk Classification

| | **Impact:** High | **Impact:** Medium | **Impact:** Low |
|---|---|---|---|
| **Likelihood:** High | High | High | Medium |
| **Likelihood:** Medium | High | Medium | Low |
| **Likelihood:** Low | Medium | Low | Low |

*Note: Informational findings may not raise security concerns but are notable and require developers' attention.*

### Issues Count

A total of **15 issues** have been identified and can be classified as:

- **7 MEDIUM**
- **3 LOW**
- **5 INFO**

The mitigation review shows that 10 issues were fixed and 1 is partially fixed ( `LOW-02` ).

4 issues are acknowledged by Solomon Labs ( `MEDIUM-01` , `MEDIUM-07` , `INFO-01` , `INFO-03` ).

## Issues

### MEDIUM-01 - Attacker can censor vault initialization

#### Description

Scope:

- context.rs#L10
- context.rs#L13

The `initialize_vault_stake` instruction allows to initialize a vault state PDA account permissionlessly. This PDA account is derived from the `VAULT_STATE_SEED` and **a `salt` controlled by the user**.

As there is no access control and the `init` Anchor constraints is used, an attacker can front-run a legit `initialize_vault_stake` transaction with the same `salt` value.

This will make the legit transaction to fail because a PDA derived from this `salt` already exist.

#### Impact

Attacker can censor vault initialization by front-running.

#### Code snippet

The issue comes from the following code:

```
#[derive(Accounts)]
#[instruction(admin: Pubkey, salt: [u8; 8], cooldown: u32)]
pub struct InitializeVaultState<'info> {
    /// The vault state for this deposit token and admin
    #[account(
        init,               // @POC: This will fail if the PDA already exist
        payer = caller,
        space = 8 + (3 * 32) + (3 * 8) + (3 * 4) + 1 + (32 * 20),
        seeds = [VAULT_STATE_SEED, salt.as_ref()], // @POC: The PDA is derived
        ↪  from `salt` controlled by the user
        bump
    )]
    pub vault_state: Box<Account<'info, VaultState>>,
```

**Recommendation**

According to documentation, vault initialization should be done by a trusted party. Considering this, the most efficient fix is to set an access control check on the `initialize_vault_stake` instruction. This can be done by creating a `global_parameters` account which holds the initialization administrator.

If vault initialization should be allowed to anyone, the issue can be fixed by creating a `global_parameters` account which holds an incremental counter used as `salt`.

*Note: These fixes are heavy and a patch couldn't be provided during the audit.*

**Resolution**

SolomonLabs Team: Acknowledged. The team states:

> "Because there's a huge amount of available salt combinations we wont bother locking it down."

Zigtur: Acknowledged. An attacker would not have incentive to censor vault initialization.

### MEDIUM-02 - `initialize_program_accounts` lacks access control

#### Description

Scope:

- context.rs#L66
- lib.rs#L79-L121

The `initialize_program_accounts` instruction allows initializing the staking token and setting its metadata for a specific `vault_state` PDA account. This `vault_state` must have been initialized through the `initialize_vault_stake` instruction.

However, the `initialize_program_accounts` instruction doesn't ensure that the caller is the admin set in the `vault_state` PDA account.

#### Impact

Attacker can set metadata parameters for a created vault before the admin sets it. He will gain authority over the metadata.

#### Code snippet

The `InitializeProgramAccounts` doesn't ensure that `vault_state.admin` is the current caller:

```
#[derive(Accounts)]
#[instruction(salt: [u8; 8])]
pub struct InitializeProgramAccounts<'info> {
    /// The vault state for this deposit token and admin
    #[account(
        seeds = [VAULT_STATE_SEED, salt.as_ref()],
        bump
    )]
    pub vault_state: Box<Account<'info, VaultState>>,


    /// ...


    #[account(mut)]
    pub caller: Signer<'info>, // @POC: `caller` must be `vault_state.admin`
```

Moreover, the `initialize_program_accounts` instruction doesn't check this neither.

**Recommendation**

`initialize_program_accounts` instruction must check that the current caller is the admin set in `vault_state`. This can be done by adding a constraint in the `InitializeProgramAccounts` structure or by adding a check in the `initialize_program_accounts` function logic.

A patch is given in Appendix to fix this issue. It adds a check in `initialize_program_accounts` function to comply with the rest of the codebase.

**Resolution**

SolomonLabs Team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

### MEDIUM-03 - `deposit_token` lacks of constraints in `InitializeProgramAccounts` structure

#### Description

Scope:

- context.rs#L64

The `InitializeProgramAccounts` structure expects a `deposit_token` account. This account must be the same than `vault_state.deposit_token`.

However, this constraint is not checked.

