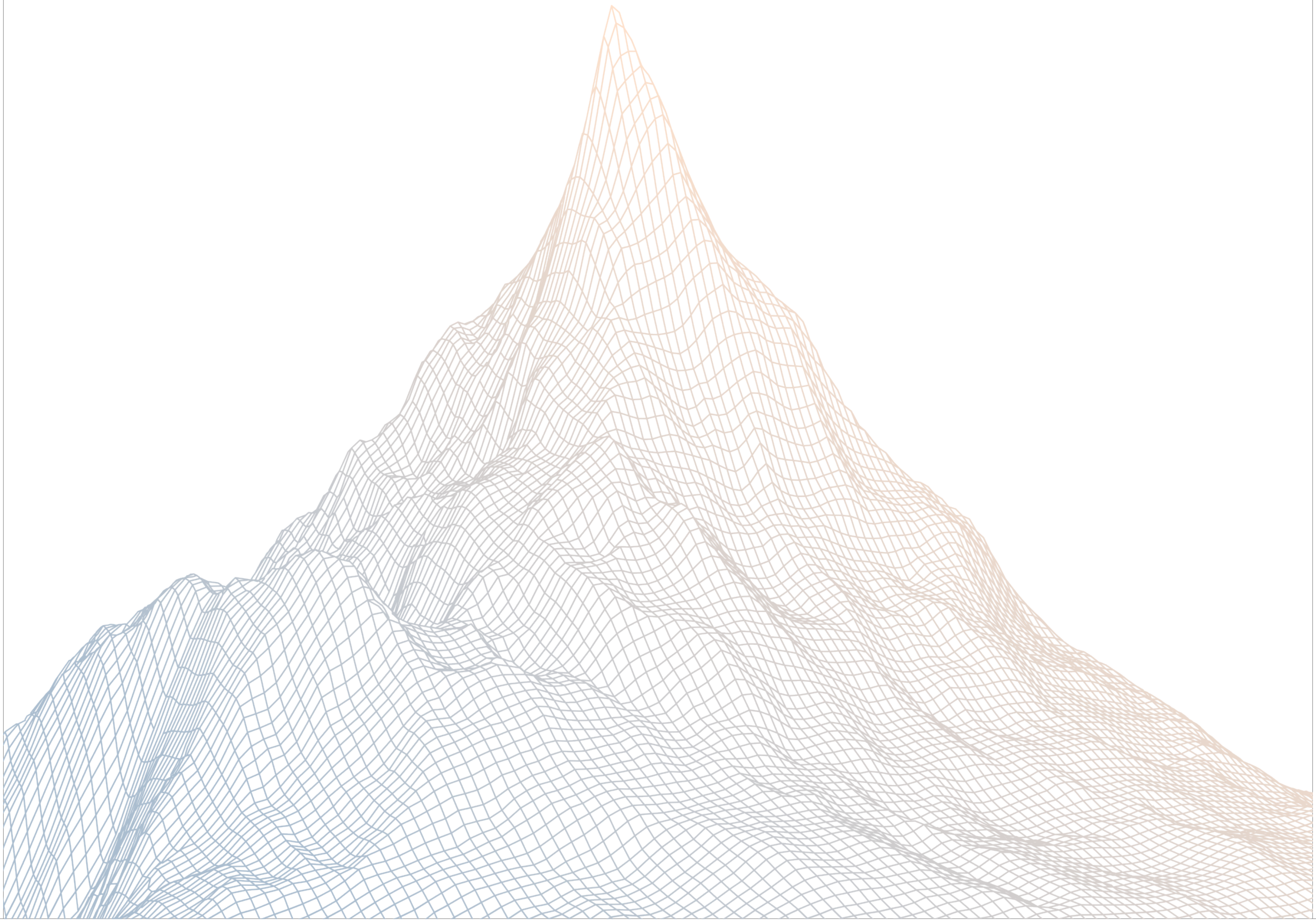
Zenith

Dinero

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	May 25nd to June 6th, 2025
AUDITED BY:	Matte peakbolt shaflow2

Contents

1 Introduction

1.1 About Zenith

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

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

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

1.2 Disclaimer

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

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

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

1.3 Risk Classification

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

2 Executive Summary

2.1 About Dinero Protocol

Dinero is a suite of products that scale yield for protocols and users. It includes: (i) an ETH liquid staking token (pxETH) which benefits from ETH staking yield from Dinero's validators; (ii) a decentralized, collateral-backed stablecoin (pxUSD) as a medium of exchange on Ethereum; and (iii) a public and permissionless RPC for users.

