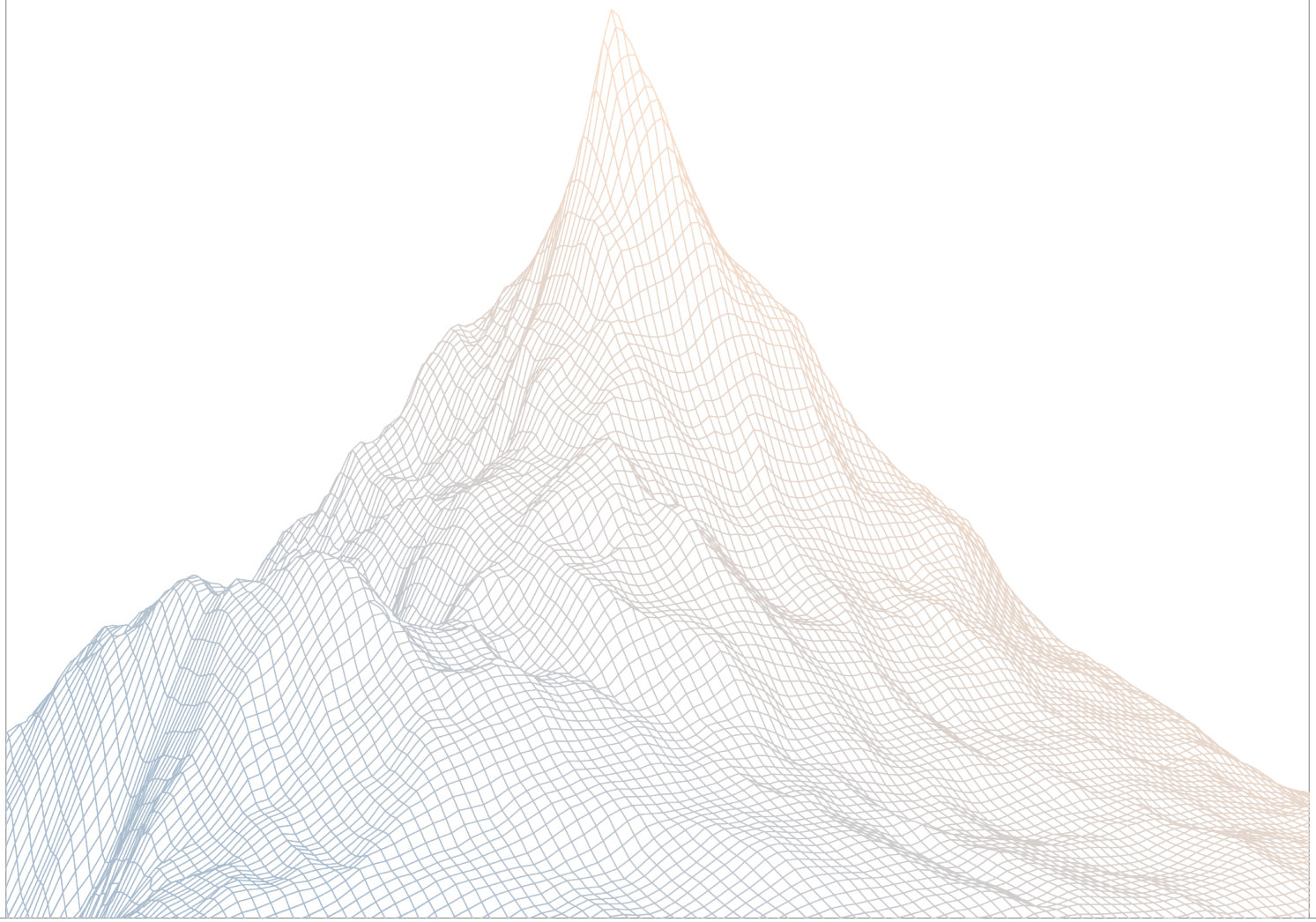
Zenith

Ego Protocol

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: April 10nd to April 17th, 2025

AUDITED BY: J4X
Peakbolt

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About Ego Protocol 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 6

4.1 Critical Risk 7

4.2 High Risk 14

4.3 Medium Risk 17

4.4 Low Risk 23

4.5 Informational 28

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

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 Ego Protocol

Ego is a protocol for launching and trading tokens linked to any profile on the Internet (i.e. X, YouTube, TikTok). The value of each token is determined by the profile's public perception, influence, real-world achievements, and the community. Ego solves key challenges that creator tokens face via token legitimacy, standardized distribution methods, brand safety features, and long-term incentive alignment. Token launches are fully permissionless, enabling viral market dynamics and communities to develop around any social profile.

