code4rena

Pie.fun

Smart Contract Security Assessment

Version 2.0

Audit dates: Dec 12 — Dec 17, 2024

Audited by: Peakbolt
VictorMagnum

Contents

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 Pie.fun

Pie.fun is the multichain memecoin index platform that allows people purchase memecoins easier across different chains

2.2 Scope

Repository	ao-labs/pie-dot-fun-solana/tree/main/programs/pie/src/
Commit Hash	d61fec279571b993f213050ea945f9c2992281d0
Mitigation Hash	0dbade95b7c42a18833ff3127e2044ef438fc1f1

2.3 Audit Timeline

DATE	EVENT
Dec 12, 2024	Audit start
Dec 17, 2024	Audit end
Jan 07, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	3
High Risk	4
Medium Risk	5
Low Risk	5
Informational	2
Total Issues	19

3. Findings Summary

ID	DESCRIPTION	STATUS
C-1	Lack of validation on `token_mint` in `execute_rebalancing` allows token theft by inflating basket token value	Resolved

C-2	Incorrect update in `execute_rebalancing()` allows malicious rebalancer to steal from basket	Resolved
C-3	Arbitrary CPI vulnerability allows malicious rebalancer to steal component tokens from basket	Resolved
H-1	Missing token account validation in mint basket token instruction	Resolved
H-2	Users will lose component tokens when calling `redeem_basket_token()`	Resolved
H-3	Incorrect mint calculation for `mint_token_basket`	Resolved
H-4	Lack of validation for `args.components` in `create_basket` will affect minting/redeeming of basket tokens	Resolved
M-1	User cannot withdraw their funds at special occasions	Resolved
M-2	Missing validation of `wrapped_sol_mint` allows bypass of margin check	Resolved
M-3	`quantity_in_sys_decimal` could round to zero and DoS components buying	Resolved
M-4	Buying/selling components could be DoS as zero amount components are not removed	Resolved
M-5	Missing `is_rebalancing == false` check for minting and redemption of basket token	Resolved
L-1	Non-Canonical Bump Used in PDA Validation	Resolved
L-2	Selling/buying of components can be DoS when `creator_token_account` is closed	Acknowledged
L-3	`stop_rebalancing()` can be DoS by with a WSOL transfer	Resolved
L-4	`update_fee()` fails to ensure fee percentages are below `BASIS_POINTS`	Resolved
L-5	`transfer_admin()` could cause admin role to be lost	Acknowledged

I-1	Redundant authorization check in update rebalancer instruction	Resolved
I-2	Missing constraint on `PROGRAM_STATE`	Resolved

4. Findings

4.1 Critical Risk

A total of 3 critical risk findings were identified.

[C-1] Lack of validation on `token_mint` in `execute_rebalancing` allows token theft by inflating basket token value

Severity: Critical	Status: Resolved

Context:

- execute_rebalancing.rs#L195-L197
- execute_rebalancing.rs#L266-L268

Description: In `execute_rebalancing()`, the account `token_mint` is used to find and update the component.

However, there is no validation on `token_mint` to ensure that it is the same as `vault_token_destination`.

This issue allows a malicious rebalancer to inflate the basket token value by passing in a `token_mint` to increase the quantity of a component token that has a higher value than the actual purchased token component. The rebalancer can then steal from the basket by redeeming basket tokens after inflating the basket token value.

```
        accounts.vault_token_destination.reload()?;

        let token_mint = accounts.token_mint.key();

        let quantity_in_sys_decimal =
Calculator::apply_sys_decimal(accounts.vault_token_destination.amount).checked_div(total_supply.try_into().unwrap()).unwrap();

        if let Some(component) = basket_config
            .components
            .iter_mut()
            .find(|c| c.mint == token_mint)
        {
            component.quantity_in_sys_decimal = quantity_in_sys_decimal;
        } else {
            basket_config.components.push(BasketComponent {
                mint: token_mint,
                quantity_in_sys_decimal,
            });
        }
```

Recommendation: For both buy and sell in `execute_rebalancing()`, use the mint account in vault_token_destination and vault_token_source instead of passing it in via `token_mint`.

