code4rena

4Real Finance

Smart Contract Security Assessment

Version 2.0

Audit dates: Oct 09 — Oct 17, 2024

Audited by: J4X
0xtheC0der

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 4Real Finance

4R vision is to become a decentralized treasury-backed protocol, with a community-owned set of inter-connected DeFi protocols, including staking vaults and a credit facility. As the roadmap expands, 4Real holders will vote for the most capital efficient use of the revenue generated, whether internally or externally.

2.2 Scope

Repository	4real-finance/contracts-audit
Commit Hash	34cf376be7325b5d755b46da1d7155cb78d6f043
Mitigation Hash	f0d636db3e1574339f3b5bbeab40758b07e9a154

2.3 Audit Timeline

DATE	EVENT
Oct 09, 2024	Audit start
Oct 17, 2024	Audit end
Oct 24, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	7
High Risk	8
Medium Risk	8
Low Risk	6
Informational	11
Total Issues	40

3. Findings Summary

ID	DESCRIPTION	STATUS
C-1	Worthless `user_underlying_token` can be provided on `stake` to mint valid LP tokens	Resolved

C-2	`unstaking_pool` can be drained by reinitializing the `unstaking_request`	Resolved
C-3	Insufficient distributor constraining can be abused to facilitate uncollateralized staking	Resolved
C-4	Users can mint arbitrary amounts of 4real as rewards	Resolved
C-5	Users can drain `sol-staking` vault by providing their own `underlying_vault`	Resolved
C-6	Missing epoch scaling on APR leads to distributing 1095x the intended rewards	Resolved
C-7	`distribute_epoch_rewards()` can be abused to mint infinite 4real tokens	Resolved
H-1	`inflection_point` is incorrectly used resulting in wrong APR	Resolved
H-2	Hardcoded `total_epoch` check in sigmoid computation leads to incorrect APR	Resolved
H-3	Users can circumvent fees by providing a custom `fees_vault` account	Resolved
H-4	Permissionless staking vault initialization allows for user-defined vault parameters	Resolved
H-5	Existing LP owners cannot be migrated as stakers by admin	Acknowledged
H-6	Incorrect static yield in sigmoid computation leads to incorrect APR	Resolved
H-7	Hardcoded boosted APR will result in incorrect calculation	Resolved
H-8	Referral will grant 400% instead of 4% apr boost	Resolved
M-1	Incorrect referral could be passed	Resolved
M-2	Fee is not charged on unstaking native sol	Resolved
M-3	Incorrect fees charged on token unstake	Resolved

M-4	Size of distributor is incorrectly set	Acknowledged
M-5	Wrong rounding on rewards will allow users to receive more than intended	Resolved
M-6	Self referral allows users to recoup half fee and get increased APR	Resolved
M-7	Vault pausing is broken	Resolved
M-8	Unstaking is not correctly batched	Resolved
L-1	Taxes could go above 100%, breaking the vault	Resolved
L-2	Division before multiplication leads to precision loss	Resolved
L-3	Missing check for `inflection_point==0` can block reward distribution	Acknowledged
L-4	`get_all_stake_positions()` will not return newest position in `sol-staking`	Acknowledged
L-5	Dust fee can be lost on referral splitting	Resolved
L-6	Taxes / Fees are rounded down allowing for a bypass	Resolved
I-1	Unused `underlying_mint` account	Resolved
I-2	Merging of duplicate files is recommended	Acknowledged
I-3	Checked operations should be used to prevent potential over/underflows	Acknowledged
I-4	`epoch_staked` is never used	Resolved
I-5	Typo in `new_stacked`	Resolved
I-6	`BPS` should be a constant	Resolved
I-7	Incorrectly named var `fee_after_tax`	Resolved
I-8	Missing admin transferrability	Resolved
I-9	Unused `rent` account	Resolved
I-10	Typo on `static_yeild` variable	Resolved

I-11	Unused errors	Acknowledged

4. Findings

4.1 Critical Risk

A total of 7 critical risk findings were identified.

[C-1] Worthless `user_underlying_token` can be provided on `stake` to mint valid LP tokens

Severity: Critical	Status: Resolved

Context:

- programs/staking/src/lib.rs#L645-L646
- programs/staking/src/lib.rs#L129-L13
- programs/staking/src/lib.rs#L161-L16

Description:

In the `staking` program's `stake` method & context, the mint of the `user_underlying_token` account is not constrained against the `vault.underlying_mint`. Therefore, a `user_underlying_token` with a different/worthless mint can be provided while the user still receives valid LP tokens.

Note that this requires that the `fees_vault` account and the `pool_token_account` have the same mint as the `user_underlying_token`. However, this precondition is given due to further lack of constraining as discussed in other findings.

Recommendation:

We recommend to constrain the mint of the `user_underlying_token` account against the `vault.underlying_mint`.

4Real:

- Staking: Fixed with the following commit
- Commit - 2

Zenith:

Verified

[C-2] `unstaking_pool` can be drained by reinitializing the `unstaking_request`

Severity: Critical	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L437-L444

Description:

When a user unstakes native sol, the process is split into 2 phases. First, the user's LP tokens are burnt, and he receives an `UnstakingRequest` structure.

```
// Create unstaking request
let unstaking_request = &mut ctx.accounts.unstaking_request;
unstaking_request.stake_id = position.stake_id;
unstaking_request.user = ctx.accounts.signer.key();
unstaking_request.amount = amount_after_tax;
unstaking_request.request_timestamp = Clock::get()?.unix_timestamp as
u64;
unstaking_request.vault = vault.key();
```

After some time, the user can retrieve his talked funds by calling `claim_unstaked.` The user's unstaking request account is removed at the end of this function.

```
// Close the unstaking request account
let dest_starting_lamports = ctx.accounts.user.lamports();
**ctx.accounts.user.lamports.borrow_mut() = dest_starting_lamports

.checked_add(ctx.accounts.unstaking_request.to_account_info().lamports())
    .unwrap();
**ctx.accounts.unstaking_request.to_account_info().lamports.borrow_mut()
= 0;

Ok(())
```

However, the contract needs to remember to zero out the request. As a result, it can be re-initialized and claimed again. By doing this, a user can consistently drain the `unstaking_pool`.