2.2 Scope

The engagement involved a review of the following targets:

Target	ego-one
Repository	https://github.com/ego-protocol/ego-one
Commit Hash	`064719d8990d159d995df2a3545c95ed1219f53d`
Files	`programs/ego-one/*`

2.3 Audit Timeline

April 10th, 2025	Audit start
April 17th, 2025	Audit end
April 24th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	4
High Risk	1
Medium Risk	3
Low Risk	4
Informational	4
Total Issues	**16**

3

Findings Summary

ID	Description	Status
C-1	execute_migration() can be DoS to prevent migration to raydium pool	Resolved
C-2	ATA can be pre-created to DoS complete_pregrad and start_distribution	Resolved
C-3	Migration can be blocked by sending funds to liquidity token account	Resolved
C-4	Migration can be blocked by donating to pregrad_token_vault	Resolved
H-1	Owner can grief protocol out of fees by providing wrong token account	Resolved
M-1	Rent for pregrad accounts is not refunded in case of refund	Acknowledged
M-2	Last user can force others to pay for his ATA to be able to migrate	Resolved
M-3	Rent is not refunded to user on closing pregrad_user_fund	Acknowledged
L-1	Pool can be migrated at different ratio than pregrad	Resolved
L-2	Graduation price can be manipulated by donating to the token accounts	Resolved
L-3	Minimum deposit is also enforced on top up	Resolved
L-4	Users can claim more than 10% per vesting	Acknowledged
I-1	can_start_refund is called twice in StartRefund IX	Acknowledged
I-2	is_valid_owner() should use require_key_eq!() for key comparison	Resolved
I-3	last_non_zero is actually first_zero_index	Resolved
I-4	Unused error VestingNotFinished	Resolved

4

Findings

4.1 Critical Risk

A total of 4 critical risk findings were identified.

[C-1] `execute_migration()` can be DoS to prevent migration to raydium pool

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

Target

- execute_migration.rs#L81-L93

Description:

In `execute_migration()`, it specifies a `pool_state` PDA that will be initialized by raydium for the storing the pool state.

However, as the `pool_state` PDA has a deterministic address, this allows an attacker to pre-create a raydium pool at that address using the claimed tokens. That will cause `execution_migration()` to fail as the raydium pool already exists and the raydium `initialize` instruction will fail. The result is that migration will always fail and funds will be stuck within the program.

```
/// CHECK: Initialize an account to store the pool state, init by cp-swap
#[account(
    mut,
    seeds = [
        POOL_SEED.as_bytes(),
        amm_config.key().as_ref(),
        token_0_mint.key().as_ref(),
        token_1_mint.key().as_ref(),
    ],
    seeds::program = cp_swap_program.key(),
    bump,
)]
pub pool_state: UncheckedAccount<'info>,
```

Recommendations:

Use a non-deterministic account for the `pool_state` with the following change,

```
#[account(
    mut,
    seeds = [
        POOL_SEED.as_bytes(),
        amm_config.key().as_ref(),
        token_0_mint.key().as_ref(),
        token_1_mint.key().as_ref(),
    ],
    seeds::program = cp_swap_program.key(),
    bump,
)]
pub pool_state: UncheckedAccount<'info>,

#[account(mut)]
pub pool_state: Signer<'info>,
```

Ego Protocol: Resolved with @04cb88841e...

Zenith: Verified. Resolved by using random pool state account for creation.

