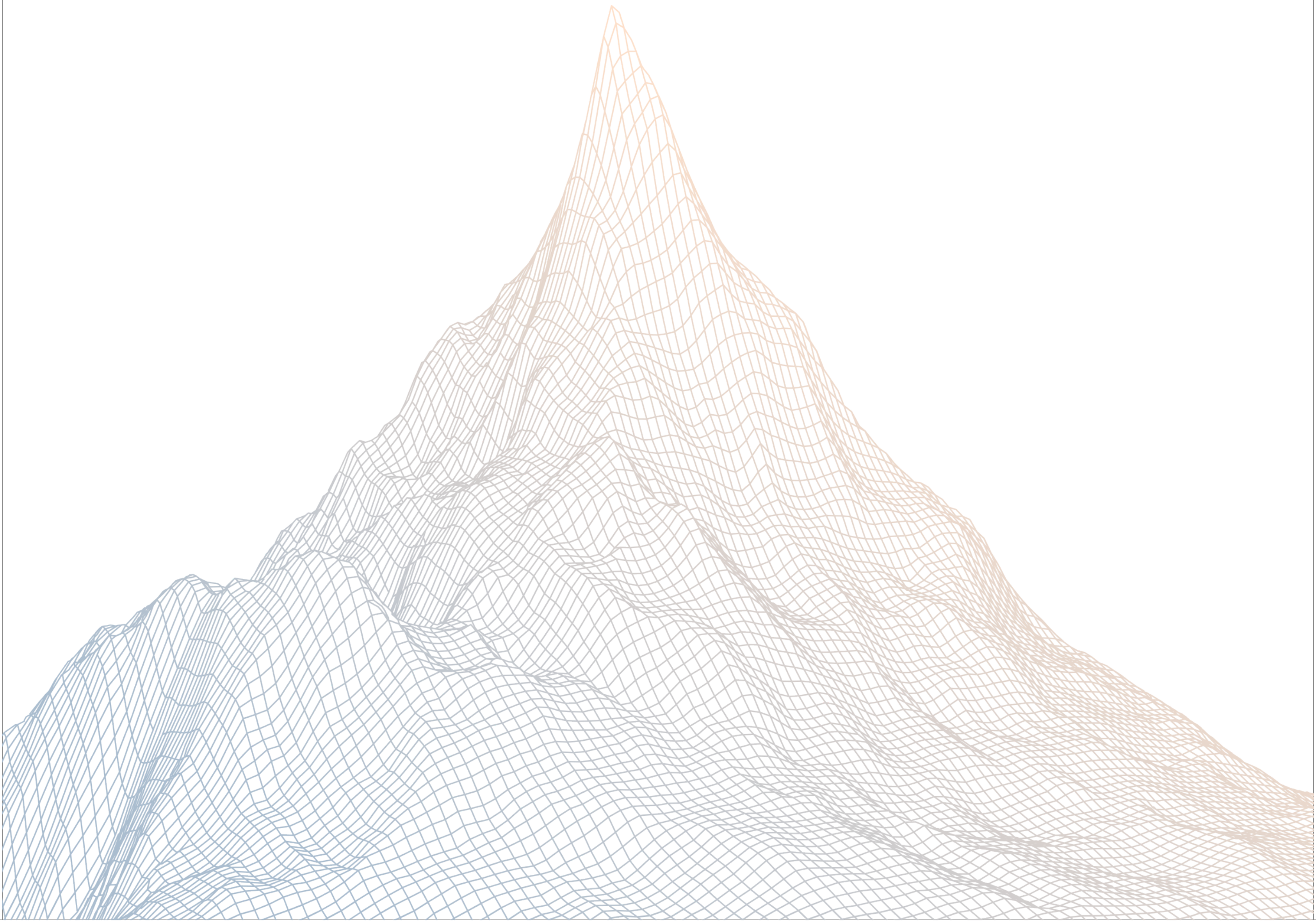
Zenith

Legion

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: June 25th to June 27th, 2025

AUDITED BY: 0xhuy
Mario Poneder

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About Legion

The goal of Legion is to create a network where anyone can freely chat and socialize without compromising their privacy, using the hashgraph consensus.