Recommendation:

We recommend zeroing out the `unstaking_request` before deleting it.

4Real:

The issue has been resolved with the following commit

Zenith:

Verified.

[C-3] Insufficient distributor constraining can be abused to facilitate uncollateralized staking

Severity: Critical	Status: Resolved

Context:

- programs/distributor/src/lib.rs#L166-L190
- programs/staking/src/lib.rs#L660-L670
- programs/staking/src/lib.rs#L696-L707
- programs/staking/src/lib.rs#L810-L820
- programs/sol-staking/src/lib.rs#L676-L681
- programs/sol-staking/src/lib.rs#L718-L729
- programs/sol-staking/src/lib.rs#L871-L881
- programs/staking/src/lib.rs#L161-L16

Description:

In the `Initialize` context and `initialize` method of the `distributor` program, the `admin` account (signer) is not constrained. Consequently, anyone can initialize a `distributor` account & `pool_token_account` on their behalf.

Furthermore, the staking programs does not constrain the `distributor` account's `admin` in any instance. Consequently, a custom `distributor` account & `pool_token_account` can be passed to the `stake`, `unstake` & `claim_rewards` methods.

In the `staking` program's `stake` method, the user's `amount_after_tax` is transferred to the `pool_token_account`, which is owned by the `distributor` account. In case the user has specified an own `distributor`, the tokens will still be in their possession after staking, i.e. they can effectively stake without collateral which allows them to arbitrarily mint LP tokens.

```
let cpi_accounts = ClassicTransfer {
    from: ctx.accounts.user_underlying_token.to_account_info(),
    to: ctx.accounts.pool_token_account.to_account_info(),
    authority: ctx.accounts.signer.to_account_info(),
};

let cpi_program = ctx.accounts.token_program.to_account_info();
let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
classic_transfer(cpi_ctx, amount_after_tax)?;
```

Recommendation:

1. We recommend to permission the `initialize` method of the `distributor` program, i.e. introduce a whitelist of accounts who can invoke this method.
2. We recommend to constraint the `distributor` account against `vault.distributor` in all instances, such that only the vault's designated `distributor` can be used.

4Real:

Fixed with: PR-2

SOL staking:

- Commit - 1
- Commit - 2

Staking:

- Commit - 1
- Commit - 2
- Commit - 3
- Commit - 4

Zenith:

Verified.

[C-4] Users can mint arbitrary amounts of 4real as rewards

Severity: Critical	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L325-L332

Description:

When a user stakes sol in the `sol-staking` contract, he receives a position account representing his staked position. Once the user unstakes his funds again, this account will be removed, and his LP tokens will be burned.

```
    // Close the position
    let rent = Rent::get()?;
    let lamports =
rent.minimum_balance(position.to_account_info().data_len());
    **position.to_account_info().try_borrow_mut_lamports()? = 0;
    **ctx.accounts.signer.to_account_info().try_borrow_mut_lamports()? +=
lamports;

    Ok(())
}
```

However, this implementation forgets to zero out the `position.stake_amount` variable. As a result, the user can reinitialize the account by transferring lamports there. While he will not be able to unstake the amount again (as his LP was burned), he will still be able to continue accruing rewards based on the `position.stake_amount`. As a result, the user can do this multiple times until he can generate arbitrary amounts of rewards each epoch.

Recommendation:

We recommend that you zero out all the information in the position before removing it.

4Real:

Addressed here: SOL staking: commit

Staking: commit

Zenith: Verified

[C-5] Users can drain `sol-staking` vault by providing their own `underlying_vault`

Severity: Critical	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L682-L683

Description:

A user can use the `stake()` function of the `sol-staking` implementation to stake native sol. In the transaction, he can provide the `underlying_vault` account as part of the context.

```
    #[account(mut)]
    pub underlying_vault: Box<Account<'info, TokenAccount>>,
```

This contract is then used to transfer the user SOL to when staking.

```
let cpi_accounts = ClassicTransfer {
    from: ctx.accounts.user_underlying_token.to_account_info(),
    to: ctx.accounts.underlying_vault.to_account_info(),
    authority: ctx.accounts.signer.to_account_info(),
};

let cpi_program = ctx.accounts.token_program.to_account_info();
let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
classic_transfer(cpi_ctx, amount_after_tax)?;
```

However, as we can see above, this account is not restricted. So, the user can provide another account he owns and receive a staking position without actually staking any tokens. The user can also unstake this position again and receive actual tokens from the pool, which allows him to drain the pool entirely.

Recommendation:

We recommend checking that the provided `underlying_vault` is equal to the `vault.underlying_vault` in the account constraints.