2.2 Scope

The engagement involved a review of the following targets:

Target	dinero-solana
Repository	https://github.com/Meridian-Labss/dinero-solana
Commit Hash	`20e7dc6056630be2210cdc9ec1674464aaf14f16`
Files	`programs/main-program/*`

2.3 Audit Timeline

May 25th, 2025	Audit start
June 6th, 2025	Audit end
June 19th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	2
Medium Risk	10
Low Risk	4
Informational	3
Total Issues	**20**

3 Findings Summary

ID	Description	Status
C-1	Missing validation for institution account allows mixing of retail and institutional funds	Resolved
H-1	Temporary stake account balances and rent are not accounted in exchange rate update	Resolved
H-2	Force-deactivated stakes can deny withdrawals and stake management	Resolved
M-1	Institutional instructions can be blocked when global whitelist is disabled	Resolved
M-2	Allowing reinitialization of entity_validator_state would compromise the staking state.	Resolved
M-3	When the APX supply is small, the APX/UP exchange rate may increase rapidly.	Resolved
M-4	The split_and_deactivate_temporary_stake and deactivate_persistent_stake instructions lacks stake check.	Resolved
M-5	Claim order may be unfair	Resolved
M-6	w_calculation could result in a zero exchange rate	Acknowledged
M-7	Slashing could prevent update of vault exchange rate	Acknowledged
M-8	Admin can deactivate and withdraw more stakes than required	Acknowledged
M-9	Admin cannot fully deactivate and withdraw stakes from rogue validators	Resolved
M-10	IPX_SOL holder can redeem and claim from any institution's token buffer account	Resolved
L-1	Interactions involving small amounts of funds are not subject to any fees.	Resolved

ID	Description	Status
L-2	The initialization of the program is vulnerable to a front-run attack.	Resolved
L-3	initialize_retail() and initialize_institutional() can be called again to change exchange rate	Resolved
L-4	Double rent calculation in pre_epoch_split_and_deactivate_temporary_stake	Resolved
I-1	buffer_limit cannot be modified	Resolved
I-2	The funds may be released to the wrong account	Resolved
I-3	Code improvements	Acknowledged

4 Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Missing validation for `institution` account allows mixing of retail and institutional funds

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

Target

- deposit.rs#L21
- deposit.rs#L104
- delay_claim.rs#L17
- delay_claim.rs#L86
- fast_redemption.rs#L15
- slow_redemption.rs#L17
- slow_redemption.rs#L101
- vault_exit_institutional.rs#L44

Description:

The protocol is designed to isolate retail (PX_SOL) funds from the institutional (IPX_SOL) funds. This is differentiated by the `institution` account where retail uses the current program account.

However, there are no validations for the `institution` account for the retail and institution instructions. This allows one to mix retail and institutional funds. For example, a user can mint PX_SOL but yet specify a non-retail `institution` account to deposit the SOL.

```
pub struct RetailDeposit<'info> {
    // Transfer SOLs
    #[account(mut)]
    pub payer: Signer<'info>,
    #[account(
        mut,
        seeds = [EntityState::BUFFER_ACCOUNT_SEED,
    institution.key().as_ref()],
        bump
    )]
    pub buffer_account: SystemAccount<'info>,
    /// CHECK: Used for seed generation
    pub institution: UncheckedAccount<'info>,
```

Recommendations:

For the retail instructions, add an anchor constraint to ensure that the `institution` account matches the current program account.

For the institutional instructions, ensure that the `institution` account is not equal to the current program account.

Dinero: Fixed in @05e8eec...

Zenith: Resolved by deriving institution address from signer for institutional instructions. For retail instruction, it is hardcoded to Program ID.

4.2 High Risk

A total of 2 high risk findings were identified.

[H-1] Temporary stake account balances and rent are not accounted in exchange rate update

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

Target

- utils.rs#L193-L220

Description:

Both `evaluate_apx_sol_valuation()` and `evaluate_iapx_sol_valuation()` requires the admin-cli to provide the `stake_account_balance`. This is calculated in `get_entity_balances()`, which obtains all the SOL balance in the different stake accounts.

However, there are two issues in the `get_entity_balances()`,

1. It only retrieves the persistent stake account balances and fails to include the temporary stake accounts. This will cause the `stake_account_balance` to be incorrect as it is expected that some of the SOL will be in temporary stake accounts when they are in the process of merging or withdrawal from the persistent stake accounts, which occurs over multiple epochs.

2. It does not exclude the rent from the balance for `persistent_stake_accounts`, causing the stake account balance to be slightly higher than expected due to the rent. That is because rent for `persistent_stake_accounts` are paid by program owner. On the other hand, rent for `temporary_stake_accounts` are paid by the stakers as the rent is refunded back to program owner.

As a result, the APX_SOL and IAPX_SOL exchange will be incorrect, allowing users exchange it at an incorrect rate.

Recommendations:

When providing the stake account balances, do include the temporary stake account balances on top of the persistent stake accounts, as they could be in the process of merging/withdrawal.

Also, reduce the balance by the rent to ensure that it accurately reflects the actual stake balance.

Dinero: Fixed in PR-106. We have changed admin cli implementation to consider temporary stake accounts balance and consider rent.

Zenith: Resolved.

[H-2] Force-deactivated stakes can deny withdrawals and stake management

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

Target

- merge_temporary_stake.rs#L114-L117
- merge_temporary_stake.rs#L124-L127
- update_stake_process_state.rs#L96-L99
- withdraw_temporary_stake.rs#L122-L125
- deactivate_persistent_stake.rs#L117-L120
- split_and_deactivate_temporary_stake.rs#L127-L130
- transfer_and_delegate_temporary_stake.rs#L136-L139

Description:

The stake management instructions enforce a strict state transition using `get_stake_process_state()` to ensure correct sequence of stake activation/de-activation and the subsequent actions like merge or withdraw.