#### Impact

Combined with MEDIUM-02, this could lead the `staking_token` to be created with an incorrect token decimals.

#### Code snippet

The `InitializeProgramAccounts` doesn't ensure that `vault_state.deposit_token` is the current `deposit_token`:

```
#[derive(Accounts)]
#[instruction(salt: [u8; 8])]
pub struct InitializeProgramAccounts<'info> {
    /// The vault state for this deposit token and admin
    #[account(
        seeds = [VAULT_STATE_SEED, salt.as_ref()],
        bump
    )]
    pub vault_state: Box<Account<'info, VaultState>>,


    /// ...


    #[account(mut)]
    pub deposit_token: Box<Account<'info, Mint>>, // @POC: `deposit_token` must be
    ↪  `vault_state.deposit_token`
```

**Recommendation**

`initialize_program_accounts` instruction must check that the current `deposit_token` is the same than `vault_state.deposit_token`. This can be done by adding a constraint in the `InitializeProgramAccounts` structure or by adding a check in the `initialize_program_accounts` function logic.

A patch is given in Appendix to fix this issue. It adds a constraint on `InitializeProgramAccounts` structure.

**Resolution**

SolomonLabs Team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

### MEDIUM-04 - Admin can't remove a user from blacklist

#### Description

Scope:

- lib.rs#L167

The `blacklist` instruction allows the administrator of a given `vault_state` account to blacklist a user for this specific vault.

However, there is no mechanism to unblacklist the user.

#### Impact

Any blacklisted user will never be unblacklistable.

#### Code snippet

The `blacklist` instruction always sets `blacklisted = true`:

```
    pub fn blacklist(ctx: Context<Blacklist>, _salt: [u8; 8], user: Pubkey) ->
    ↪  Result<()> {
        // ...

        ctx.accounts.blacklisted.user = user;
        ctx.accounts.blacklisted.blacklisted = true; // @POC: Only `blacklisted =
        ↪  true` is possible

        // ...
    }
```

#### Recommendation

The `blacklist` instruction should be modified such that the administrator uses a boolean to indicate if the user should be blacklisted or not.

A patch is given in Appendix to fix this issue. It adds boolean parameter to the `blacklist` instruction.

#### Resolution

SolomonLabs Team: Fixed. `remove_from_blacklist` instruction has been added.

Zigtur: Fix reviewed and approved.

### MEDIUM-05 - Reward vesting can be bypassed by a rewarder

#### Description

- lib.rs#L315-L317

The `reward` instruction can be executed by whitelisted rewarders. It sets the current time as the last distribution time and set the vesting amount to the rewarded amount. These parameters are set to then be used in reward vesting amount calculation.

However, `reward` instruction will overwrite these current parameters without checking if a rewards vesting period is still on-going. This allows a rewarder to bypass the rewards vesting period.

Moreover, as there could be multiple rewarders, a rewarder can delete the rewards vesting period of another rewarder by calling `reward` instruction with a small amount.

*Note: Even if there is no malicious rewarder, this issue may still be triggered if multiple distributions happen in a small timeframe.*

#### Impact

A reward vesting will be cancelled before its end.

#### Code snippet

The `reward` instruction doesn't ensure that there is no vesting period on-going before overwriting the rewards parameters.

```
    pub fn reward(ctx: Context<Reward>, amt: u64, _salt: [u8; 8]) -> Result<()> {
        // ...

        ctx.accounts.vault_state.last_distribution_time = time; // @POC: Last
        ↪  distribution time is not checked before being set
        ctx.accounts.vault_state.total_assets += amt;
        ctx.accounts.vault_state.vesting_amount = amt;
```

**Recommendation**

`reward` instruction must ensure that there is no vesting period on-going. This can be done with the following check.

```
    let time = Clock::get()?.unix_timestamp as u32;
    let time_passed = (time - ctx.accounts.vault_state.last_distribution_time) as
    ↪  u64;
    let vesting_period = ctx.accounts.vault_state.vesting_period as u64;

    if time_passed < vesting_period {
        return Err(StakeError::RewardVestingOngoing.into());
    }
```

A patch is given in Appendix to fix this issue.

**Resolution**

SolomonLabs Team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

### MEDIUM-06 - Unsafe casting to `u64` in `convert_to_shares` and `convert_to_assets`

#### Description

- lib.rs#L355
- lib.rs#L366

The `convert_to_shares` and `convert_to_assets` functions are doing calculations with `u128` type. The result of the calculations is then casted into an `u64` type.

However, this casting is not safe.