Pie.fun: Fixed in the following commit

Zenith: Verified

[C-2] Incorrect update in `execute_rebalancing()` allows malicious rebalancer to steal from basket

Severity: Critical Status: Resolved

Context:

- execute_rebalancing.rs#L201-L212
- execute_rebalancing.rs#L272-L282

Description: In `execute_rebalancing()`, it will update the basket components based on `vault_token_destination` if it is a buy and based on `vault_token_source` if it is a sell.

However, it should actually update the basket components based on both `vault_token_destination` and `vault_token_source` regardless of a buy or sell. That is because the rebalancing will change the component quantity for both the source and destination tokens.

A malicious rebalancer can exploit this issue by minting basket tokens and then `execute_rebalancing` with `is_buy = true` to cause the basket components quantity to be inflated since it only increase the quantity for the purchased destination tokens, without reducing the source token that was used for the purchase. The rebalancer can then steal from the basket by redeem the basket tokens based on the inflated component quantity.

```
pub fn execute_swap<'a: 'info, 'info>(
        ...
    if is_buy {
        ...
        let quantity_in_sys_decimal =

Calculator::apply_sys_decimal(accounts.vault_token_destination.amount).ch
ecked_div(total_supply.try_into().unwrap()).unwrap();

        if let Some(component) = basket_config
            .components
            .iter_mut()
            .find(|c| c.mint == token_mint)
        {
            component.quantity_in_sys_decimal = quantity_in_sys_decimal;
        } else {
            basket_config.components.push(BasketComponent {
                mint: token_mint,
                quantity_in_sys_decimal,
            });
        }
        ...
    } else {
       ...
        let quantity_in_sys_decimal =

Calculator::apply_sys_decimal(accounts.vault_token_source.amount).checked_div(total_supply.try_into().unwrap()).unwrap();

        if quantity_in_sys_decimal == 0 {
            basket_config.components.retain(|c| c.mint != token_mint);
        } else {
            if let Some(component) = basket_config
                .components
                .iter_mut()
                .find(|c| c.mint == token_mint)
            {
                component.quantity_in_sys_decimal =
quantity_in_sys_decimal;
            }
        }
     ...
```

Recommendation: For both buy and sell in `execute_rebalancing()`, update the basket components based on both `vault_token_destination` and `vault_token_source` regardless of a buy or sell.

Pie.Fun: Fixed in PR-60

Zenith: Verified

[C-3] Arbitrary CPI vulnerability allows malicious rebalancer to steal component tokens from basket

Severity: Critical	Status: Resolved

Context:

- buy_component.rs#L169-L172
- sell_component.rs#L179-L183
- execute_rebalancing.rs#L260-L264

Description: Raydium AMM is used to perform the swap in buying/selling of components and rebalancing of basket.

However, there are no checks to ensure that the correct Raydium program is invoked for the swaps in these functions. This allows malicious users to call the functions with a fake raydium program.

For `buy_component()`, it allows malicious users to bypass fee payments by using a fake raydium program that simply transfer the component tokens to the destination vault token account with `amount_swapped = 0`. This will cause platform and creator fees to be zero as the fees are charged on the `amount_swapped`.

Similiarly for `sell_component()`, it allows one to evade fee payments with a fake raydium program that causes `amount_received` to be zero.

In the worst case, a malicious rebalancer can use a fake raydium program to steal all the component tokens using `execute_rebalancing` to swap all the tokens in the basket for zero WSOL.

```
    pub token_program: Program<'info, Token>,
    /// CHECK: amm_program
>>> pub amm_program: AccountInfo<'info>,
    pub system_program: Program<'info, System>,

         ...

        let cpi_context = CpiContext::new_with_signer(
>>>         accounts.amm_program.to_account_info(),
            SwapBaseIn {

         ...

        solana_program::program::invoke_signed(
            &swap_base_out_inx,
            &ToAccountInfos::to_account_infos(&cpi_context),
            signer,
        )?;
```

Recommendation: Check that `ctx.accounts.amm_program.key()` matches raydium program id.