However, it incorrectly assumes that deactivation can only be performed by the stake authority. That is because delegated stake can be force-deactivated by others such as via `deactivate-delinquent` instruction, cluster restart or some other means.

For example, if the validator vote account has not voted in the last 5 epochs, the stakes delegated to that vote account can be forced-deactivated using `deactivate-delinquent()`.

In this case the deactivated stakes will be stuck as there are no means to withdraw the stakes. That is because `deactivate_persistent_stake()` will fail as it requires the stake to be active. This will then prevent the admin from calling `withdraw_persistent_stake()`.

In addition, there are other instances (below) where it impose the condition `activation_state == StakeAccountActivationState::Active()`, which may not be possible if the stakes are force-deactivated.

- `merge_temporary_stakes()`
- `update_stake_process_state()`
- `withdraw_temporary_stake()`
- `deactivate_persistent_stake()`

- `split_and_deactivate_temporary_stake()`
- `transfer_and_delegate_temporary_stake()`

Recommendations:

Review the stake status checks in these instructions to handle the situation where the delegated stake is force-deactivated.

Consider adding a state reset in`post_epoch_update_stake_process_state()` to get the state transitions back on track. Include a new instruction to transit from `DelegatingTemporaryStake` to `Finished` when the persistent state account is deactivated. Same for `DelegatingPersistentStake`.

Then update the withdrawal instructions to allow admin to withdraw these force-deactivated stakes into the token buffer account.

Dinero: Fixed in PR-105.

Zenith: Resolved with forced withdrawals instructions to allow withdrawals of force-deactivated stakes.

4.3 Medium Risk

A total of 10 medium risk findings were identified.

[M-1] Institutional instructions can be blocked when global whitelist is disabled

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

Target

- whitelist_handler.rs#L30-L33

Description:

The `toggle_user_whitelist_state()` allows admin to enable or disable the entity state for the specific institution.

However, when `program_state.whitelist_feature_enabled == false`, the instruction `toggle_user_whitelist_state()` can be called by anyone as `is_admin_check_for_whitelisted_feature()` will not enforce access control to restrict to admin.

In that situation, `toggle_user_whitelist_state()` can be called to set `entity_state.is_enabled = false` for any institution. This will block the institution instructions as `validate_whitelist_user_self()` will fail when `entity_state.is_enabled = false`.

```
impl WhitelistHandler for WhitelistUserToggleHandler {
    fn execute<'info>(&self, ctx: &mut WhitelistContext<'_, 'info>) ->
    Result<()> {
        let entity_state = &mut ctx.entity_state;

        if let Some(payer) = ctx.payer.as_ref() {
            // Authentication
            is_admin_check_for_whitelisted_feature(&ctx.program_state,
    payer)?;
        }
        if ctx.institution.key() == crate::ID {
            msg!("No whitelisting necessary for retail users");
        } else {
            // institutional user
            entity_state.toggle_whitelist_state(ctx.set_to)?;
        }
        Ok(()) } }
```

Recommendations:

Always restrict `toggle_user_whitelist_state()` to admin regardless of the status of the global whitelist.

```
impl WhitelistHandler for WhitelistUserToggleHandler {
    fn execute<'info>(&self, ctx:  &mut WhitelistContext<'_, 'info>) ->
    Result<()> {
        let entity_state = &mut ctx.entity_state;
        if let Some(payer) = ctx.payer.as_ref() {
            // Authentication

    is_admin_check_for_whitelisted_feature(&ctx.program_state, payer)
                ?;
        }

    let payer = ctx.payer.clone().ok_or(DineroSolanaError::InvalidInput)?;
        is_admin(&ctx.program_state, &payer)?;

        if ctx.institution.key() == crate::ID {
            msg!("No whitelisting necessary for retail users");
        } else {
            // institutional user
            entity_state.toggle_whitelist_state(ctx.set_to)?;
        }
        Ok(())    }
}
```

Dinero: Fixed in PR-94, @9ded3d75690..., @7b8151f9d86..., PR-95.

Zenith: Resolved by only allowing admin to call `toggle_user_whitelist_state()`.

[M-2] Allowing reinitialization of `entity_validator_state` would compromise the staking state.

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

Target

- initialize_persistent_stake_account.rs

Description:

For the InitializePersistentStakeAccount instruction, anyone can call this instruction freely without enabling the whitelist. Observe the logic of this instruction if the `entities_validator_state` and `persistents_stake_account` accounts have already been initialized. It will also reinitialize the data of the `entities_validator_state` and skip `persistents_stake_account`, which will cause staking state errors and lock funds

```
pub fn process_init_stake_account(ctx:
    &mut StakeAccountInitializationContext) → Result<()> {

    is_admin_check_for_whitelisted_feature(&ctx.program_state,
    &ctx.program_owner)?;

    // Initialize persistent stake account state
    let entity_validator_state = &mut ctx.entity_validator_state;
    entity_validator_state.initialize(&ctx.clock)?;
    let persistent_stake_account = &mut
    ctx.persistent_stake_account.clone();

    let already_initialized;

    if persistent_stake_account.data_is_empty() {
        already_initialized = false;
    } else {
        let data = &persistent_stake_account.data.borrow();
        let stake_state = StakeState::deserialize(&mut &data[..]);
        already_initialized = match stake_state {
            Ok(StakeState::Initialized(..)) | Ok(StakeState::Stake(..)) ⇒
    true,
            _ ⇒ false,
        };
    }

    if already_initialized {
        msg!("Persistent stake account is already initialized");
    } else {
        ...
    }
```

For example, admin has already called `deactivate_persistent_stake` to deactivate stake. The status of `entity_validator_state` has changed to `StakeProcessState::DeactivatingPersistentStake()`.