*Note: This issue is not likely to happen, but it would have high impact on the protocol.*

#### Code snippet

The casting from `u128` to `u64` is done with `as` keyword.. This casting is not safe.

```
355     let x = (assets as u128 * total_supply as u128 / total_assets as u128) as u64;
```

#### Recommendation

Consider replacing the `as u64` casting with a casting that reverts if the value is greater than `u64::MAX`. For example, `try_into().unwrap()` will do this check.

```
355     let x: u64 = (assets as u128 * total_supply as u128 / total_assets as
        ↪  u128).try_into().unwrap();
```

A patch is given in Appendix to fix this issue.

#### Resolution

SolomonLabs Team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

### MEDIUM-07 - Rewarding during low liquidity periods will inflate share price

#### Description

- lib.rs#L355
- lib.rs#L316

When a user calls the `stake` instruction, the amount of shares minted for the input amount is calculated in `convert_to_shares`. This amount of shares is defined through a pricing calculation as follow:

$$Shares = (Assets * TotalShares)/TotalAssets$$

When the total number of assets increases without an increase of other elements, the given amount of shares will decrease. This happens when rewards are distributed through the `reward` instruction.

The issue lies in the fact that when there is low liquidity in the vault (total assets and total shares are low), the reward distribution will have a tremendous impact on the share price (increase of total assets). This issue is even more severe when combined with `MEDIUM-05` which bypass the vesting period.

*Note: Low liquidity periods will happen, especially right after the vault was created and users didn't stake in it yet.*

#### Impact

Share price will increase tremendously due to rewards during low liquidity periods.

#### Scenarios

*Note: The following scenarios do not take into considerations the token decimals representation.*

Scenario 1 - Rewards distribution during normal liquidity period:

- `total_assets` = `100_000`
- `total_shares` = `100_000`
- share price before rewards = `100_000 / 100_000 = 1` => 1 token will give 1 share
- `rewards` = `1_000`
- share price after rewards = `100_000 / (100_000 + 1_000) = 0.990_099` => 1 token will give 0.99 share

Scenario 2 - Satisfactory liquidity during low liquidity period:

- `total_assets` = `5_000`
- `total_shares` = `5_000`
- share price before rewards = `5_000 / 5_000 = 1` => 1 token will give 1 share
- `rewards` = `1_000`

- share price after rewards = `5_000 / (5_000 + 1_000) = 0.833_333` => 1 token will give 0.833 share

**Recommendation**

Rewards should not be distributed during low liquidity periods. Because rewarders are trusted, fixing this issue may not be required but is still recommended.

A way to mitigate the issue is to calculate the share price impact of a reward distribution and deny it if the impact is too high (more than 10% for example).

**Resolution**

SolomonLabs Team: Acknowledged. The team states:

> "In regards to share inflation - think in general this should be ok, we will also probably have the initial vesting time set to longer i.e. a week, so this is likely to help mitigate it as well."

Zigtur: Acknowledged.

### LOW-01 - All events lack a parameter to identify the vault

#### Description

Scope:

- lib.rs#L415-L475

The `stake` program handles multiple vaults. Each vault PDA account is derived from a given `salt` value.

However, events emitted by the `stake` program do not emit informations to identify which vault is affected.

#### Code snippet

For example, the `transfer_admin` instruction is given:

```
pub fn transfer_admin(ctx: Context<TransferAdmin>, new_admin: Pubkey, _salt:
↪  [u8; 8]) -> Result<()> {
    if ctx.accounts.caller.key() != ctx.accounts.vault_state.admin {
        return Err(StakeError::NotAdmin.into());
    }

    let vault_state = &mut ctx.accounts.vault_state;
    vault_state.admin = new_admin;

    emit!(AdminTransferEvent{
        old_admin: ctx.accounts.caller.key(), // @POC: Old admin
        new_admin: new_admin,                 // @POC: New admin
    });

    Ok(())
}
```

As we can see, when the admin role is transfered, the emitted event does not allow identifying for which vault the admin transfer has been executed.

#### Recommendation

Every event should emit an information to identify which vault is affected. This information can be the vault PDA account address or the `salt` used to derive this address.

A patch is given in Appendix to fix this issue. It adds the `salt` value to every emitted events.

**Resolution**

SolomonLabs Team: Fixed. Provided patch applied.

Zigtur: Fix reviewed and approved.

### LOW-02 - Users may have to wait more than expected after a cooldown decrease

#### Description

- lib.rs#L406-L408

The `get_available_assets` function iterates through an `unstake_queue` and adds the cooled-down assets to the user's available assets. This iteration stops when one unstake action does not satisfy the cooldown period.

In case of a cooldown decrease, a user may end up with valid unstake actions locked because an older unstake action does not meet the old cooldown period.

#### Recommendation

`get_available_assets` should be reviewed to handle this edge case. Fixing this issue may require heavy modifications.

If the issue is not fixed, it should at least be documented so users are informed.