Pie.fun: Fixed in the following [commit]()

Zenith: Verified. Resolved as with address constraint on `amm_program` to check against raydium public key.

4.2 High Risk

A total of 4 high risk findings were identified.

[H-1] Missing token account validation in mint basket token instruction

Severity: High	**Status:** Resolved

Context:

- mint_basket_token.rs

Description: The mint_basket_token instruction lacks token account validations for the user_basket_token_account. The current implementation:

```
pub user_basket_token_account: Box<Account<'info, TokenAccount>>,
```

fails to validate that the token account's:

1. Mint matches the basket_mint being used for minting
2. Account's authority (owner) is the user signing the transaction

This could lead to several issues such as:

1. Tokens could be minted to an account of the wrong mint type
2. Transaction could succeed even with mismatched accounts, leading to locked or lost tokens
3. Potential exploit vector where an attacker provides a token account they control but shouldn't have access to

Note that the same issue is present here

Recommendation: Add proper token account validation constraints using Anchor's account validation system:

Pie.fun: Fixed with PR-56

Zenith: Verified

[H-2] Users will lose component tokens when calling `redeem_basket_token()`

Severity: High	Status: Resolved

Context:

- redeem_basket_token.rs#L81-L97

Description: When redeeming basket token, users could receive component tokens that are already created in `user_fund`.

However, when the component does not exist in `user_fund`, it will not update the component in `user_fund` as the else case is missing.

This will prevent users from selling the components, losing those tokens.

```
    for token_config in basket_config.components.iter() {
        if let Some(asset) = user_fund
            .components
            .iter_mut()
            .find(|a| a.mint == token_config.mint)
        {
            let amount_return: u128 = token_config
                .quantity_in_sys_decimal
                .checked_mul(amount.into())
                .unwrap();

            asset.amount = asset
                .amount

.checked_add(Calculator::restore_raw_decimal(amount_return))
                .unwrap();
        }
    }
```

Recommendation: Update `redeem_basket_token()` to add component to `user_fund` if it does not already exists.

Pie.fun: Fixed with the following commit

Zenith: Resolved as per recommendations.

[H-3] Incorrect mint calculation for `mint_token_basket`

Severity: High Status: Resolved

Context:

- mint_basket_token.rs#L110-L120

Description: The calculation for `calculate_possible_mint_amount()` and `calculate_deduct_amount()` will be incorrect when the basket decimals is not `SYS_DECIMALS`.

```
fn calculate_deduct_amount(basket_token_amount: u128,
quantity_in_sys_decimal: u128) -> Result<u64> {
    let amount_to_deduct = quantity_in_sys_decimal
        .checked_mul(basket_token_amount).unwrap();
    Ok(Calculator::restore_raw_decimal(amount_to_deduct))
}

fn calculate_possible_mint_amount(user_asset_amount: u64,
quantity_in_sys_decimal: u128) -> Result<u128> {
    let user_amount_in_system_decimal =
Calculator::apply_sys_decimal(user_asset_amount);


Ok(user_amount_in_system_decimal.checked_div(quantity_in_sys_decimal).unw
rap())
}
```

To understand the issue, lets look at `calculate_possible_mint_amount()`.

Suppose we have the following decimal precisions for minting basket X, assuming we have only one component token A.

- `quantity_in_sys_decimal` is 10^9 (this is the required quantity of token A for 1 basket X token)
- `user_asset_amount` is 10^9 (this is the user's balance of token A in user_fund)
- basket mint decimals is 10^9
- `calculate_possible_mint_amount()` = 10^9 * 10^6 / 10^9 = 10^6

As we can see, `calculate_possible_mint_amount()` returns 10^6, which represents 0.001 basket token (basket mint decimals is 10^9). It should instead be 10^9, to represents 1 basket token.