If the whitelist is not enabled, the malicious party calls `InitializePersistentStakeAccount` to reinitialize the status of `entity_validator_state` to `StakeProcessState::Finished()`.

Admin need to repeatedly call `transfer_and_delegate_persistent_stake`, then wait for one epoch, then call `update_stake_process_state`,then call `deactivate_persistent_stake`, wait for another epoch , and only then can call `withdraw_persistent_stake`. During this period, if the whitelist is not enabled, the attacker can continue exploiting.

Recommendations:

It is recommended not to reinitialize the `entity_validator_state` account

Dinero: Resolved with PR-90.

Zenith: Verified.

[M-3] When the APX supply is small, the APX/UP exchange rate may increase rapidly.

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

Target

- w_calculation.rs
- vault_base.rs

Description:

```
impl<'info> IapxSolValuation<'info> {
    pub fn process(
        &mut self,
        stake_account_balance: u64,
        buffer_account_balance: u64,
    ) → Result<()> {
        ...
        // only calculate w when iapxSOL supply is greater than zero
        if iapx_sol_supply > 0 {
            let numerator = accounts_balances_sum
                .checked_sub(token_valuation_sum)
                .ok_or(DineroSolanaError::ArithmeticError)? as u128;

            let iapx_sol_value = numerator
                .checked_mul(PRECISION)
                .ok_or(DineroSolanaError::ArithmeticError)?
                .checked_div((iapx_sol_supply * TOKEN_SOL_DIFFERENCE)
    as u128)
                .ok_or(DineroSolanaError::ArithmeticError)?;

            program_state.w_institutional = iapx_sol_value;
        }
```

When the APX supply is small, the exchange rate may rise sharply. A higher exchange rate will increase the minimum amount of PX that needs to be burned to enter the vault, thereby raising the entry threshold for the vault.

```
pub fn process_vault_entry<'info>( ... ) → Result<()> {
    ...
    let apx_sol_to_mint: u64 = ((vault_entry_amount as u128 * PRECISION) /
    w_value) as u64;
    if apx_sol_to_mint < 1 {
        msg!("Insufficient vault entry amount");
        return err!(DineroSolanaError::InsufficientVaultEntryAmount);
    }
```

In extreme cases, assuming the system lacks APX holders and only a malicious actor holds a small amount of APX, the malicious actor could immediately donate a large number of tokens to the stake account when the system is frozen. After the system updates, `w_value` would increase significantly. Since only the malicious actor exists in the vault, all the donated tokens would be allocated to them, allowing them to exit the vault and mint PX. However, the increase in `w_value` would not be reversed, permanently raising the entry threshold for the vault.

For example:

1. At the beginning of the system, there is no one user in the vault. The malicious actor enters the vault at the end of the cycle and mints 1 APX.
2. When the system is frozen, they donate 1,000 SOL to the stake account.
3. `w_value` would then be updated to (10^12 * PRECISION) / (1 * 10^3) = 10^9 * PRECISION.
4. The malicious actor exits the vault and mints 10^9 PX(withdrew the donated funds).
5. The minimum cost to enter the vault is now raised to 10^9 PX -> 1,000 SOL.

Recommendations:

It is recommended that when the APX supply equals zero, entering the vault must involve minting an amount of APX that exceeds the specified minimum threshold. Additionally, when exiting the vault, if the post-exit APX supply is not zero, measures should be taken to ensure that the remaining APX supply does not fall below the minimum required value.

Dinero: Resolved with PR-112

Zenith: Verified.

[M-4] The `split_and_deactivate_temporary_stake` and `deactivate_persistent_stake` instructions lacks stake check.