4Real:

The issue was mitigated in the following PR:

Fixed here: PR-4

Zenith: Verified

[C-6] Missing epoch scaling on APR leads to distributing 1095x the intended rewards

Severity: Critical	Status: Resolved

Context:

- programs/staking/src/lib.rs#L375
- programs/sol-staking/src/lib.rs#L354

Description:

The protocol allows a user to stake funds to earn rewards. These rewards will be distributed in the `distribute_epoch_rewards()` function. To calculate how many rewards a user should earn, the `get_user_dynamic_rewards()` function is called.

```
let rewards_apr = get_user_dynamic_rewards(epoch,
vault.sigmoid_params).div(100.0);
```

The value returned by this function will plateau at around `8`, so we are correctly scaling this to percent. However, afterward, we forget to also scale it based on the number of epochs in the year. Currently, we will give an 8% yield to the user with every epoch, while it should be 8% for the whole year. This leads to a user receiving, if the epoch is 8 hours, about 1000 times the rewards that he actually should.

Recommendation:

We recommend dividing the APR by the number of epochs within a year. So the calculation should be:

$$addedAPR = returnedAPR / (secondsInOneYear / epochLength)$$

4Real:

Fixed in the following PRs:

SOL staking: PR-4

Staking: PR-7

Zenith: Verified

[C-7] `distribute_epoch_rewards()` can be abused to mint infinite 4real tokens

Severity: Critical	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L334

Description:

The `distribute_epoch_rewards()` function on the `sol-staking/lib.rs` contract distributes the rewards. However, these rewards will be distributed in 4real, not in native sol. As a result, a current conversion rate is needed. This rate is passed as a parameter in the function header.

```
pub fn distribute_epoch_rewards(ctx: Context<DistributeRewards>,
conversion_rate: u64) -> Result<()> {
```

Later on, the actual amount fo 4real lp that the user will receive is calculated based on this conversion rate.

```
let rewards_in_token =
rewards_in_sol.checked_mul(conversion_rate).unwrap().checked_div(10_000).
unwrap();
msg!("Rewards in SOL: {}", rewards_in_sol);
msg!("Rewards in token: {}", rewards_in_token);
position.rewards_amount = position
    .rewards_amount
    .checked_add(rewards_in_token)
    .unwrap();
```

The problem here is that anyone can call this function. So, a user could pass a higher incorrect conversion rate here, allowing him to receive infinite amounts of 4real.

Recommendation:

We recommend restricting the `distribute_epoch_rewards()` function to being called only by the `operator`.

4Real:

Fixed in the following pull requests:

[SOL staking - PR-4]

[Staking - PR-7]

Zenith: Verified.

4.2 High Risk

A total of 8 high risk findings were identified.

[H-1] `inflection_point` is incorrectly used resulting in wrong APR

Severity: High	Status: Resolved

Context:

- programs/staking/src/sigmoid.rs#L3-L16
- programs/sol-staking/src/sigmoid.rs#L3-L16

Description:

The sigmoid function is used to calculate the APR a user will receive on his staking position.

However, the current calculation is incorrect, as the inflection point should be the point at which the function returns to x = 0.

Recommendation:

We recommend adapting the formula so that the inflection point is the value that is returned at epoch 0.

4Real:

Updated here: PR-3

Zenith:

Verified.

[H-2] Hardcoded `total_epoch` check in sigmoid computation leads to incorrect APR

Severity: High	Status: Resolved

Context:

- programs/staking/src/sigmoid.rs#L4
- programs/sol-staking/src/sigmoid.rs#L4

Description:

The sigmoid computation, which is used to determine the staking rewards APR, returns a static yield from epoch X onwards.

However, the hardcoded value of X is actually meant to be `sigmoid_params.total_epoch`. Therefore, the APR computations is incorrect in case of a configuration such that `sigmoid_params.total_epoch != X`.

Recommendation:

We recommend to use the `sigmoid_params.total_epoch` parameter:

```
if current_epoch > sigmoid_params.total_epoch {
    return sigmoid_params.static_yield;
}
```

4Real:

Fixed: PR-3

Zenith:

Verified

[H-3] Users can circumvent fees by providing a custom `fees_vault` account

Severity: High	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L673-L67
- programs/staking/src/lib.rs#L654-L655
- programs/staking/src/lib.rs#L693-L694

Description:

In the above instances, the `fees_vault` token account is not constrained. Therefore, a user can specify their own `fees_vault` on `stake`/`unstake` and siphon the fees on their own behalf.

Recommendation:

We recommend to constrain `fees_vault` against `vault.fees_vault`.

4Real:

- SOL staking
- SOL Staking - 2
- Staking
- Staking

Zenith:

Verified.

[H-4] Permissionless staking vault initialization allows for user-defined vault parameters

Severity: High	Status: Resolved

Context:

- programs/staking/src/lib.rs#L555-L584
- programs/sol-staking/src/lib.rs#L546-L588

###Description:

In the `InitStakingVault` context and `initialize` method, the `signer` account is not constrained. Consequently, anyone can initialize a staking vault for a given `base` & `underlying_mint` with custom parameters like `buy_tax`, `sell_tax`, `distributor`, `epoch_duration`, `unstaking_interval`, `fee` & `sigmoid_params`.

In the worst-case, the protocol owners' vault creation attempts can be front-run since there can only exist one staking vault for a given `base` &`underlying_mint`, and therefore fully breaking the program's intended use cases.

Recommendation:

We recommend to permission the `initialize` method, i.e. introduce a whitelist of accounts who can invoke this method.

4Real:

Fixed here:

- Sol staking
- Staking

Zenith:

Verified.

[H-5] Existing LP owners cannot be migrated as stakers by admin

Severity: High	Status: Acknowledged

Context:

- programs/staking/src/lib.rs#L527

Description:

The `staking` program's `admin_upload_stake_position` method intends migrate existing LP owners as stakers by creating a position for them.

However, due to the check

```
require!(stake_id < vault.positions_count, ErrorCode::InvalidStakeId);
```

only existing stake positions can be overwritten in this method instead of creating a new position as

```
vault.positions_count = vault.positions_count.checked_add(1).unwrap();
```

suggests.