That is because `user_amount_in_system_decimal` is adjusted by `SYS_DECIMALS` due to `apply_sys_decimal()`, which is not the same as the basket token deicmals.

Recommendation: This can be resolved by simply hardcoding basket decimals to `SYS_DECIMALS` during `create_basket`.

Pie.fun: Fixed in the following commit

Zenith: Verified. Resolved by fixing to 6 decimals (SYS_DECIMALS).

[H-4] Lack of validation for `args.components` in `create_basket` will affect minting/redeeming of basket tokens

Severity: High

Status: Resolved

Context:

- create_basket.rs#L94

Description: The function `create_basket()` does not validate `args.components`, which can lead to the following issues,

1. Creators can call `create_basket()` with duplicate components. This will cause `mint_basket_token()` and `redeem_basket_token()` to mint/redeem basket token incorrectly, as both assumes the components are unique.
2. They can also create components with invalid mint or `quantity_in_sys_decimal = 0`, which prevents minting of basket tokens.

```
pub fn create_basket(ctx: Context<CreateBasketContext>, args:
CreateBasketArgs) -> Result<()> {
    let program_state = &ctx.accounts.program_state;

    if !program_state.enable_creator {
        let current_admin = program_state.admin;
        if ctx.accounts.creator.key() != current_admin {
            return Err(PieError::Unauthorized.into());
        }
    }

    let basket_config = &mut ctx.accounts.basket_config;
    let config = &mut ctx.accounts.program_state;

    basket_config.bump = ctx.bumps.basket_config;
    basket_config.creator = ctx.accounts.creator.key();
    basket_config.rebalancer = args.rebalancer;
    basket_config.id = config.basket_counter;
    basket_config.mint = ctx.accounts.basket_mint.key();
>>> basket_config.components = args.components.clone();
```

Recommendation:

1. Add a check to ensure there are no duplicate components during creation of basket.

2. Validate that the mint is valid (e.g. not empty and program id is token_program) and `quantity_in_sys_decimal > 0`.

Pie.fun: Fixed in the following commit

Zenith: Verified. Resolved with checks to ensure `component.mint` is not empty and it is not duplicated,. Also check that `component.quantity_in_sys_decimal > 0`.

4.3 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] User cannot withdraw their funds at special occasions

Severity: Medium	Status: Resolved

Context:

- buy_component.rs
- execute_rebalancing.rs

Description: *Scenario*

- Let's say basket 1 is comprised of token A only.
- User calls buyComponent to buy token A.
- User forgets to mintBasketToken.
- The basket is rebalanced, now it is 100% token B.
- Now the user cannot withdraw funds by calling sellComponent (because there is no token A in the vault), the user cannot mint basket token either because the user needs token B.
- In this case, the user's fund will get locked

This will not happen in most cases since we are using JITO bundle to make buyComponent & mintBasketToken transactions atomic. However this can happen when JITO unbundles or if a user tries to buy components buy directly calling the functions.

Recommendation:

- keep track of the amounts of tokens in the vault that have not been used to mint basket tokens.
- when rebalancing, the rebalancer can only use the amount that has been used to mint basket tokens (aka. amount that are locked)

Pie.fun: Fixed in PR-60

Zenith: Verified.

Resolved by only rebalancing amount that was used to mint basket tokens. It is done by computing the amount not used for minting basket token, when updating for token_source and token destination. It is assumed that the rebalancer is trusted by the users to not swap into malicious tokens that can alter the un-minted amounts.

[M-2] Missing validation of `wrapped_sol_mint` allows bypass of margin check

Severity: Medium	Status: Resolved

Context:

- stop_rebalancing.rs#L35-L36

Description:

`stop_rebalancing()` has a margin check that verifies `wrapped_sol_balance < rebalance_margin_lamports`.

However, there is no constraint on `wrapped_sol_mint` to validate that it is indeed the mint account of wrapped SOL.

