Table of contents

Contest Summary

Sponsor: First Flight #36

Dates: Mar 20th, 2025 - Mar 27th, 2025

See more contest details here

Results Summary

Number of findings:

- High: 2
- Medium: 4
- Low: 1

High Risk Findings

H-01. Missing Contribution Amount Tracking

Description

The `contribution.amount` field is initialized to `0` but **never updated**, even after a contributor sends SOL to the fund. This breaks refund logic and contribution tracking.

Impact

- **Incorrect Refunds:** Contributors will receive `0` SOL on refund, even if they contributed multiple times.
- **Loss of Trust:** Contributors cannot verify their total contributions.

Affected Code

```
// Line 41
// Initialize contribution.amount to 0
contribution.amount = 0;

// Missing: Update contribution.amount after transfer
```

Recommendation

Update `contribution.amount` using `checked_add`:

```
contribution.amount = contribution.amount
    .checked_add(amount)
    .ok_or(ErrorCode::CalculationOverflow)?;
```

H-02. Missing Deadline and Goal Checks in `withdraw` Function

Description

The `withdraw` function allows the creator to withdraw funds **without verifying**:

1. If the campaign deadline has passed.

2. If the fundraising goal (`amount_raised >= goal`) has been met.

This enables the creator to drain funds prematurely, even if the campaign failed or is ongoing.

Impact

- **Fund Theft:** The creator can withdraw funds before the deadline or before the goal is met, violating the protocol's rules.
- **Loss of Trust:** Contributors lose confidence in the platform, as funds are not safeguarded by basic campaign logic.

Affected Code

```
pub fn withdraw(ctx: Context<FundWithdraw>) -> Result<()> {
    let amount = ctx.accounts.fund.amount_raised;

    // Missing checks for deadline and goal!
    // Creator can withdraw at any time.
}
```

Recommendation

Add explicit checks for the deadline and goal:

```
// Ensure deadline has passed
let current_time = Clock::get()?.unix_timestamp.try_into().unwrap();
require!(fund.deadline < current_time, ErrorCode::DeadlineNotReached);

// Ensure goal is met
require!(fund.amount_raised >= fund.goal, ErrorCode::GoalNotMet);
```

1. Add Missing Error Variant:

```rust
#[error_code]
pub enum ErrorCode {
    // ...
    #[msg("Campaign goal not met")]
    GoalNotMet,
}
```

Medium Risk Findings

M-01. Missing `amount_raised` Reset After Withdrawal

Description

The `withdraw` function transfers the total `amount_raised` to the creator but **does not reset** `fund.amount_raised` to `0` afterward. This allows the creator to repeatedly withdraw the same funds, draining the fund account.

Impact

- **Fund Drainage:** The creator can withdraw the `amount_raised` multiple times, even after funds have already been transferred.
- **Protocol Integrity Loss:** The `amount_raised` value becomes untrustworthy, breaking fund accounting logic.

Affected Code

```
pub fn withdraw(ctx: Context<FundWithdraw>) -> Result<()> {
    let amount = ctx.accounts.fund.amount_raised;

    // Transfers funds to creator...
    **ctx.accounts.fund.to_account_info().try_borrow_mut_lamports()? =
        ctx.accounts.fund.to_account_info().lamports()
        .checked_sub(amount)
        .ok_or(ProgramError::InsufficientFunds)?;

    **ctx.accounts.creator.to_account_info().try_borrow_mut_lamports()?
=
        ctx.accounts.creator.to_account_info().lamports()
        .checked_add(amount)
        .ok_or(ErrorCode::CalculationOverflow)?;

    // MISSING: Reset amount_raised to 0!
    Ok(())
}
```

Recommendation

Reset `amount_raised` to `0` after withdrawal:

```
// After transferring funds:
fund.amount_raised = 0;
```

M-02. Missing `deadline_set` Flag Update and Typo

Description

The `set_deadline` function updates the `deadline` field but **does not set** the `deadline_set` flag (misspelled as `dealine_set` in the account struct) to `true`. This allows the creator to call `set_deadline` multiple times, bypassing the intended single-use constraint.

Impact

- **Unintended Deadline Changes:** The creator can repeatedly modify the deadline, disrupting fund logic (e.g., extending deadlines indefinitely).
- **Code Confusion:** The misspelled `dealine_set` field causes readability issues and potential future bugs.

Affected Code

```
// Misspelled struct field:
pub dealine_set: bool, //  Should be `deadline_set`

// In set_deadline function:
pub fn set_deadline(...) {
    // ...
    fund.deadline = deadline;
    // MISSING: fund.deadline_set = true;
}
```

Recommendation

1. **Fix the Typo:** Rename the struct field to `deadline_set`.
2. **Update the Flag:** Set `deadline_set` to `true` after assigning the deadline.