Consequently, existing LP owners cannot be migrated as stakers and will therefore miss out on staking rewards.

Recommendation:

We recommend to remove the `stake_id` parameter & `require` check, and create a new position similar to the `stake` method:

```
position.stake_id = vault.positions_count;
```

4Real:

Won't fix.

Zenith:

Acknowledged.

[H-6] Incorrect static yield in sigmoid computation leads to incorrect APR

Severity: High	Status: Resolved

Context:

- programs/sol-staking/src/sigmoid.rs#L4-L6

Description:

The sigmoid computation, which is used to determine the `sol-staking` rewards APR, intends to return a static yield from epoch 91 onwards. The static yield is determined by the function's `sigmoid_params.static_yield` parameter. However, a hardcoded value is used instead which leads to an incorrect APR in the long-term.

Recommendation:

We recommend the use the `sigmoid_params.static_yield` parameter.

4Real:

Fixed with the following commit

Zenith:

Verified

[H-7] Hardcoded boosted APR will result in incorrect calculation

Severity: High	Status: Resolved

Context:

- programs/staking/src/lib.rs#L378

Description:

In the rewards distribution process, users who were referred to the protocol will be granted additional APR. The APR they receive is hardcoded and will be added for each epoch.

```
if position.referral != Pubkey::default() {
    // 4% / 1095 (epochs in year)
    rewards_apr = rewards_apr.add(0.003652968037);
}
```

However, the calculation of `4% / 1095` will only be accurate if the period duration is 8 hours. The period duration can be set to any value in the initializer; the added apr should be calculated based on the set period duration and not hardcoded.

Recommendation:

We recommend not hardcoding the boosted APR. Instead, it should be calculated based on the epoch duration set in the initializer.

4Real:

Removed in the following PR

Zenith:

Verified

[H-8] Referral will grant 400% instead of 4% apr boost

Severity: High	Status: Resolved

Context:

- programs/staking/src/lib.rs#L378

Description:

If a user gets referred on his staking he should receive an additional APR.

Unfortunately, the calculation is wrong because `0.003652968037` results from `4/1095`. We want to calculate the percentage so that the correct calculation is `0.04/1095`.

As a result of the incorrect hardcoded value, the user will receive a 400% APR boost instead of a 4% one.

Recommendation:

We recommend (if the value should stay hardcode) to change it to `0.00003652968037`.

4Real:

Removed in the following PR

Zenith:

Verified

4.3 Medium Risk

A total of 8 medium risk findings were identified.

[M-1] Incorrect referral could be passed

Severity: Medium	Status: Resolved

Context:

- programs/staking/src/lib.rs#L98

Description:

When users stake in the `staking` contract, they can provide 2 parameters related to someone referring them. First, they can pass the `referral` param in the function parameters.

```
pub fn stake(ctx: Context<Stake>, amount_in: u64, min_lp_amount_out: u64,
referral: Option<Pubkey>) -> Result<()> {
```

This param will be added to their staking struct. Additionally, in the context, they can add the `referral_underlying_token`.

```
#[account(mut)]
pub referral_underlying_token: Option<Account<'info, TokenAccount>>,
```

This account is where the referral fee will be transferred. However, we never check that this token account is owned by the `referral` pubkey. As a result, even if we would check referral against a whitelist (other issue), we would still run into the issue that a different token account could be provided

Recommendation:

We recommend verifying that `referral` is the owner of the `referral_underlying_token` account.

4Real:

- Sol staking
- staking

Added here:

- sol-staking
- staking

Zenith:

Verified.

[M-2] Fee is not charged on unstaking native sol

Severity: Medium	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L244

Description:

The 4real protocol implements fees for staking and unstaking of tokens in both the `staking` and `sol-staking` contracts. This is also described in the comments on the `sol-staking` contract.

```
// 2. Unstake
// 2.1 User burns 0.95 frSOL (2nd tax)
// 2.2 User request enters cooldown period (up to 3 days)
// 2.3 Contract saves information about how much all users have unstaked
// 2.3 Admin should top up unstaking pool before the end of cooldown
period using method that uses saved information
// 2.4 When cooldown period is over, user can claim his principal (0.90
SOL) - taxed another time.
// 2.3 another 80% of 0.05 SOL (Tax amount) is transferred to treasury
while claiming
// 2.4 20% of 0.05 SOL (Tax amount) is transferred to feesVault
```

Unfortunately, this was not implemented for the unstaking of tokens.

```
// Now proceed with the unstaking process
let sell_tax_amount = position.stake_amount
    .checked_mul(vault.sell_tax)
    .unwrap()
    .checked_div(10000)
    .unwrap();

let amount_after_tax =
position.stake_amount.checked_sub(sell_tax_amount).unwrap();
```

As one can see in the snippet above, the tax is charged, but a fee is never deducted.

Recommendation:

We recommend adding functionality to take a percentage of the `sell_tax` as a fee.

4Real:

Fixed in the following PR

Zenith:

Verified

[M-3] Incorrect fees charged on token unstake

Severity: Medium	Status: Resolved

Context:

- programs/staking/src/lib.rs#L306-L310

Description:

Throughout the staking and unstaking, fees are charged on the amounts. These are usually based on the `vault.fee` set by the admin.

```
let fee_amount = sell_tax_amount
    .checked_div(100)
    .unwrap()
    .checked_mul(20)
    .unwrap();
```

However, as one can see in the snippet above, the fee is not used on unstaking tokens; instead, it is hardcoded.