This allows the rebalancer to bypass the margin check with a mint address for another token that has a balance less than `rebalance_margin_lamports`.

```
#[account(mut)]
pub wrapped_sol_mint: Box<InterfaceAccount<'info, Mint>>,
```

Recommendation: Validate that `wrapped_sol_mint == NATIVE_MINT`.

Consider removing the check if the rebalancer is trusted to rebalance the basket token composition to the benefits of the users.

Pie.fun: Fixed with the following commit

Zenith: Verified. Resolved by adding constraint on `wrapped_sol_mint` to ensure `address = NATIVE_MINT`.

[M-3] `quantity_in_sys_decimal` could round to zero and DoS components buying

Severity: Medium　　Status: Resolved

Context:

- execute_rebalancing.rs#L198-L199

Description: During `execute_rebalancing()` the component's `quantity_in_sys_decimal` is updated using `vault_token_destination.amount/ total_supply` for buy.

```
        let quantity_in_sys_decimal =

Calculator::apply_sys_decimal(accounts.vault_token_destination.amount).checked_div(total_supply.try_into().unwrap()).unwrap();
```

However, `quantity_in_sys_decimal` could round to zero when `total_supply > vault_token_destination.amount * SYS_DECIMALS`, which will cause it to push zero quantity `component` into basket `components` or update the existing component amount to zero. This will lead to a DoS for `buy_component()` as it cannot buy zero component.

Recommendation: For buy in `execute_rebalancing`, add a check to revert on when `quantity_in_sys_decimal == 0`.

Pie.fun: Fixed in the following commit

Zenith: Verified.

[M-4] Buying/selling components could be DoS as zero amount components are not removed

Severity: Medium Status: Resolved

Context:

- mint_basket_token.rs#L70-L83
- sell_component.rs#L215-L216

Description: In both `mint_basket_token()` and `sell_component()`, the `component` is not removed from `user_fund` when the `component.amount == 0` after deducting it.

This will cause the number of zero amount `components` in the `user_fund` to increase if the basket is rebalanced with new components. When it hits `MAX_COMPONENTS (50)`, it will cause the `buy_components()` to fail, preventing users from calling `mint_basket_token()`.

```
    for token_config in basket_config.components.iter() {
        if let Some(asset) = user_fund
            .components
            .iter_mut()
            .find(|a| a.mint == token_config.mint)
        {
            let amount_to_deduct_in_raw_decimal =
calculate_deduct_amount(basket_token_amount.into(),
token_config.quantity_in_sys_decimal)?;

            asset.amount = asset
                .amount
                .checked_sub(amount_to_deduct_in_raw_decimal)
                .ok_or(PieError::InsufficientBalance)?;
        }
    }
```

Recommendation: Remove `component` from `user_fund` when `component.amount == 0` after deduction, in both `mint_basket_token()` and `sell_component()`.

Pie.fun: Fixed in the following commit

Zenith: Verified.

[M-5] Missing `is_rebalancing == false` check for minting and redemption of basket token

Severity: Medium	Status: Resolved

Context:

- mint_basket_token.rs#L51
- redeem_basket_token.rs#L57

Description: Both `mint_basket_token()` and `redeem_basket_token()` fails to check `is_rebalancing == false`, which allows users to mint/redeem basket tokens when rebalancing is occurring.

This will cause the mint/redeem to fail as the buy/sell components done with bundler will not match the basket components.

Recommendation: Add a check in `mint_basket_token()` and `redeem_basket_token()` to revert on `is_rebalancing == false`.

Pie.fun: Fixed in the following commit

Zenith: Verified.

4.4 Low Risk

A total of 5 low risk findings were identified.

[L-1] Non-Canonical Bump Used in PDA Validation

Severity: Low	Status: Resolved

Context:

- update_rebalancer.rs

Description:

The update_rebalancer.rs instruction uses a non-canonical bump for PDA validation in the basket_config account constraint. Instead of using the canonical bump stored in the BasketConfig account's bump field, it uses a derived bump through the bump constraint. This is problematic because It allows the instruction to validate against any valid bump that produces a PDA, not just the canonical one that was used during account initialization

You can read more here

Note, there are other occurrences in the codebase with the same issue.