[C-2] ATA can be pre-created to DoS `complete_pregrad` and `start_distribution`

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

Target

- complete_pregrad.rs#L69-L76
- start_distribution.rs#L54-L63
- start_distribution.rs#L98-L107
- start_distribution.rs#L109-L118

Description:

Both `complete_pregrad` and `start_distribution` instructions uses the anchor `init` constraint to create the relevant associated token accounts (ATA).

However, ATA can be created by interacting directly with the Associated Token Program, which allows anyone to pre-create these ATA before the instructions are called.

That means an attacker can cause `complete_pregrad` and `start_distribution` to fail by pre-creating the required token accounts. The impact is that distribution cannot commence, causing tokens to be stuck in program.

```
/// Profile vest vault account
#[account(
    init,
    payer = payer,
    associated_token::mint = token_mint,
    associated_token::authority = profile_vest,
    associated_token::token_program = token_program,
)]
pub profile_vest_vault: Box<Account<'info, TokenAccount>>,
```

Recommendations:

Use `init_if_needed` instead of `init`.

Ego Protocol: Resolved with @4b618f83afc...

Zenith: Verified.

[C-3] Migration can be blocked by sending funds to liquidity token account

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

Target

- ego-one/src/instructions/admin_migration/execute_migration.rs#L201

Description:

On migration via the `ExecuteMigration` IX the following check occurs.

```
require!(
    ctx.accounts.creator_token_1.amount
    == ctx.accounts.pregrad_data.liquidity_token_supply,
    ErrorCode::InvariantViolated
); // token == liquidity token supply
```

However, before this migration occurs, the pregrad users have already received their tokens. So a malicious user could send a single token to the `creator_token_1` and block migration, resulting in all tokens intended for migration being lost.

Recommendations:

We recommend changing the check as follows.

```
require!(
            ctx.accounts.creator_token_1.amount
    >= ctx.accounts.pregrad_data.liquidity_token_supply,
            ErrorCode::InvariantViolated
        );
```

Any additional donations can be burned afterwards.

Ego Protocol: Resolved with PR-69

Zenith: Verified

[C-4] Migration can be blocked by donating to `pregrad_token_vault`

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

Target

- ego-one/src/instructions/pregrad/complete_pregrad.rs#L140-L146

Description:

When a pregrad gets completed, the program tries to close the `pregrad_token_vault`.

```
close_token_account(
    ctx.accounts.pregrad_token_vault.to_account_info(),
    ctx.accounts.fee_dest.to_account_info(),
    ctx.accounts.pregrad_data.to_account_info(),
    ctx.accounts.token_program.to_account_info(),
    &[&seeds[..]],
)?;
```

The SPL program requires a token account to not hold any tokens to allow someone to close it. The program correctly tries to ensure this by checking if some tokens are left from the rounding in the distribution and burning those.

```
let token_diff = ctx.accounts.pregrad_data.is_within_rounding_tolerance()?;

// Init the profile vest
ctx.accounts.profile_vest.initialize(ctx.bumps.profile_vest,
    ctx.accounts.token_mint.key())?;

// Burn leftover tokens if any
if token_diff > 0 {
    let seeds = PregradDataV1::get_signer_seeds(
        &ctx.accounts.pregrad_data.social,
        &ctx.accounts.pregrad_data.bump,
    );
    burn_tokens_from_vault(
        ctx.accounts.pregrad_data.to_account_info(),
```

```
        ctx.accounts.pregrad_token_vault.to_account_info(),
        ctx.accounts.token_mint.to_account_info(),
        ctx.accounts.token_program.to_account_info(),
        token_diff,
        &[&seeds[..]],
    )?; // the token account balance must be 0 to be closed
}
// Manually close the pregrad_token_vault
let seeds = PregradDataV1::get_signer_seeds(
    &ctx.accounts.pregrad_data.social,
    &ctx.accounts.pregrad_data.bump,
);
```

However, this diff is only `pregrad_token_supply - distributed_token_amount`. As a result, a user who has received their tokens during the distribution can donate one single token to the `pregrad_token_vault`, leading to the close instruction reverting. This way, migration will be blocked entirely, and the tokens will be lost.