2.2 Scope

The engagement involved a review of the following targets:

Target	solana-contracts
Repository	https://github.com/Legion-Team/solana-contracts
Commit Hash	`cc553a368398269a1cb00a3342150ccf12eb2928`
Files	`diff up to cc553a368398269a1cb00a3342150ccf12eb2928`

2.3 Audit Timeline

June 25, 2025	Audit start
June 27, 2025	Audit end
July 29, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	3
Low Risk	2
Informational	1
Total Issues	**7**

3 Findings Summary

ID	Description	Status
H-1	Users can bypass vesting and claim 100% of tokens immediately after program upgrade due to uninitialized token release options	Resolved
M-1	Fee calculation based on total_amount increases absolute fee values	Resolved
M-2	Zero duration vesting prevents token claims	Resolved
M-3	Incorrect TOKEN_SALE_EVENT_ACCOUNT_SPACE calculation breaks backward compatibility	Resolved
L-1	Struct space calculation inconsistencies	Resolved
L-2	Missing validation for refund period parameter	Resolved
I-1	SDK function getDepositTokensToVaultTransaction uses depositAmount instead of totalAmount	Resolved

4 Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Users can bypass vesting and claim 100% of tokens immediately after program upgrade due to uninitialized token release options

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- instructions/claim_tokens_from_vault.rs#L80-L91
- state/vesting_options.rs#L31-L58

Description:

After the program upgrade, the newly created `token_release_options` member of the existing `TokenSaleEvent` account will contain default values where both `token_percentage_release_on_tge` and `token_vesting_start_time` are set to 0. This creates a vulnerability that allows users to bypass the intended vesting mechanism and claim their entire token allocation immediately.

The root cause lies in the `calculate_and_validate_claimable_amount()` function in `VestingOptions`. When `token_release_options.token_percentage_release_on_tge` and `token_release_options.token_vesting_start_time` are 0, the elapsed time calculation becomes `current_time.saturating_sub(0) = current_time`. If the current timestamp exceeds the vesting duration (very likely to happen), the function calculates a vested percentage of 100%, effectively allowing users to claim all tokens without respecting the vesting schedule.

In the `claim_tokens_from_vault()` instruction, the code attempts to use user-specific `token_release_options` first, falling back to the token sale event's options if the user's options are invalid. However, if both sources contain uninitialized default values after an upgrade, the vesting mechanism is completely bypassed.

The highest impact scenario occurs when:

1. A program is upgraded to this version

2. Existing token sale events and user accounts retain default values (0 for both fields)
3. Users can immediately claim 100% of their vested tokens, regardless of the intended vesting schedule

Recommendations:

Consider only allowing the `claim_tokens_from_vault()` instruction to happen when the `token_sale_event.token_release_options` is initialized

```
let token_release_options
    = if
    ctx.accounts.user_account.token_release_options.is_valid(validation_time)
    {
        ctx.accounts.user_account.token_release_options.clone()
    } else {
      require!(
        ctx.accounts.token_sale_event.token_release_options.is_valid(
            validation_time),
        ...
      )
        ctx.accounts.token_sale_event.token_release_options.clone()
    };

        ...
    }
```

Legion: Resolved with @c5dc6d4ed4...

Zenith: Verified.

4.2 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] Fee calculation based on `total_amount` increases absolute fee values

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- anchor/programs/token-sale/src/instructions/deposit_tokens_to_vault.rs#L38-L52
- anchor/programs/token-sale/src/state/token_sale_event.rs#L99-L122

Description:

The calculation of `legion_treasury_wallet_amount` and `referrer_treasury_wallet_amount` is now based on `total_amount` instead of `deposit_amount`. This change results in higher absolute fee values for the same intended deposit. For example, previously, with a `deposit_amount` of 1000 and a 1% fee for each, the `total_amount` required was 1020. Now, with a `total_amount` of 1020 and a 1% fee for each, the resulting `deposit_amount` is only 999.6. This breaks backward compatibility and may cause users to receive less than expected.

Recommendations:

It is recommended to clearly document the resulting net fee/deposit change due to the new basis value for fee calculations.

Or adapt the fee calculation as follows to preserve the previous behaviour:

```
/// Calculate the referrer fee on tokens sold.
pub fn calculate_referrer_fee_on_tokens_sold(&self, tokens_sold: u64) ->
  u64 {
    if self.referrer_percentage_fee_on_tokens_sold > 10000 {
        return 0;
    }
```

```
        (tokens_sold as f64 * self.referrer_percentage_fee_on_tokens_sold as f64 /
            10000.0) as u64
        (tokens_sold as f64 * 10000.0 / (10000.0 + self.referrer_percentage_fee_
            on_tokens_sold as f64)) as u64
    }


/// Calculate the legion fee on tokens sold.
pub fn calculate_legion_fee_on_tokens_sold(&self, tokens_sold: u64) -> u64 {
    if self.legion_percentage_fee_on_tokens_sold > 10000 {
        return 0;
    }

    (tokens_sold as f64 * self.legion_percentage_fee_on_tokens_sold as f64 /
        10000.0) as u64

    (tokens_sold as f64 * 10000.0 / (10000.0 + self.legion_percentage_fee_on_
        tokens_sold as f64)) as u64
}
```

Legion: Resolved by reverting fee calculation change in PR-70.