Recommendation: Use the canonical bump stored in the BasketConfig account for PDA validation:

```
#[account(
    mut,
    seeds = [BASKET_CONFIG, &basket_config.id.to_be_bytes()],
    bump = basket_config.bump,  // <-- Fix: Use stored canonical bump
    constraint = basket_config.creator == creator.key() @
PieError::Unauthorized
)]
```

Pie.fun: Fixed with the following commit

Zenith: Verified

[L-2] Selling/buying of components can be DoS when `creator_token_account` is closed

Severity: Low Status: Acknowledged

Context:

- sell_component.rs#L204-L213
- buy_component.rs#L197-L206

Description: When selling/buying components, the `creator_fee_amount` is transferred to the `creator_token_account`.

However, if the creator closes the `creator_token_account`, this transfer will fail and cause `sell_component()` to revert since there is no more account to transfer to.

The impact is a temporary DoS as users will can workaround it by creating `creator_token_account` on behalf and pay the account opening cost. It is also in the creator's interests to keep the account and collect fees.

```
    // Transfer creator fee to creator
    if creator_fee_amount > 0 {
        transfer_from_user_to_pool_vault(
            &ctx.accounts.user_token_source.to_account_info(),
            &ctx.accounts.creator_token_account.to_account_info(),
            &&ctx.accounts.user_source_owner.to_account_info(),
            &ctx.accounts.token_program.to_account_info(),
            creator_fee_amount,
        )?;
    }
```

Recommendation: To fix this, one option is to skip transfer if the account does not exists like this. Note that the anchor constraints for `creator_token_account` will need to be shifted to a check in the function itself, so that it will not fail during constraint checking.

```
if creator_fee_amount > 0 {
    if let Ok(_) = anchor_spl::token::get_account_info(
        &ctx.accounts.creator_token_account.to_account_info()
    ) {
        transfer_from_user_to_pool_vault(...)?;
    }
}
```

Another option is to create an creator fees escrow PDA that creator can withdraw from.

Pie.fun: Acknowledged.

Zenith: Acknowledged by the client as the creator is trusted for V1.

[L-3] `stop_rebalancing()` can be DoS by with a WSOL transfer

Severity: Low	Status: Resolved

Context:

- stop_rebalancing.rs#L54-L57

Description: `stop_rebalancing()` has a check that ensures `wrapped_sol_balance < program_state.rebalance_margin_lamports`.

However, this makes it possible for a malicious user to transfer wrapped SOL to `vault_wrapped_sol` and cause this check to fail.

If the `rebalance_margin_lamports` value is very small (i.e. dust), the cost of doing so to DoS `stop_rebalancing()` is quite low and make it risky to have this check.

```
pub fn stop_rebalancing(ctx: Context<StopRebalancing>) -> Result<()> {
    let program_state = &mut ctx.accounts.program_state;
    let basket_config = &mut ctx.accounts.basket_config;
    require!(basket_config.is_rebalancing, PieError::NotInRebalancing);

    let wrapped_sol_balance = ctx.accounts.vault_wrapped_sol.amount;

    require!(
>>>     wrapped_sol_balance < program_state.rebalance_margin_lamports,
        PieError::InvalidMarginBottom
    );

    basket_config.is_rebalancing = false;

    emit!(StopRebalancingEvent {
        basket_id: ctx.accounts.basket_config.id,
        mint: ctx.accounts.basket_config.mint,
        components: ctx.accounts.basket_config.components.clone(),
        timestamp: Clock::get()?.unix_timestamp,
    });

    Ok(())
}
```

Recommendation: Consider removing the check if the `rebalancer` is trusted to rebalance the basket token composition to the benefits of the users.

This can also be acknowledged if admin ensures that `rebalance_margin_lamports` is not a dust amount.

Pie.fun: Fixed in PR-60

Zenith: Verified. Resolved by removing the check as WSOL is made into a basket component.