Recommendation:

We would recommend that you use the fee set by the admin here.

```
let fee_amount = sell_tax_amount
    .checked_div(100)
    .unwrap()
    .checked_mul(vault.fee)
    .unwrap();
```

4Real:

The issue was mitigated in the following PRs:

- SOL staking
- Staking

C4 Zenith:

Verified.

[M-4] Size of distributor is incorrectly set

Severity: Medium	Status: Acknowledged

Context:

- programs/distributor/src/lib.rs#L263

Description:

The `distributor` is used to distribute 4real tokens.

```
#[account]
pub struct Distributor {
    pub admin: Pubkey,
    pub mintable_token_mint: Pubkey,
    pub pool_token_mint: Pubkey,
    pub authorized_addresses: Vec<Pubkey>,
}

impl Distributor {
    pub const LEN: usize = 8 + 32 + 32 + 8 * 32 + 8;

    pub fn is_authorized(&self, address: &Pubkey) -> bool {
        self.authorized_addresses.contains(address)
    }
}
```

However, a problem occurs as the `distributor` struct includes a vector of arbitrary size. As a result, the size of the account can be indefinitely big, but it is initialized so that only a size of 8 is possible.

Recommendation:

We recommend making the vector an array of a fixed reasonable size and calculating the total `LEN` based on that.

4Real:

Fixed with the following commit

Zenith:

The Partial fix was verified and 4Real acknowledged the rest.

[M-5] Wrong rounding on rewards will allow users to receive more than intended

Severity: Medium	Status: Resolved

Context:

- programs/staking/src/lib.rs#L383

Description:

When rewards are distributed, this is done by multiplying the calculated APR by the number of staked tokens.

```
let rewards_in_token = (amount_for_rewards as f64 * rewards_apr).round()
as u64;
```

As `round()` is used, the value will round to the closest integer. So in 50% of cases, or if a user intentionally deposits a reward that will lead to positive rounding, the user will receive more rewards than intended.

Recommendation:

We recommend always rounding down in this case so the user can not receive more rewards than intended.

4Real:

Removed here

- SOL staking
- Staking

Zenith:

Verified.

[M-6] Self referral allows users to recoup half fee and get increased APR

Severity: Medium	Status: Resolved

Context:

- programs/staking/src/lib.rs#L137-L150

Description:

The `staking` branch includes the possibility of referrals on stakes. So, every user can provide an optional `referral` address.

```
pub fn stake(ctx: Context<Stake>, amount_in: u64, min_lp_amount_out: u64,
referral: Option<Pubkey>) -> Result<()> {
```

If an address is provided, half of the fee taken on the deposit will be sent to the referrer.

```
if let Some(referral_underlying_token) = referral_underlying_token {
    classic_transfer(cpi_ctx, fee_amount.checked_div(2).unwrap())?;

    let cpi_accounts = ClassicTransfer {
        from: ctx.accounts.user_underlying_token.to_account_info(),
        to: referral_underlying_token.to_account_info(),
        authority: ctx.accounts.signer.to_account_info(),
    };
    let cpi_program = ctx.accounts.token_program.to_account_info();
    let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
    let ref_fee = fee_amount.checked_div(2).unwrap();
    msg!("Referral gets {}", ref_fee);
    classic_transfer(cpi_ctx, ref_fee)?;
} else {
    classic_transfer(cpi_ctx, fee_amount)?;
}
```

Additionally, each staking that was referred will receive an additional APR.

However, nothing stops a user from just setting himself as the referrer. In that case, he would get an increased APR and would be able to recoup half of the fee.

Recommendation:

We recommend implementing a whitelist for referrers. This role can then be granted to affiliates. On referral it should be checked that the referrer is one of the whitelisters.

4Real:

Addressed self-referral

- Sol staking
- Staking

Zenith:

Verified the partial mitigation with the fix.

[M-7] Vault pausing is broken

Severity: Medium	Status: Resolved

Context:

- programs/staking/src/lib.rs#L72
- programs/sol-staking/src/lib.rs#L96

Description:

The vault includes the `enabled` parameter, which, if set to false, should be used to pause the vault.

```
pub struct Vault {
    /// The flag, if admin set enable = false, then the user can only
withdraw and cannot deposit in the vault.
    pub enabled: u8,
```

When the vault gets initialized, the value is directly set to true.

```
pub fn initialize(
    ctx: Context<InitStakingVault>,
    base: Pubkey,
    bumps: VaultBumps,
    buy_tax: u64,
    sell_tax: u64,
    distributor: Pubkey,
    epoch_duration: u16,
    unstaking_interval: u64,
    fee: u16,
    sigmoid_params: SigmoidParams,
) -> Result<()> {
    // Other setters

    vault.enabled = 1; // <= Here

    // Other setters

    Ok(())
}
```

Afterwards, the value can be changed via the `set_vault_enabled()` function.

```
pub fn set_vault_enabled(ctx: Context<SetVaultEnabled>, enabled: u8) ->
Result<()> {
    let vault = &mut ctx.accounts.vault;
    vault.enabled = enabled;
    Ok(())
}
```

However, the problem is that this value is never checked anywhere. So, even if it is set to false, every entry point can still be called.

Recommendation:

We recommend checking if `vault.enabled` is set to true on the `deposit()` and reverting otherwise.

4Real:

Fixed in the following PRs:

Sol staking: Stake Rewards

FR staking:

Stake Rewards

Zenith:

Verified.

[M-8] Unstaking is not correctly batched

Severity: Medium	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L408-L417

Description:

The `sol-staking` branch allows users to stake funds. When a user wants to unstake these funds, a two step process takes place.

1. User unstakes via `unstake()` function
2. User claims stake via `claim_unstaked()` function once some time has passed.