Zenith: Verified.

[M-2] Zero duration vesting prevents token claims

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- state/vesting_options.rs#L23-L28
- state/vesting_options.rs#L31-L58

Description:

The `VestingOptions.is_valid()` function allows `duration_in_seconds` to be zero as long as either `cliff_in_seconds > 0` OR `duration_in_seconds > 0`. However, when `duration_in_seconds` is zero, the `calculate_vested_amount()` function produces invalid results that prevent users from claiming vested tokens.

In the `calculate_vested_amount()` function, the vested percentage is calculated as:

```
let vested_percentage = vesting_time as f64 / vesting_period as f64;
```

When `duration_in_seconds` is zero, both `vesting_period` and `vesting_time` become zero, resulting in `0 / 0 = NaN`. This NaN value is then used to calculate the vested amount:

```
(user_token_allocation as f64 * vested_percentage).round() as u64
```

The NaN multiplication and rounding operation results in 0, causing `vested_amount = 0` and preventing all token claims for affected vesting schedules.

Recommendations:

Add a check in the `calculate_vested_amount()` function to handle the zero duration case explicitly:

```
pub fn calculate_vested_amount(
    &self,
    user_token_allocation: u64, // The total token allocation for the user
    current_time: u64,          // The current timestamp
    token_vesting_start_time: u64, // The timestamp when the token vesting
    starts
) -> u64 {
    ...

    // Calculate the vesting time as the minimum of the elapsed time and the
    total vesting duration
    let vesting_period = self.duration_in_seconds as u64;

   if vesting_period == 0 {
        return user_token_allocation;
   }

    let vesting_time = elapsed_time.min(vesting_period);

    // Calculate the vested percentage based on the elapsed time and the
    vesting duration
    let vested_percentage = vesting_time as f64 / vesting_period as f64;

    // Calculate the vested amount based on the user's token allocation
    (user_token_allocation as f64 * vested_percentage).round() as u64
}
```

Legion: Resolved with PR-66.

Zenith: Verified.

[M-3] Incorrect `TOKEN_SALE_EVENT_ACCOUNT_SPACE` calculation breaks backward compatibility

`SEVERITY`: Medium	`IMPACT`: Medium
`STATUS`: Resolved	`LIKELIHOOD`: Medium

Target

- anchor/programs/token-sale/src/state/token_sale_event.rs#L40-L63

Description:

The constant `TOKEN_SALE_EVENT_ACCOUNT_SPACE` is currently set to 265 bytes, but it was previously 261 bytes before the recent changes. This discrepancy is due to a miscalculation of the extra padding for upgrades: the value is set to `+ 26` instead of the correct value, which should be `+ 22`. This inconsistency breaks compatibility with existing accounts, as the account size no longer matches what was previously allocated and expected on-chain.

Recommendations:

It is recommended to update the `TOKEN_SALE_EVENT_ACCOUNT_SPACE` calculation to use `+ 22` for the extra padding instead of `+ 26`.

Legion: Resolved with PR-63.

Zenith: Verified.

4.3 Low Risk

A total of 2 low risk findings were identified.

[L-1] Struct space calculation inconsistencies

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- state/token_sale_event.rs#L40-L63
- state/user_account.rs#L39-L58

Description:

Account space calculation constants do not match actual struct field definitions, potentially causing account creation failures.

Key inconsistencies:

- `UserAccount.claim_total_supply_percentage` is `u64` (8 bytes) but space calculation allocates 2 bytes
- `USER_ACCOUNT_SPACE` includes non-existent `is_funds_claimed` field
- `TOKEN_SALE_EVENT_ACCOUNT_SPACE` includes non-existent `token_mint` and `token_total_supply` fields

This mismatch between expected account size and actual struct memory layout leads to incorrect extra padding calculation, which can potentially be dangerous in future upgrades.