Corrected Code:

```rust
// Line 222
#[account]
#[derive(InitSpace)]
pub struct Fund {
 // ...

  pub dealine_set: bool,

}
// Line 67
pub fn set_deadline(...) {
    // ...
    fund.deadline = deadline;
    fund.deadline_set = true; // Set flag
}
```

M-03. Missing `amount_raised` Update on Refund

Description

The `refund` function resets the contributor's `contribution.amount` to `0` but **does not reduce** the fund's `amount_raised` by the refunded amount. This results in incorrect tracking of the total funds raised, making the protocol believe the fund has more SOL than it actually holds.

Impact

- **Incorrect Fund Accounting:** The `amount_raised` value becomes inflated, misleading contributors and creators about the fund's progress.
- **Operational Risks:** Creators might withdraw more funds than available, or contributors could be denied refunds due to insufficient SOL in the fund.

Affected Code

```
pub fn refund(ctx: Context<FundRefund>) -> Result<()> {
    let amount = ctx.accounts.contribution.amount;

    // Refund logic...
    ctx.accounts.contribution.amount = 0;

    // MISSING: Update fund.amount_raised
    Ok(())
}
```

Recommendation

Subtract the refunded amount from `fund.amount_raised` using `checked_sub` to prevent underflow:

```
fund.amount_raised = fund.amount_raised
    .checked_sub(amount)
    .ok_or(ErrorCode::CalculationOverflow)?;
```

M-04. `amount_raised` Not Reduced on Refund

Description

When a contributor requests a refund, the `fund.amount_raised` value is **not reduced** by the refunded amount. This results in an inflated total of raised funds, misrepresenting the actual balance held by the fund.

Impact

- **Inaccurate Accounting:** The protocol will report a higher `amount_raised` than the actual SOL held in the fund.
- **Operational Risks:** Creators may withdraw more funds than available, or contributors might be denied refunds due to insufficient SOL.

Affected Code

```
// Line 83
pub fn refund(ctx: Context<FundRefund>) -> Result<()> {
    let amount = ctx.accounts.contribution.amount;
    // ...
    ctx.accounts.contribution.amount = 0;
    // MISSING: fund.amount_raised -= amount;
}
```

Recommendation

Use `checked_sub` to safely decrement `amount_raised` and handle underflow:

```
fund.amount_raised = fund.amount_raised
    .checked_sub(amount)
    .ok_or(ErrorCode::CalculationOverflow)?;
```

Low Risk Findings

L-01. Direct Lamport Manipulation

Description

The `refund` and `withdraw` functions directly manipulate lamports (SOL) using `try_borrow_mut_lamports()`, bypassing Solana's native `system_program::transfer` logic. This approach is error-prone and violates best practices for handling SOL transfers.

Impact

- **Fund Loss:** Incorrect lamport arithmetic (e.g., underflow/overflow) can corrupt balances.
- **Reentrancy Risks:** Manual lamport updates lack atomicity, opening vectors for exploits.
- **Protocol Instability:** Direct manipulation bypasses system-level security checks.

Affected Code

```
// In refund():
// Line 83
**ctx.accounts.fund.to_account_info().try_borrow_mut_lamports()? =
    ctx.accounts.fund.lamports().checked_sub(amount)?;

**ctx.accounts.contributor.to_account_info().try_borrow_mut_lamports()?
=
    ctx.accounts.contributor.lamports().checked_add(amount)?;

// In withdraw():
// Line 113
**ctx.accounts.fund.to_account_info().try_borrow_mut_lamports()? =
    ctx.accounts.fund.lamports().checked_sub(amount)?;

**ctx.accounts.creator.to_account_info().try_borrow_mut_lamports()? =
    ctx.accounts.creator.lamports().checked_add(amount)?;
```

Recommendation

Replace manual lamport updates with `system_program::transfer` for secure SOL movements:

For `refund()`:

```
let cpi_context = CpiContext::new(
    ctx.accounts.system_program.to_account_info(),
    system_program::Transfer {
        from: ctx.accounts.fund.to_account_info(),
        to: ctx.accounts.contributor.to_account_info(),
    },
);
system_program::transfer(cpi_context, amount)?;
```

For `withdraw()`:

```rust
let cpi_context = CpiContext::new(
    ctx.accounts.system_program.to_account_info(),
    system_program::Transfer {
        from: ctx.accounts.fund.to_account_info(),
        to: ctx.accounts.creator.to_account_info(),
    },
);
system_program::transfer(cpi_context, amount)?;
```