The intention here is that the unstakings queue up, and at the end of each period, all can be claimed. Some functionality related to that is also implemented (`unstaking_period_start` and `unstaking_period_end`), but these are never used.

```
// Check if the current period has ended
if current_time >= vault.unstaking_period_end {
    vault.unstaking_period_start = vault.unstaking_period_end;
    vault.unstaking_period_end =
vault.unstaking_period_start.checked_add(vault.unstaking_interval).unwrap
();
}

require!(

unstaking_request.request_timestamp.checked_add(vault.unstaking_interval)
.unwrap() <= current_time,
    ErrorCode::UnstakingPeriodNotEnded
);
```

As one can see, each unstaking individual has to wait for the interval instead of being batched.

Recommendation:

We recommend adapting the unstaking process so that unstakings save the period that they were unstaked in and can be unstaked once this period has ended.

4Real:

Fixed with the following commit

Zenith:

Verified.

4.4 Low Risk

A total of 6 low risk findings were identified.

[L-1] Taxes could go above 100%, breaking the vault

Severity: Low	Status: Resolved

Context:

- programs/staking/src/lib.rs#L59-L60
- programs/sol-staking/src/lib.rs#L74-L75

Description:

In both contracts, the user provides the `buy_tax` and `sell_tax` in the `initialize()` function.

```
pub fn initialize(
    ctx: Context<InitStakingVault>,
    base: Pubkey,
    bumps: VaultBumps,
    buy_tax: u64,
    sell_tax: u64,
    distributor: Pubkey,
    epoch_duration: u16,
    unstaking_interval: u64,
    fee: u16,
    sigmoid_params: SigmoidParams,
) -> Result<()> {
    let vault = &mut ctx.accounts.vault;
    vault.base = base;
    vault.buy_tax = buy_tax;
    vault.sell_tax = sell_tax;
```

However they are never restricted to only up to 100% or 10,000 BPS. As a result, if they would be higher, the contract would break as it would underflow when trying to subtract more than 100% of the amounts.

Recommendation:

We recommend checking that the taxes can stay under 10,000 BPS.

4Real:

SOL staking:

- Init
- Setter

Staking:

- Init
- setter

Zenith:

Verified

[L-2] Division before multiplication leads to precision loss

Severity: Low	Status: Resolved

Context:

- programs/staking/src/lib.rs#L120-L124
- programs/staking/src/lib.rs#L306-L310
- programs/sol-staking/src/lib.rs#L163-L167

Description:

During the computation of `fee_amount` in the above instances, a division by 100 is performed before multiplication with the fee percentage. This can lead to a precision loss.

```
let fee_amount = buy_tax_amount
    .checked_div(100)
    .unwrap()
    .checked_mul(vault.fee.into())
    .unwrap();
```

Recommendation:

We recommend to always perform multiplications before divisions.

4Real:

- SOL staking

Staking:

- Commit - 1
- Comit -2

Zenith:

Verified.

[L-3] Missing check for `inflection_point==0` can block reward distribution

Severity: Low	Status: Acknowledged

Context:

- programs/sol-staking/src/sigmoid.rs#L3-L16 -programs/staking/src/sigmoid.rs#L3-L16

Description:

The `get_user_dynamic_rewards()` function calculates a user's rewards apr for a given epoch.

The parameters used for this are set when initializing the vault. However, we never check if `inflection_point==0`. This can lead to issues as we divide by `inflection_point` during the calculation. If `inflection_point` were accidentally set to 0, the function would break, and rewards could not be distributed.

Recommendation:

We recommend adding a check to the vaults' initialization that verifies that `inflection_point != 0`.

4Real:

Acknowledged.

[L-4] `get_all_stake_positions()` will not return newest position in `sol-staking`

Severity: Low	Status: Acknowledged

Context:

- programs/sol-staking/src/lib.rs#L498

Description:

When a user stakes some funds, his staking position will be tracked using a `StakePosition` account.

```
#[account(
    init,
    payer = signer,
    space = 8 + std::mem::size_of::<StakePosition>(),
    seeds = [
        b"stake_position",
        vault.key().as_ref(),
        &vault.positions_count.checked_add(1).unwrap().to_le_bytes(),
    ],
    bump
)]
pub stake_position: Account<'info, StakePosition>,
```

As we can see, this is seeded with a string, the vault key, and the current position count + 1.

When a user wants to retrieve the addresses of all positions, he can call `get_all_stake_positions()`.

```
pub fn get_all_stake_positions(ctx: Context<GetAllStakePositions>) ->
Result<()> {
    let vault = &ctx.accounts.vault;

    for i in 0..vault.positions_count {
        let vault_key = vault.key();
        let position_seeds = &[b"stake_position", vault_key.as_ref(),
&i.to_le_bytes()];
        let (position_address, _) =
            Pubkey::find_program_address(position_seeds, ctx.program_id);

        msg!("Stake Position {}: {}", i, position_address);
```

```
        }

        Ok(())
    }
```

However, as you can see, we will only loop up to `vault.positions_count` with the seeds, so the newest generated `StakingPosition` will not be included here.

Recommendation:

As there is no reason not to start at 0, we recommend adapting the seeding/setting of the stake ID of the positions to use the current count.

4Real:

This function is going to be removed.

Zenith:

Acknowledged.