#### Resolution

SolomonLabs Team: Fixed. `refresh_cooldowns` instruction has been added.

Zigtur: Fix reviewed and approved. The new `refresh_cooldowns` instruction partially fixes the issue, it reduces an old cooldown period in specific cases.

### LOW-03 - `get_unvested` may panic due to division by zero

#### Description

- lib.rs#L380

The `get_unvested` function may revert when `vesting_period = 0` due to a division by zero.

This will happen when a rewards distribution was made in the same block than another transaction (so that `time_passed > vesting_period == false`).

#### Code snippet

The following code is taken from `get_unvested`.

```
    if time_passed > vesting_period {
        return Ok(0); // @POC: this will not be executed when `time_passed ==
        ↪  vesting_period == 0`
    }

    let amt = ((vesting_period - time_passed) * self.vesting_amount) /
    ↪  vesting_period; // @POC: This will revert due to division by zero.
```

#### Recommendation

The `get_unvested` function should return zero when `vesting_period == 0`.

A patch is given in Appendix to fix this issue. It returns zero when `vesting_period == 0`.

#### Resolution

SolomonLabs Team: Fixed. The team states:

> "`set_vesting_period` was modified to not allow vesting periods of `0`, so the division by `0` should not ever occur. If we need to the lowest we will set it is to a value of `1`".

Zigtur: Fix reviewed and approved.

### INFO-01 - Adding a rewarder doesn't check for the size of rewarders

#### Description

- lib.rs#L177-L196

The `add_rewarder` instruction allows the administrator to add a rewarder public key to the `vault_state.rewarders` field. This field is a vector of 20 elements according to the space allocated.

However, there is no check to ensure that the number of rewarders is actually lower than or equal to 20.

#### Recommendation

Consider adding a length check on `vault_state.rewarders` in the `add_rewarder` logic.

#### Resolution

SolomonLabs Team: Acknowledged.

Zigtur: Acknowledged.

### INFO-02 - Vault initialization does not emit event

#### Description

Scope:

- lib.rs#L59-L121

The `initialize_vault_state` and `initialize_program_accounts` instructions do not emit any event.

#### Recommendation

Consider emitting an event to be able to monitor new vault initialization.

#### Resolution

SolomonLabs Team: Fixed, `initialize_vault_state` now emits a `NewVaultEvent` event.

Zigtur: Fix reviewed and approved.

### INFO-03 - `blacklist` can be more efficient

#### Description

- lib.rs#L35-L39
- context.rs#L113

The `Blacklisted` structure an address field which will be filled with the user's address.

However, as this structure is used as a PDA account **derived from the user's address**, writing the user's address in this structure is not necessary.

#### Recommendation

Consider removing the `user` Pubkey field in the `Blacklisted` structure.

#### Resolution

SolomonLabs Team: Acknowledged. As PDA derivation is a one way process, keeping track of the user's address is more convenient.

Zigtur: Acknowledged.

### INFO-04 - Incorrect comments indicate ATA when it is not

#### Description

- context.rs#L45
- context.rs#L102
- context.rs#L206
- context.rs#L306

In multiple comments, the term "ATA" is used.

However, it is incorrectly used. Simple "Token Accounts" are used in the current codebase and not "Associated Token Accounts".

*Note: An ATA can be initialized by anyone while a Token Account can only be initialized by the owner. This difference can have security impacts.*

#### Recommendation

Consider fixing the comments to indicate "Token Account" instead of "ATA".

#### Resolution

SolomonLabs Team: Fixed.

Zigtur: Fix reviewed and approved.

### INFO-05 - `vault_state` space is incorrect

#### Description

- context.rs#L12
- lib.rs#L41-L53

The `vault_state` PDA account is initialized with the following space.

```
    space = 8 + (3 * 32) + (3 * 8) + (3 * 4) + 1 + (32 * 20),
```

However, it uses the following `VaultState` structure.

```
#[account]
pub struct VaultState {          // Anchor discriminator = 8
    pub admin: Pubkey,           // 32
    pub deposit_token: Pubkey,   // 32
    pub vesting_amount: u64,     // 8
    pub total_assets: u64,       // 8
    pub min_shares: u64,         // 8
    pub last_distribution_time: u32,    // 4
    pub cooldown: u32,           // 4
    pub vesting_period: u32,     // 4
    pub bump: u8,                // 1
    pub rewarders: Vec<Pubkey>, // N/A
}
```

As we can see, space indicates `3 * 32` while it should be `2 * 32` according to the structure.

#### Recommendation

Replace `3 * 32` with `2 * 32`.

*Note: This issue doesn't have security impact, only performance impact.*

#### Resolution

SolomonLabs Team: Fixed.

Zigtur: Fix reviewed and approved.