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

Target

- split_and_deactivate_temporary_stake.rs
- deactivate_persistent_stake.rs

Description:

When withdrawing funds from a stake account, the system checks the rent conditions to determine which method to use for fund withdrawal. If the remaining balance is sufficient, it will use the `split_and_deactivate_temporary_stake` and `withdraw_temporary_stake` instructions; otherwise, it will use `deactivate_persistent_stake` and `withdraw_persistent_stake`.

```
impl<'info> SplitAndDeactivateTemporaryStake<'info> {
    pub fn process(&mut self, stake_authority_bump: u8) → Result<()> {
        ...
        require!(
            self.persistent_stake_account.lamports()
                ⩾ amount_to_split
                    .checked_add(minimum_delegation)
                    .ok_or(DineroSolanaError::ArithmeticError)?
                    .checked_add(
                        stake_account_rent_exemption_balance
                            .checked_mul(2)
                            .ok_or(DineroSolanaError::ArithmeticError)?
                    )
                    .ok_or(DineroSolanaError::ArithmeticError)?,
            DineroSolanaError::InvalidMethodCall
        );
        ...
}

impl<'info> DeactivatePersistentStake<'info> {
    pub fn process(&mut self, stake_authority_bump: u8) → Result<()> {
        ...
        require!(
            self.persistent_stake_account.lamports()
                < amount_to_split
                    .checked_add(minimum_delegation)
                    .ok_or(DineroSolanaError::ArithmeticError)?
                    .checked_add(
                        stake_account_rent_exemption_balance
                            .checked_mul(2)
                            .ok_or(DineroSolanaError::ArithmeticError)?
                    )
                    .ok_or(DineroSolanaError::ArithmeticError)?,
            DineroSolanaError::InvalidMethodCall
        );
        ...
}
```

However, the rent check does not account for whether the actual stake amount in the `persistent_stake_account`, after subtracting `amount_to_split`, would fall below the `minimum_delegation`. Instead, it relies solely on lamports checks. This oversight could allow malicious actors to perform a donation attack to temporarily block withdrawals.

For example, consider the following scenario:

1. `persistent_stake_account.lamports() = 1010`, where the rent is 10, `delegate.stake = 1000`, and the `minimum_delegation` is 100.
2. A request is made to withdraw 910 from the `persistent_stake_account`. Since `1010 < 10 + 100 + 910`, the admin should execute `deactivate_persistent_stake` followed by `withdraw_persistent_stake`.
3. A malicious actor could donate 10 to the `persistent_stake_account`, increasing `persistent_stake_account.lamports()` to 1020.
   - Now, since `1020 = 10 + 100 + 910`, the admin is forced to call `split_and_deactivate_temporary_stake`.
   - However, during the split operation, the `delegate.stake` remains at 1000. After splitting, the stake amount in the `persistent_stake_account` would be 90, which is below the `minimum_delegation`.
   - This causes the transaction to revert by this code, preventing the admin from processing the withdrawal request in this cycle.

This vulnerability could be exploited to disrupt legitimate withdrawal operations.

Recommendations:

It is recommended that when `delegation.stake - split_lamports < minimum_delegation`, the system should permit invoking the `deactivate_persistent_stake` instruction while preventing the execution of `split_and_deactivate_temporary_stake` instruction.

Dinero: Resolved with PR-101

Zenith: Verified.

[M-5] Claim order may be unfair

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

Target

- delay_claim.rs

Description:

For `slow_redemption`, once users convert PX to UPX, they can redeem immediately as long as there are sufficient funds in the `token_buffer_account`. Otherwise, they must wait for `split_and_deactivate_temporary_stake` to request funds from the stake account.

However, this fund request is delayed — it takes one epoch for the funds to become available in the `token_buffer_account`. During this waiting period, new redemption requests in the next epoch may front-run and consume the withdrawn funds, causing users from the previous epoch to be continuously unable to withdraw.

For example, consider the following scenario:

1. In epoch 10, **User 1** submits a `slow_redemption` request to redeem **100 PX**.
2. At the end of the epoch, the **owner** calls `split_and_deactivate_temporary_stake` to request SOL equivalent to the 100 PX via a split and deactivate operation.
3. After waiting one full epoch, the **owner** calls `withdraw_temporary_stake` to transfer the deactivated funds into the `token_buffer_account`.
4. At this point, **User 2**, who holds **APX**, exits the vault right after the staking rewards are settled and then calls `slow_redemption` followed by `delay_claim`. Because the funds are now available in the `token_buffer_account`, User 2 can redeem **immediately**.
5. However, **User 1**, who submitted the redemption request a full epoch earlier, still cannot redeem because the available funds have already been consumed. They must now wait for another full `split_and_deactivate_temporary_stake` → `withdraw_temporary_stake` cycle to receive their redemption.

Recommendations:

It is recommended to enforce a true withdrawal delay regardless of the availability of funds.

For example, if `split_and_deactivate_temporary_stake` is called at the end of epoch 10 to settle withdrawal requests made during that epoch, then after funds are withdrawn in epoch 11, only UPX tokens minted in epoch 10 or earlier should be eligible for withdrawal in epoch 11.

Any UPX minted in epoch 11 should be ineligible for immediate withdrawal, even if there is sufficient balance in the `token_buffer_account`. These tokens should instead be included in the next withdrawal settlement cycle—i.e., `split_and_deactivate_temporary_stake` at the end of epoch 11—and only become withdrawable after epoch 12.

Dinero: Resolved with PR-122

Zenith: Verified.

[M-6] `w_calculation` could result in a zero exchange rate

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

Target

- w_calculation.rs#L90-L93
- w_calculation.rs#L178-L181

Description:

The instruction `evaluate_apx_sol_valuation` is used to calculated the new exchange rate `apx_sol_value` based on the APX_SOL share of the account balance and `apx_sol_supply`.