[L-5] Dust fee can be lost on referral splitting

Severity: Low	Status: Resolved

Context:

- programs/staking/src/lib.rs#L137-L150

Description:

When a user was referred to stake, the fee taken from that user will be split into half and one half will go to the protocol, while the other half will go to the referrer.

```
classic_transfer(cpi_ctx, fee_amount.checked_div(2).unwrap())?;

                let cpi_accounts = ClassicTransfer {
                    from:
ctx.accounts.user_underlying_token.to_account_info(),
                    to: referral_underlying_token.to_account_info(),
                    authority: ctx.accounts.signer.to_account_info(),
                };
                let cpi_program =
ctx.accounts.token_program.to_account_info();
                let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
                let ref_fee = fee_amount.checked_div(2).unwrap();
```

However, the problem is that if we divide by 2 in both cases and if the `fee_amount` is an uneven number, one token will be lost and become part of the tax again.

Recommendation: We recommend calculating the `ref_fee` as follows.

```
let ref_fee =
fee_amount.checked_sub(fee_amount.checked_div(2).unwrap()).unwrap();
```

4Real:

- SOL staking
- Staking

Zenith: Verified.

[L-6] Taxes / Fees are rounded down allowing for a bypass

Severity: Low	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L147-L151
- programs/sol-staking/src/lib.rs#L163-L167
- programs/sol-staking/src/lib.rs#L288-L292
- programs/staking/src/lib.rs#L104-L108
- programs/staking/src/lib.rs#L120-L124
- programs/staking/src/lib.rs#L271-L275

Description:

During the staking / unstaking process, the taxes and fees charged to the user are calculated. These calculations are all pretty similar; we can take the one for the `buy_tax` as an example.

```
let buy_tax_amount = amount_in
    .checked_mul(vault.buy_tax)
    .unwrap()
    .checked_div(10000)
    .unwrap();
```

The problem here is that the division will always round down. So a user could, if the `buy_tax` is one basis point, deposit 9999 tokens and would pay 0% tax due to the rounding.

Recommendation:

We recommend rounding up in these calculations to make bypassing the taxes/fees impossible.

4Real:

Fixed in the following PRs:

SOL staking:

- Commit - 1
- Commit - 2
- Commit - 3
- Commit- 4
- Commit -5

- Commit - 6
- Commit - 7

Staking:

- Commit - 1
- Commit - 2
- Commit - 3
- Commit - 4
- Commit - 5
- Commit - 6
- Commit - 7
- Commit - 8

Zenith:

Verified.

4.5 Informational

A total of 11 informational findings were identified.

[I-1] Unused `underlying_mint` account

Severity: Informational	Status: Resolved

Context:

- programs/staking/src/lib.rs#L643
- programs/sol-staking/src/lib.rs#L662

Description:

The `underlying_mint` account, defined in the above instances, is never used in the corresponding methods. Instead the `vault.underlying_mint` is used which is the correct approach.

Recommendation:

We recommend to remove the `underlying_mint` definitions from the contexts in the given instances.

4Real:

- SOL staking
- Staking

Zenith:

Verified.

[I-2] Merging of duplicate files is recommended

Severity: Informational	Status: Acknowledged

Context:

- programs/sol-staking/src/epoch.rs
- programs/sol-staking/src/errors.rs
- programs/sol-staking/src/sigmoid.rs
- programs/staking/src/epoch.rs
- programs/staking/src/errors.rs
- programs/staking/src/sigmoid.rs

Description:

Throughout the codebase, the three files `errors.rs`, `epoch.rs`, and `sigmoid.rs` are identical. However, they are duplicates in both the `staking` and `sol-staking` folders. As shown by some other issues, this has already led to problems due to bugs only being fixed on one of both versions.

Recommendation:

We recommend merging the duplicate files in a separate folder and linking them there from both the `staking` and the `sol-staking` contracts.

4Real:

Not gonna implement on our side.

Zenith:

Acknowledged.

[I-3] Checked operations should be used to prevent potential over/underflows

Severity: Informational	Status: Acknowledged

Context:

- programs/sol-staking/src/lib.rs#L340
- programs/sol-staking/src/lib.rs#L347
- programs/sol-staking/src/lib.rs#L351
- programs/staking/src/lib.rs#L361
- programs/staking/src/lib.rs#L368
- programs/staking/src/lib.rs#L372

Description:

Multiple additions/subtractions don't use checked mathematical operations throughout the codebase. Rust has no built-in overflow protection, which could result in either overflows or underflows.

Recommendation:

We recommend using checked addition/subtraction on each of these cases.

4Real:

Acknowledged.

[I-4] `epoch_staked` is never used

Severity: Informational	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L233
- programs/staking/src/lib.rs#L208

Description:

The `position` struct tracks user staking positions. It includes the `epoch_staked` variable.

```
pub struct StakePosition {
    pub stake_id: u32,
    pub staked_at: u64,
    pub epoch_staked: u16, // <= Here
    pub epoch_at_last_update: u16,

    pub user: Pubkey,
    pub stake_amount: u64,
    pub rewards_amount: u64,

    pub vault: Pubkey,
}
```

The variable tracks the epoch in which a position was staked. However, it is only set to 0 on the initialization of the position and is never used afterward.

Recommendation:

We recommend either using the variable or removing it.

4Real:

Removed:

- SOL Staking
- Staking

Zenith:

Verified.

[I-5] Typo in `new_stacked`

Severity: Informational	Status: Resolved

Context:

- programs/staking/src/lib.rs#L422

Description:

A variable in the `distribute_epoch_rewards()` function should represent the new amount of staked tokens. However, this variable is called `new_staked`, which is incorrect.

```
let new_stacked =
vault.total_staked.checked_add(rewards_to_distribute).unwrap();
vault.total_staked = new_stacked;
```

Recommendation:

We recommend renaming the variable to `new_staked`.

4Real:

Fixed in the following PRs:

- Solana staking
- Staking

Zenith:

Verified.