Recommendations:

We recommend just burning the whole balance instead of just the diff.

Ego Protocol: Resolved with @b023268fd0...

Zenith: Verified

4.2 High Risk

A total of 1 high risk findings were identified.

[H-1] Owner can grief protocol out of fees by providing wrong token account

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

Target

- ego-one/src/instructions/owner/collect_fees_lp.rs#L125-L139

Description:

Upon creating the distribution, two token vaults are created, which will be used to store LP fees.

```
/// Vault that holds fees. Regular token account (token_mint →
    profile_data).
#[account(
    init,
    payer = payer,
    seeds = [SEED_PROFILE_FEE_LP_TOKEN_PREFIX, token_mint.key().as_ref()],
    bump,
    token::mint = token_mint,
    token::authority = profile_data, // owned by profile data
    token::token_program = token_program,
)]
pub profile_fee_lp_token_vault:  Box<InterfaceAccount<'info, TokenAccount>>,

/// Vault that holds fees. Regular token account (quote_mint →
    profile_data).
#[account(
    init,
    payer = payer,
    seeds = [SEED_PROFILE_FEE_LP_QUOTE_PREFIX, token_mint.key().as_ref()],
    bump,
    token::mint = quote_mint,
```

```
    token::authority = profile_data, // owned by profile data
    token::token_program = token_program,
)]
pub profile_fee_lp_quote_vault:  Box<InterfaceAccount<'info, TokenAccount>>,
```

However, when claiming the LP fees in the `claim_fees_lp` IX, the accounts provided by the owner/admin are only checked to be owned by the `profile_data`, but not to have been created with the correct seed.

```
/// The token account for receive token_0
#[account(
    mut,
    token::mint = token_0_vault.mint,
    token::authority = profile_data, // fee vault is owned by the
    profile_data
)]
pub recipient_token_0_account: Box<InterfaceAccount<'info, TokenAccount>>,

/// The token account for receive token_1
#[account(
    mut,
    token::mint = token_1_vault.mint,
    token::authority = profile_data, // fee vault is owned by the
    profile_data
)]
pub recipient_token_1_account: Box<InterfaceAccount<'info, TokenAccount>>,
```

On claiming the fees again, we require the correct seeds.

```
/// LP fee source account
#[account(
    mut,
    seeds = [SEED_PROFILE_FEE_LP_QUOTE_PREFIX, token_mint.key().as_ref()],
    bump,
    token::mint = quote_mint,
    token::authority = profile_data, // owned by profile_data
    token::token_program = token_program,
)]
pub profile_fee_lp_quote_vault: Box<InterfaceAccount<'info, TokenAccount>>,
```

As a result, a malicious owner could pass a different token account, the authority of which they had previously transferred to `profile_data`, to claim LP fees. This would effectively burn the entire fee, as it can't be claimed anymore, and neither the owner nor the protocol could receive their share. This will also break all further claims as the invariant will be permanently broken.

Recommendations:

We recommend enforcing the check on the seeds in the `collect_fees_lp.rs` instruction.

Ego Protocol: Resolved with @8a508c05fe...

Zenith: Verified

4.3 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] Rent for pregrad accounts is not refunded in case of refund

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: High

Target

- ego-one/src/instructions/pregrad/delete_pregrad.rs#L17-L47

Description:

On calling to init a pregrad, the creator has to pay the rent for 3 accounts:

- `PregradDataV1`
- `TokenAccount`
- `ProfileDataV1`

When a refund occurs, the creator can close these accounts again once all users have been refunded. This is done by calling the `DeletePregrad` IX. However, the protocol will take the rent the creator paid for all these accounts.

```
#[account(
    mut,
    close = fee_dest, // close the pregrad and send the fee collected to
    fee_destination
    constraint = pregrad_data.token_mint == token_mint.key() @
    ErrorCode::InvalidTokenMint,
    seeds = [
        SEED_PREGRAD_DATA_PREFIX,
        pregrad_data.social.social_type.as_bytes(),
        pregrad_data.social.social_id.as_ref(),
    ],
    bump = pregrad_data.bump,
)]
pub pregrad_data: Box<Account<'info, PregradDataV1>>,
```

```
/// Vault that holds the tokens temporarily during the pre-graduation period
#[account(
    mut,
    close = fee_dest, // close the temp vault; token balance should be 0
    already
    associated_token::mint = token_mint,
    associated_token::authority = pregrad_data,
    associated_token::token_program = token_program,
)]
pub pregrad_token_vault: Box<Account<'info, TokenAccount>>,

/// Profile data account
#[account(
    mut,
    close = fee_dest, // close the profile data
    seeds = [SEED_PROFILE_PREFIX, token_mint.key().as_ref()],
    bump = profile_data.bump,
)]
pub profile_data: Box<Account<'info, ProfileDataV1>>,
```

Recommendations:

We recommend refunding the rent to the original creator.

Ego Protocol: Acknowledged

[M-2] Last user can force others to pay for his ATA to be able to migrate

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

Target

- ego-one/src/instructions/pregrad/distribute_tokens.rs#L81-L89

Description:

To migrate the pregrad, all distributions must be finalized. On distribution, an ATA for the new holder of the tokens must be created.

```
/// User's token account to distribute the tokens to
#[account(
    init_if_needed,
    payer = payer,
    associated_token::mint = token_mint,
    associated_token::authority = user,
    associated_token::token_program = token_program,
)]
pub user_token_vault: Box<InterfaceAccount<'info, TokenAccount>>,
```

As a result, a user can not call his distribution and force others to call it for him so that the pregrad can be finished and they can start trading their tokens. This will force others to cover the signature costs as well as the rent for the ATA.

Recommendations:

After discussing this with the protocol team, we learned that the distribution is intended to be solely handled by the protocol team. However, this leads to another issue: the protocol team will need to cover the rent for the ATA, which would severely cut into their fees in case of minimum deposits. As a result, we recommended enforcing an additional deposit at the value of the rent of an ATA by the pregrad depositors. This way the depositors themselves fund their ATA.

Ego Protocol:

Resolved with @e883823e553...

Zenith: Verified

[M-3] Rent is not refunded to user on closing `pregrad_user_fund`

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- ego-one/src/instructions/pregrad/distribute_tokens.rs#L42

Description:

When a user creates a `PregradUserFundV1` account using the `add_fund` IX, he must cover the rent.

```
/// Holds user's fund amount and data during the pre-graduation period
#[account(
    init_if_needed,
    payer = user,
    seeds = [
        SEED_PREGRAD_USER_FUND_PREFIX,
        pregrad_data.key().as_ref(),
        user.key().as_ref(),
    ],
    bump,
    space = ANCHOR_DISCRIMINATOR + PregradUserFundV1::INIT_SPACE,
)]
pub pregrad_user_fund: Box<Account<'info, PregradUserFundV1>>,
```

However, when the tokens get distributed, the account gets closed, but the user isn't refunded the rent as the full amount (deposit and rent) goes to the `pregrad_data`.

```
/// Holds user's fund amount and data during the pre-graduation period
#[account(
    mut,
    close = pregrad_data,
    seeds = [
        SEED_PREGRAD_USER_FUND_PREFIX,
        pregrad_data.key().as_ref(),
        user.key().as_ref(), // enforces that this account is user's account
```

```
    ],
    bump = pregrad_user_fund.bump,
)]
pub pregrad_user_fund: Box<Account<'info, PregradUserFundV1>>,
```

Recommendations:

We recommend refunding the rent to the user who originally paid.

Ego Protocol: Acknowledged.

4.4 Low Risk

A total of 4 low risk findings were identified.

[L-1] Pool can be migrated at different ratio than pregrad

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

Target

- ego-one/src/state/global/config_data.rs#L75-L78

Description:

The program config includes two parameters, which are used for the tokens provided in the pregrad and as liquidity for the pool that gets created on migration.

```
/// Token supply reserved for the liquidity pool (i.e. Raydium)
pub pregrad_liquidity_token_supply: u64,
/// Token supply that will be distributed to the pre-graduation depositors
pub pregrad_token_supply: u64,
```

During the pregrad, the tokens will be distributed at a ratio of `total_fund_amount: pregrad_token_supply`, and in the pool, the starting ratio will be `total_fund_amount: pregrad_liquidity_token_supply`. However, if `pregrad_liquidity_token_supply` $\neq$ `pregrad_token_supply`, the pool will start at a higher or lower ratio than the original pregrad.