However, there is an edge case that could cause `apx_sol_value` to be zerorized, when `accounts_balances_sum == token_valuation_sum && apx_sol_supply > 0`.

In that situation, we get the following calculation,

- `numerator = accounts_balances_sum - token_valuation_sum = 0`
- `apx_sol_value = numerator * PRECISION / apx_sol_supply = 0`

This situation is possible when the `accounts_balances_sum` is reduced at the end of the epoch (e.g. due to slashing), such that it equals the supply of `PX_SOL` and `UPX_SOL`. while there is a non-zero supply of `APX_SOL`.

The probability of `accounts_balances_sum == token_valuation_sum` is low but when it do occurs, it causes the `apx_sol_value` to become zero leading to total loss of value for `APX_SOL`.

The same issue applies to `evaluate_iapx_sol_valuation`.

```
let token_valuation_sum = px_sol_valuation
    .checked_add(upx_sol_valuation)
    .ok_or(DineroSolanaError::ArithmeticError)?;

// Sum of buffer account and stake account should be greater than
// value of pxSOL and upxSOL combined
require!(
    accounts_balances_sum >= token_valuation_sum,
    DineroSolanaError::InvalidStakeAccountBalance
);

// only calculate w when apxSOL supply is greater than zero
if apx_sol_supply > 0 {
    let numerator = accounts_balances_sum
        .checked_sub(token_valuation_sum)
        .ok_or(DineroSolanaError::ArithmeticError)? as u128;

    let apx_sol_value = numerator
        .checked_mul(PRECISION)
        .ok_or(DineroSolanaError::ArithmeticError)?
        .checked_div((apx_sol_supply * TOKEN_SOL_DIFFERENCE) as u128)
        .ok_or(DineroSolanaError::ArithmeticError)?;

    program_state.w_retail = apx_sol_value;
} // otherwise keep existing w as it is.
```

Recommendations:

This issue can be resolved as follows, to prevent `evaluate_apx_sol_valuation` and `evaluate_iapx_sol_valuation` from zerorizing the exchange rates.

```
        require!(
            accounts_balances_sum >= token_valuation_sum,
            accounts_balances_sum > token_valuation_sum,
             DineroSolanaError::InvalidStakeAccountBalance
        );
```

Dinero: Acknowledged. Issue highlighted above only comes in an event of slashing and slashing is out of scope for first release.

[M-7] Slashing could prevent update of vault exchange rate

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

Target

- w_calculation.rs#L176-L181
- w_calculation.rs#L88-L93

Description:

The instruction `evaluate_apx_sol_valuation()` has a check that requires `accounts_balances_sum ≥ token_valuation_sum`, which assumes that the total SOL backing the PX_SOL/UPX_SOL will not be less than the supply of PX_SOL/IPX_SOL. The same applies for `evaluate_iapx_sol_valuation()`.

However, this may not always hold true as it is possible for the validator's stakes to be 100% slashed if the validator is found to have violated the safety of the network as indicated in the docs.

This issue will prevent the update of the exchange rate for the vault to reflect a lower rate due to slashing, causing it to be inaccurate.

Recommendations:

Remove the check and allow update of exchange rate in all situation.

Dinero: Currently slashing is not in scope for initial version.

Zenith: Acknowledged by client.

[M-8] Admin can deactivate and withdraw more stakes than required

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

Target

- split_and_deactivate_temporary_stake.rs#L150-L172

Description:

`split_and_deactivate_temporary_stake()` allows the admin to split stake from the validator's persistent stake account, to top-up the gap in the token buffer account.

However, it does not keep track of pending top-ups. That means it is possible to split and deactivate from multiple validator stake accounts at the same time, to deactivate and withdraw more than what is required to top-up the gap in the token buffer account.

Recommendations:

This can be resolved by tracking and accounting for pending deactivating stakes.

Dinero: Initial scope is for single validator and to be resolved when moving to multi-validators.

Zenith: Acknowledged by client.

[M-9] Admin cannot fully deactivate and withdraw stakes from rogue validators

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

Target

- deactivate_persistent_stake.rs#L140-L159
- split_and_deactivate_temporary_stake.rs#L150-L172

Description:

Both `deactivate_persistent_stake()` and `split_and_deactivate_temporary_stake()` have a safety check to prevent the admin from deactivating more stakes than required to top-up the unstaked token buffer account.

However, these safety checks also prevent the admin from completely withdrawing stakes from rogue validators that are not performing/offline, preventing those stakes from earning rewards.

Recommendations:

Allow admin to fully deactivate and withdraw stakes from rogue/underperforming validators.

Dinero: Fixed in PR-105

Zenith: Resolved with forced activation and deactivation instructions.

[M-10] IPX_SOL holder can redeem and claim from any institution's token buffer account

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

Target

- delay_claim.rs#L82-L93

Description:

IPX_SOL is designed to allow whitelisted institutions to earn staking yields while ensuriing that their funds are not mixed with the retail PX_SOL. This is achieved with a separate instance of user instructions for institutions. A dedicated token buffer account is provided for each institution to isolate their funds.

However, IPX_SOL are actually shared between all the institutions, and using dedicate token buffer accounts for each institution do not help with fund isolation.