Recommendations:

Consider following this fix suggestion

```
pub const TOKEN_SALE_EVENT_ACCOUNT_SPACE: usize = 8 // discriminator
          + 8   // event_id (u64)
          + 32  // project_authority (Pubkey)
          + 1   // status (TokenSaleEventStatus enum)
          + 32  // treasury_wallet (Pubkey)
          // Dates
          + 8   // created_at (u64)
          + 8   // close_refund_period_at (u64)
          // Token related
         + 32  // token_mint (Pubkey)
         + 8   // token_total_supply (u64)
          // Vesting options
          + VESTING_ACCOUNT_SPACE
          // Token Info
          + TOKEN_SALE_EVENT_TOKEN_INFO_ACCOUNT_SPACE
          // Referrer
          + 32  // referrer_treasury_wallet (Pubkey)
          + 2   // referrer_percentage_fee_on_capital_raised (u16)
          + 2   // legion_percentage_fee_on_capital_raised (u16)
          + 2   // referrer_percentage_fee_on_tokens_sold (u16)
          + 2   // legion_percentage_fee_on_tokens_sold (u16)
          + 4   // refund_period_in_seconds (u32)
          + TOKEN_RELEASE_OPTIONS_ACCOUNT_SPACE // token_release_options
         + 22; // extra padding for upgrades
         + 62; // extra padding for upgrades

pub const USER_ACCOUNT_SPACE: usize = 8 // discriminator
          + 32  // token sale event (Pubkey)
          + 1   // status (UserAccountStatus enum)
         + 1   // is_funds_claimed (boolean)
          + 32  // creator (Pubkey)
          // Dates
          + 8   // deposited_at (u64)
          // Token related
          + 8   // claimed_amount (u64)
         + 2   // claim_total_supply_percentage (u16)
         + 8   // claim_total_supply_percentage (u64)

          + 8   // deposit_amount (u64)
          + 32  // deposit_token_mint (Pubkey)
          + 8   // accepted_deposit_amount (u64)
          + 1   // is_deposit_withdrawn (boolean)
          + 1   // is_deposit_amount_accepted (boolean)
          + 1   // is_refundable_amount_withdrawn (boolean)
          + VESTING_ACCOUNT_SPACE // vesting_options
          + 1   // version (u8)
          + TOKEN_RELEASE_OPTIONS_ACCOUNT_SPACE // token_release_options
         + 30; // extra padding for upgrades
         + 25; // extra padding for upgrades
```

Legion: Resolved with PR-68.

Zenith: Verified.

[L-2] Missing validation for refund period parameter

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- instructions/update_token_sale_event_close_refund_period.rs#L45

Description:

The `_update_token_sale_event_close_refund_period()` function accepts a `refund_period_in_seconds` parameter without proper validation, allowing authorities to set unreasonable values that could disrupt the token sale's refund and withdrawal mechanisms.

The function directly assigns the input parameter to the token sale event without bounds checking:

```
ctx.accounts.token_sale_event.refund_period_in_seconds
    = refund_period_in_seconds;
```

This `refund_period_in_seconds` value is critical to the token sale's business logic, as it determines:

1. **User refund eligibility**: The `validate_refund_period()` function uses this value to determine if users can still request refunds through `_refund_funds_from_user_account()`
2. **Project withdrawal timing**: The `validate_withdrawal_period()` function ensures projects can only withdraw deposits AFTER the refund period expires via `_withdraw_deposit_from_user_account()`

There's no upper bound validation, meaning an authority could set the refund period to extremely large values, effectively preventing project withdrawals indefinitely.

Recommendations:

Add bounds validation for the `refund_period_in_seconds` parameter to ensure it falls within reasonable limits

Legion: Resolved with PR-73.

Zenith: Verified.

4.4 Informational

A total of 1 informational findings were identified.

[I-1] SDK function `getDepositTokensToVaultTransaction` uses `depositAmount` instead of `totalAmount`

`SEVERITY`: Informational	`IMPACT`: Informational
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- sdk/src/initialize-sdk.ts#L535

Description:

The `getDepositTokensToVaultTransaction` function is currently still built around the `depositAmount` parameter, which it passes to `getDepositTokensToVaultInstruction`. However, the correct parameter should be `totalAmount`. This can lead to incorrect transaction construction and potentially incorrect amounts being deposited to the vault, as the function does not use the intended value.

Recommendations:

It is recommended to refactor `getDepositTokensToVaultTransaction` to accept and use a `totalAmount` parameter instead of `depositAmount` and update the call to `getDepositTokensToVaultInstruction` to pass `totalAmount`.

Legion: Resolved with PR-62.

Zenith: Verified.