Recommendations:

We recommend ensuring that `pregrad_liquidity_token_supply` $=$ `pregrad_token_supply`

Ego Protocol: Resolved with @774b6583329...

Zenith: Verified

[L-2] Graduation price can be manipulated by donating to the token accounts

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

Target

- ego-one/src/instructions/admin_migration/execute_migration.rs#L245-L250

Description:

When the pregrad is migrated to a Raydium pool, the `execute_migration` IX is used. This will call `raydium::initialize` and provide the starting amount of both tokens in the pool.

```
cpi::initialize(
    cpi_context,
    ctx.accounts.creator_token_0.amount,
    ctx.accounts.creator_token_1.amount,
    0, // immediately open
)?;
```

As both of these are just set to the amount of tokens held by the two vaults, an attacker can donate to either of these vaults and manipulate the starting price of the pool.

Recommendations:

We recommend using `pregrad_data.total_fund_amount` for the WSOL and `liquidity_token_supply`. This way the migration will be done at the final price ratio at which the users in the pregrad bought and can not be manipulated.

Ego Protocol:

Resolved with @ddd8477f7c1...

Zenith: Verified

[L-3] Minimum deposit is also enforced on top up

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

Target

- ego-one/src/instructions/pregrad/add_fund.rs#L66-L69

Description:

The pregrad enforces a minimum fund amount, which is intended to prevent dust deposits that could block the migration process. This is implemented in the `add_fund` instruction.

```
let amount = params.amount;
require!(
    amount ⩾ ctx.accounts.config_data.pregrad_min_fund_amount,
    ErrorCode::InvalidInputParams
);
```

However, the `add_fund` instruction can also be used to top up an existing deposit, which must be above the threshold due to the check. This check will enforce the top-up to also be at a minimum of `pregrad_min_fund_amount`, which restricts usage but does not provide any additional safety.

Recommendations:

We recommend ensuring that `amount + pregrad_user_fund.fund_amount ⩾ pregrad_min_fund_amount`. This way, top-ups are not restricted.

Ego Protocol: Resolved with @67c6b8dba54...

Zenith: Verified

[L-4] Users can claim more than 10% per vesting

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: High

Target

- ego-one/src/state/profile/profile_data.rs#L299-L300

Description:

A owner should only be able to claim at max `MAX_VESTING_PPM` üper vesting he claims. This is enforced by the `is_valid_vesting()` function.

```
pub fn is_valid_vesting(&self, vest_amount: u64) -> Result<i64> {
    let now = self.is_after_vesting_cooldown()?; // previous vesting
    cooldown must be over
    self.is_within_max_vesting_amount(vest_amount)?; // cannot exceed max
    vesting amount
    require_gte!(
        self.available_amount,
        vest_amount,
        ErrorCode::InvalidInputParams
    ); // cannot exceed available amount
    Ok(now)
}
```

This function will then use `is_within_max_vesting_amount()` to calculate the maximum vesting amount.

```
fn is_within_max_vesting_amount(&self, vest_amount: u64) -> Result<()> {
    let max_amount =
        Ppm::calculate_ppm_amt(u128::from(self.allocation_amount),
    MAX_VESTING_PPM)
            .ok_or(ErrorCode::InvalidInputParams)?;
    require_gte!(
        max_amount,
        u128::from(vest_amount),
        ErrorCode::InvalidInputParams
    );
```

```
    Ok(())
}
```

To get the ppm into a value, `calculate_ppm_amt()` will be used, which rounds up.

```
pub fn calculate_ppm_amt(amount: u128, ppm: u32) → Option<u128> {
    ceil_div(amount, ppm)
}
```

As a result of the rounding the user will actually be able to vest one lamport more than the intended 10% on each vesting.