For example, a user can deposit SOL into institution A's token buffer account for IPX_SOL and then later redeem the IPX_SOL for SOL from institution B's token buffer account.

```
#[derive(Accounts)]
pub struct InstitutionDelayClaim<'info> {
    #[account(mut)]
    pub user: Signer<'info>,
    /// CHECK: CPI
    pub institution: UncheckedAccount<'info>,

    #[account(
        mut,
        seeds = [EntityState::BUFFER_ACCOUNT_SEED,
    institution.key().as_ref()],
        bump
    )]
    pub token_buffer_account: SystemAccount<'info>,
```

Recommendations:

If the intent is to isolate the funds at the institution level, then there would be a need to have separate IPX_SOL mint for each institution.

Dinero: Fixed in @05e8eec.

Zenith: Resolved by restricting institution (by verifying as the signer) to their own token buffer accounts.

4.4 Low Risk

A total of 4 low risk findings were identified.

[L-1] Interactions involving small amounts of funds are not subject to any fees.

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

Target

- deposit_base.rs
- redemption_base.rs
- vault_base.rs

Description:

The protocol calculates all fees using floor rounding and does not enforce a minimum fee threshold. As a result, interactions involving small amounts of funds may cause the calculated fees to round down to zero.

For example, in the deposit_process function, the deposit_fee is set to 1%. However, due to floor rounding and the lack of a minimum fee, any deposit amount below 100000 will result in a fee of zero.

```
let fee = token_utils::calculate_fee(amount, fee_percentage);
let fee_deducted_amount_to_mint:
    u64 = token_utils::spl_token_amount(amount - fee);
let fee_amount_to_mint = token_utils::spl_token_amount(fee);
...

pub fn spl_token_amount(amount: u64) → u64 {
    amount / TOKEN_SOL_DIFFERENCE
}
```

Recommendations:

It is recommended to set minimum amount for deposit, redeem, entry, and exit operations.

Dinero: Resolved with PR-111

Zenith: Verified.

[L-2] The initialization of the program is vulnerable to a front-run attack.

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

Target

- initialize.rs

Description:

The program does not enforce proper access control over the initialization of mint accounts and the State account. This allows a malicious actor to frontrun the initialization instruction, potentially undermining the intended functionality of the program.

For mint accounts, a malicious actor could initialize `PXSOL/IPXSOL`, `APXSOL/IAPXSOL`, and `UPXSOL/IUPXSOL` by different token_programs. This would lead to disruptions in protocol functionality.

```
#[derive(Accounts)]
pub struct InitializePXSOL<'info> {
    #[account(
        init_if_needed,
        payer = signer,
        mint::decimals = 6,
        mint::authority = px_sol_mint.key(),
        mint::freeze_authority = px_sol_mint.key(),
        seeds = [seed_constant::PX_SOL],
        bump
    )]
    pub px_sol_mint: InterfaceAccount<'info, Mint>,
    pub token_program: Interface<'info, TokenInterface>,
    ...
}

#[derive(Accounts)]
pub struct InitializeAPXSOL<'info> {
    #[account(
        init_if_needed,
        payer = signer,
        mint::decimals = 6,
        mint::authority = apx_sol_mint.key(),
        mint::freeze_authority = apx_sol_mint.key(),
        seeds = [seed_constant::APX_SOL],
        bump
    )]
    pub apx_sol_mint: InterfaceAccount<'info, Mint>,
    pub token_program: Interface<'info, TokenInterface>,
    ...
}
```

For example, if `PXSOL` and `APXSOL` are initialized with different `token_programs`, the staking system will fail to function properly, as the instructions are designed to operate with a single `token_program`.

For the state account, if a malicious actor frontruns the `InitializeInstitutional` or `InitializeRetail` instruction, they could gain control of privileged program accounts.

```
pub fn process_init_retail(ctx: &mut InitializationContext) → Result<()> {
    // Authentications
    if admin_exists(&ctx.state_account) {
        is_admin(&ctx.state_account, &ctx.signer)?;
    } else {
        ctx.state_account.admin = ctx.signer.key();
    }
    ...
```

Recommendations:

It is recommended to add a constraint in the initialization function to ensure that the signer is the expected hardcoded admin address. Alternatively, ensure that the program deployment and the initialization operation are executed within a single atomic transaction.

Dinero: Resolved with PR-110. Added auth for token initialization instructions. Original PR: PR-92

Zenith: Verified.

[L-3] `initialize_retail()` and `initialize_institutional()` can be called again to change exchange rate

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

Target

- initialize.rs#L98
- initialize.rs#L207

Description:

Both `initialize_retail()` and `initialize_institutional()` can be called again as they do not track if the relevant states have been intialized again.

This allows the admin to use these instruction to set a new exchange rate for the vaults, though the admin is trusted not to do so.

Recommendations:

Track the initialization states and prevent repeated calls of `initialize_retail()` and `initialize_institutional()`.

Dinero: Resolved with @98f3bf8....

Zenith: Resolved by using `init` instead of `init_if_needed`.