[I-6] `BPS` should be a constant

Severity: Informational	Status: Resolved

Context:

- programs/sol-staking/src/lib.rs#L150
- programs/sol-staking/src/lib.rs#L291
- programs/sol-staking/src/lib.rs#L513-L515
- programs/sol-staking/src/lib.rs#L527-L529
- programs/staking/src/lib.rs#L107
- programs/staking/src/lib.rs#L274
- programs/staking/src/lib.rs#L495-L497
- programs/staking/src/lib.rs#L509-L511

Description:

Throughout the codebase, calculations are often done using basis points (1BPS = 10,000). Unfortunately, the value is always hardcoded.

Recommendation:

We recommend defining a constant for BPS.

4Real:

The issue was mitigated in the following PRs:

- SOL staking
- Staking

Zenith:

Verified.

[I-7] Incorrectly named var `fee_after_tax`

Severity: Informational	Status: Resolved

Context:

- programs/staking/src/lib.rs#L154
- programs/sol-staking/src/lib.rs#L179

Description:

The vault contracts invoke a tax on deposits. After that, a fee is invoked on the tax.

```
let fee_after_tax = buy_tax_amount.checked_sub(fee_amount).unwrap();
msg!("Tax share of distributor {}", fee_after_tax);
```

However, this var is incorrectly named as the tax without the fee.

Recommendation:

We recommend renaming the variable to `tax_after_fee`.

4Real:

Mitigated in the following PRs:

- SOL Staking
- Staking

Zenith:

Verified

[I-8] Missing admin transferrability

Severity: Informational	Status: Resolved

Context:

- /sol-staking/src/lib.rs#L54
- programs/staking/src/lib.rs#L40

Description:

The vault contracts include an `admin` variable. This address allows access to admin functionality and changes other contract variables. However, there is no way to transfer the ownership of this variable.

Recommendation:'

We recommend ading functionality that allows transferring the admin.

4Real:

Issue mitigated in the following PRs:

- SOL staking
- Staking

Zenith:

Verified.

[I-9] Unused `rent` account

Severity: Informational	Status: Resolved

Context:

- programs/distributor/src/lib.rs#L189

Description:

The `Initialize` context in the `distributor/src/lib.rs` file includes the `rent` account which is never used anywhere.

```
#[derive(Accounts)]
pub struct Initialize<'info> {
    #[account(init,
         payer = admin,
         space = 8 + Distributor::LEN,
         seeds = [b"distributor", admin.key().as_ref()],
         bump)]
    pub distributor: Box<Account<'info, Distributor>>,
    #[account(mut)]
    pub admin: Signer<'info>,
    pub system_program: Program<'info, System>,
    #[account(
        init,
        payer = admin,
        seeds = [b"pool_token_account", pool_token_mint.key().as_ref()],
        bump,
        token::mint = pool_token_mint,
        token::authority = distributor,
    )]
    pub pool_token_account: Box<Account<'info, TokenAccount>>,
    pub pool_token_mint: Box<Account<'info, Mint>>,
    pub mintable_token_mint: Box<InterfaceAccount<'info, MintInterface>>,
    pub token_program: Program<'info, Token>,
    pub rent: Sysvar<'info, Rent>, //<= This account
}
```

Recommendation:

We would recommend that you remove the unused account.

4Real:

Fixed with the following PR-2

Zenith:

Verified.

[I-10] Typo on `static_yeild` variable

Severity: Informational	Status: Resolved

Context:

- programs/staking/src/state/vault.rs#L51
- programs/sol-staking/src/state/vault.rs#L71

Description:

The `vault.rs` file includes the `SigmoidParams` struct. In this one of the values includes a typo.

```
pub static_yeild: f64,
```

###. Recommendation:

We recommend renaming the variable to `static_yield`.

4Real:

Fixed in the following:

- Commit - 1
- Commit - 2
- Commit - 3
- Commit - 4

Zenith:

Verified.

[I-11] Unused errors

Severity: Informational	Status: Acknowledged

[I-01] Unused errors

Context:

- programs/staking/src/errors.rs#L4-L26
- programs/sol-staking/src/errors.rs#L4-L26

Description:

Multiple errors in the errors.rs file, found in both branches, are either not used at all or only used in one of both branches.

- VaultNotEnabled => Not used at all
- InsufficientFunds => Not used at all
- InsufficientStakeAmount => Not used at all
- InsufficientTaxAmount => Not used at all
- InvalidStakeId => Not used in sol-staking branch
- UnstakingPeriodNotEnded => Not used in staking branch
- InsufficientTopUpAmount => Not used at all

Recommendation:

We would recommend that you remove the unused errors.

4Real:

Not going to focus on this in terms of the audit - thanks for the recommendation, we'll refactor all the extra variables internally.

Zenith:

Acknowledged

5. Centralization / Systemic Risks

Unrestricted admin access to user funds

Once users stake funds in one of the vaults, they are transferred to an underlying token account, from which they will be used to generate yield using multiple strategies. This token account is controlled by the admin/protocol, which transfers the tokens to generate yield. This setup introduces a significant centralization risk. If an admin were to act maliciously, they could steal all staked funds without facing any consequences. This means that once a user has staked funds, they are entirely dependent on the admin's goodwill for their return.

Admin could delete/change any position

The admin_upload_stake_position() function was implemented to allow an admin to add new positions for users who have already bought LPs in the presale. However, this function does not work as intended and only allows the admin to overwrite existing positions. Even if the admin_upload_stake_position () function were to function as intended, it would still pose a significant centralization risk. This is because it allows the admin to alter staking positions at their discretion, potentially leading to scenarios where a malicious admin could nullify user positions or inflate their own.