Recommendations:

We recommend rounding down when calculating the vesting amounts.

Ego Protocol: Acknowledged

4.5 Informational

A total of 4 informational findings were identified.

[I-1] `can_start_refund` is called twice in `StartRefund` IX

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- ego-one/src/instructions/pregrad/refund_user.rs#L58

Description:

The `StartRefund::process()` function calls `can_start_refund()` right at the start.

```
impl<'info> StartRefund<'info> {
    pub fn process(ctx: Context<StartRefund>) -> Result<()> {
        // Validate
        require!(
            ctx.accounts.pregrad_token_vault.amount == TOKEN_SUPPLY_TOTAL,
            ErrorCode::InvariantViolated,
        );
        ctx.accounts.config_data.require_pregrad_enabled()?;
        ctx.accounts.pregrad_data.can_start_refund()?;
```

At the end, it calls `ctx.accounts.pregrad_data.start_refund()` which will call `can_start_refund()` again.

```
// ----- Refund -----
pub fn start_refund(&mut self) -> Result<()> {
    self.can_start_refund()?;
    self.status = PregradStatus::RefundStart;
    Ok(())
}
```

Recommendations:

We recommend removing one of the checks.

Ego Protocol: Acknowledged

[I-2] `is_valid_owner()` should use `require_key_eq!()` for key comparison

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

Target

- profile_data.rs#L252

Description:

`is_valid_owner()` uses `require_eq!()` to compare two pubkeys, but it is recommended to use `require_keys_eq!()` instead as stated in anchor docs.It is more expensive to compare pubkeys using `require_eq!()`.

```
pub fn is_valid_owner(&self, signer: Pubkey) → Result<()> {
    self.is_claimed()?; // must be claimed
    self.is_after_owner_social_cooldown()?; // cooldown must be over
    require_eq!(self.profile_owner, signer, ErrorCode::NotProfileOwner); //
    profile owner must match
    Ok(())
}
```

Recommendations:

```
pub fn is_valid_owner(&self, signer: Pubkey) -> Result<()> {
    self.is_claimed()?; // must be claimed
    self.is_after_owner_social_cooldown()?; // cooldown must be over

require_eq!(self.profile_owner, signer, ErrorCode::NotProfileOwner);
        // profile owner must match

require_keys_eq!(self.profile_owner, signer, ErrorCode::NotProfileOwne
        r); // profile owner must match
    Ok(())
}
```

Ego Protocol: Resolved with @752b7699f47...

Zenith: Verified

[I-3] `last_non_zero` is actually `first_zero_index`

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

Target

- ego-one/src/utils/social.rs#L63

Description:

In the `get_valid_social_type_id()` function, the code verifies the correctness of the provided social ID. One constraint is that the ID, after the first zero, should only include zeros.

```
// Validate that zeros only appear at the end
let mut last_non_zero = MAX_SOCIAL_ID_LEN;
for i in 0..MAX_SOCIAL_ID_LEN {
    if fixedlen_social_id[i] == 0 {
        last_non_zero = i;
        break;
    }
}

// Ensure we found at least one non-zero byte (no leading zeros)
require!(last_non_zero > 0, ErrorCode::InvalidInputParams);
// Check that all bytes after the first zero are also zero
for i in last_non_zero..MAX_SOCIAL_ID_LEN {
    require!(fixedlen_social_id[i] == 0, ErrorCode::InvalidInputParams);
}
```

Here, the `last_non_zero` gets set to the index where the first zero is found. So the actual index of the last non-zero entry is `i-1`.

Recommendations:

We recommend renaming the variable to `first_zero_index`.

Ego Protocol: Resolved with @ff9fdd862c4...

Zenith: Verified

[I-4] Unused error `VestingNotFinished`

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

Target

- ego-one/blob/abefdd32b34421674d3f4568c4d37f57a9c56153/programs/ego-one/src/error.rs

Description:

The `errors.rs` file includes the `VestingNotFinished` error which is not used anywhere.

Recommendations:

We recommend removing the unused error.

Ego Protocol: Resolved with @752b7699f47...

Zenith: Verified