[L-4] Double rent calculation in `pre_epoch_split_and_deactivate_temporary_stake`

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

Target

- split_and_deactivate_temporary_stake.rs
- deactivate_persistent_stake.rs

Description:

```
// Persistent stake account has the balance required for split
// NOTE: after split, persistent stake account must have at least
    minimum_delegation + stake_account_rent_exemption_balance lamports
require!(
    self.persistent_stake_account.lamports()
        >= amount_to_split
            .checked_add(minimum_delegation)
            .ok_or(DineroSolanaError::ArithmeticError)?
            .checked_add(
                stake_account_rent_exemption_balance
                    .checked_mul(2)
                    .ok_or(DineroSolanaError::ArithmeticError)?
            )
            .ok_or(DineroSolanaError::ArithmeticError)?,
    DineroSolanaError::InvalidMethodCall
);
```

When executing the `pre_epoch_split_and_deactivate_temporary_stake` instruction, a balance check is performed on the `persistent_stake_account` to ensure that it has enough funds to continue staking after the split. However, during the calculation of the required minimum balance, the `stake_account_rent_exemption_balance` is mistakenly multiplied by 2. This results in an overestimation of the required amount, which may cause the `split_and_deactivate_temporary_stake` operation to fail unexpectedly, even though it should have succeeded.

The same issue exists in `deactivate_persistent_stake`.

Recommendations:

Don't multiply the rent by 2.

```
        // Persistent stake account has the balance required for split
        // NOTE: after split, persistent stake account must have at least
    minimum_delegation + stake_account_rent_exemption_balance lamports
        require!(
            self.persistent_stake_account.lamports()
                >= amount_to_split
                    .checked_add(minimum_delegation)
                    .ok_or(DineroSolanaError::ArithmeticError)?
                    .checked_add(
                        stake_account_rent_exemption_balance
                          .checked_mul(2)
                            .ok_or(DineroSolanaError::ArithmeticError)?
                    )
                    .ok_or(DineroSolanaError::ArithmeticError)?,
            DineroSolanaError::InvalidMethodCall
        );
```

Dinero: Resolved with PR-98

Zenith: Verified.

4.5 Informational

A total of 3 informational findings were identified.

[I-1] `buffer_limit` cannot be modified

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

Target

- admin_base.rs

Description:

`buffer_limit` is used in the retail system to reserve buffer funds for immediate withdrawals via `fast_redemption`. However, this value cannot be adjusted, which prevents the admin from tuning the buffer amount based on actual conditions. Additionally, when the last user exits the system, they must use `fast_redemption` to withdraw the remaining buffer funds.

```
pub fn process_init_entity(ctx: &mut EntityInitializationContext) →
    Result<()> {
    // Authentications
    is_admin_check_for_whitelisted_feature(&ctx.program_state_account,
    &ctx.signer)?;

    let already_initialized = ctx.entity_state_account.initialized;

    if already_initialized {
        msg!("Entity is already initialized");
        return Ok(());
    } else {
        if ctx.institution.key() == crate::ID {
            // retail account
            match ctx.buffer_limit {
                Some(limit) ⇒ {
                    // Retail account
                    ctx.entity_state_account.buffer_limit = limit;
                    ctx.entity_state_account.is_enabled = true; // Enabled by
    default
```

```
                }
                None => {
                    return err!(DineroSolanaError::InvalidArgumets);
                }
            }
        } else {
            // Institute account
            ctx.entity_state_account.buffer_limit = 0; // NOTE: If buffer
    limit is 0, we consider it as no limit
            ctx.entity_state_account.is_enabled = true; // Enabled by default
        }
        ...
```

Recommendations:

It is recommended to allow the adjustment of `buffer_limit`.

Dinero: Resolved with PR-113

Zenith: Verified.

[I-2] The funds may be released to the wrong account

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

Target

- fee_redemption.rs

Description:

```
#[derive(Accounts)]
pub struct InstitutionalFeeRedemption<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,
    #[account(
        init_if_needed,
        payer = admin,
        associated_token::mint = ipxsol_mint,
        associated_token::authority = admin,
    )]
    pub instituional_ata: InterfaceAccount<'info, TokenAccount>,
```

When releasing institutional fees, the recipient ATA account is named "institutional_ata." However, due to the `associated_token::authority` constraint, this account must actually be an ATA account controlled by the program admin.

Recommendations:

If the fees are intended to be sent to an institution-controlled account, then the `associated_token::authority` constraint needs to be removed. And it is recommended to revise the naming convention to avoid confusion if the fees are intended to be sent to the program admin.

Dinero: Resolved with PR-103

Zenith: Verified.

[I-3] Code improvements

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

Recommendations:

The code could be further improved with the following changes,

1. Allow PDA to be closed if there are no further use for it, to allow the rent payer to claim back the rent for the accounts.
2. Add in validation for fee percentages, to ensure that they are below 100% or within a reasonable range.
3. Consider saving the bumps for the PDA and load them, to ensure constant Compute Units (CU) utilization as described here. This is to prevent exceeding the CU limit when searching for 'far away' bumps.

Dinero: No. 2 and 3 are fixed with the following commit. Acknowledged for No. 1.

Zenith: Verified No. 2 and 3.