[L-4] `update_fee()` fails to ensure fee percentages are below `BASIS_POINTS`

Severity: Low   Status: Resolved

Context:

- update_fee.rs#L25-L46

Description: `update_fee()` is used to set the fee percentage for both creator and platform.

However, there is no check that ensures that the fee percentages do not exceed `BASIS_POINTS`. This will cause `buy_component()` and `sell_component()` to always fail due to insufficient amount for fee transfer.

```
pub fn update_fee(
    ctx: Context<UpdateFeeContext>,
    new_creator_fee_percentage: Option<u64>,
    new_platform_fee_percentage: Option<u64>,
) -> Result<()> {
    let program_state = &mut ctx.accounts.program_state;

    if let Some(new_creator_fee_percentage) = new_creator_fee_percentage
{
        program_state.creator_fee_percentage =
new_creator_fee_percentage;
    }

    if let Some(new_platform_fee_percentage) =
new_platform_fee_percentage {
        program_state.platform_fee_percentage =
new_platform_fee_percentage;
    }

    emit!(UpdateFeeEvent {
        new_creator_fee_percentage,
        new_platform_fee_percentage
    });

    Ok(())
}
```

Recommendation: Check that `creator_fee_percentage + platform_fee_percentage <= BASIS_POINTS` or set a reasonable limit.

Pie.fun: Fixed in the following commit

Zenith: Verified.

[L-5] `transfer_admin()` could cause admin role to be lost

Severity: Low	Status: Acknowledged

Context:

- transfer_admin.rs#L27

Description: `transfer_admin()` allows updating `program_state.admin` to transfer the admin role to the `new_admin`.

However, there is no validation for `new_admin`. If the admin role will set to a wrong address, the admin role could be lost or transferred to a wrong wallet.

```
pub fn transfer_admin(ctx: Context<TransferAdminContext>, new_admin:
Pubkey) -> Result<()> {
    let old_admin = ctx.accounts.program_state.admin;
>>> ctx.accounts.program_state.admin = new_admin;

    emit!(TransferAdminEvent {
        old_admin,
        new_admin,
    });

    Ok(())
}
```

Recommendation: Consider making `new_admin` a co-signer to ensure that the transfer is made to a valid `new_admin`.

```
pub struct TransferAdminContext<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

+    #[account(mut)]
+    pub new_admin: Signer<'info>,

```

Pie.fun: Acknowledged. Will be careful during transfer.

4.5 Informational

A total of 2 informational findings were identified.

[I-1] Redundant authorization check in update rebalancer instruction

Severity: Informational	**Status:** Resolved

Context:

- update_rebalancer.rs

Description: The update_rebalancer instruction contains a redundant authorization check. The same validation is performed in two places:

In the account validation through the UpdateRebalancerContext struct:

```
#[account(
    mut,
    seeds = [BASKET_CONFIG, &basket_config.id.to_be_bytes()],
    bump,
    constraint = basket_config.creator == creator.key() @
PieError::Unauthorized
)]
```

and In the instruction logic:

```
require!(
    ctx.accounts.creator.key() == ctx.accounts.basket_config.creator,
    PieError::Unauthorized
);
```

Recommendation: Remove the redundant check from the instruction logic since the constraint in the UpdateRebalancerContext already enforces this requirement.

Pie.fun: Fixed with the following commit

Zenith: Verified

[I-2] Missing constraint on `PROGRAM_STATE`

Severity: Informational	**Status:** Resolved

Context:

- buy_component.rs#L26-L27
- sell_component.rs#L26-L27

Description: Consider adding constraint to validate `program_state` for `buy_component` and `sell_component`, to ensure it is the actual `program_state`.

```
pub program_state: Box<Account<'info, ProgramState>>,
```

Recommendation:

```
-    #[account(mut)]
+    #[account(mut,
+    seeds = [PROGRAM_STATE],
+    bump = program_state.bump
+    )]
    pub program_state: Box<Account<'info, ProgramState>>,
```

Pie.fun: Fixed with the following commit

Zenith: Verified.