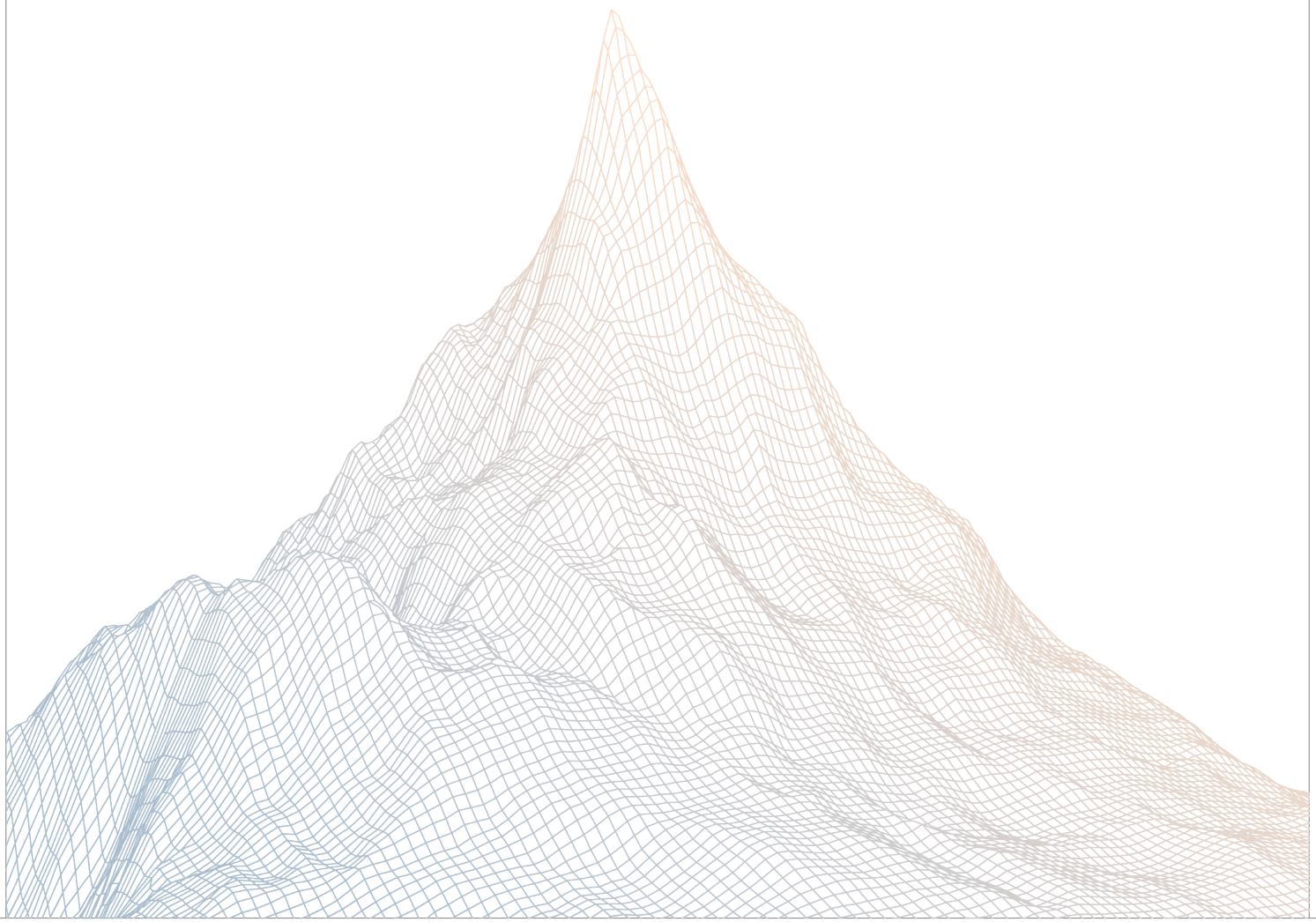
Zenith

GMX Solana

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: September 23th to October 9th, 2025

AUDITED BY: Mario Poneder
peakbolt

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 GMX Solana Protocol

GMX Solana is a decentralized spot and perpetual exchange that supports low swap fees and low price impact trades.

Trading is supported by unique multi-asset pools that earns liquidity providers fees from market making, swap fees and leverage trading.

Dynamic pricing is supported by Chainlink Oracles and an aggregate of prices from leading volume exchanges.

2.2 Scope

The engagement involved a review of the following targets:

Target	gmx-solana
Repository	https://github.com/gmsol-labs/gmx-solana/
Commit Hash	`bae0050e5fad88790c20b93fd9bd709413b95da8`
Files	`Diff from bae0050e5fad88790c20b93fd9bd709413b95da8 to ad0c4e88f736722a5a52a071ff6bd37fb682cc8c. gmsol_liquidity_provider program (located in programs/liquidity-provider).`

Target	GMX Solana Mitigation Review
Repository	https://github.com/gmsol-labs/gmx-solana/
Commit Hash	`7618169dfda4858ec87890bb66a1caafc8dca08c`
Files	`Diff from bae0050e5fad88790c20b93fd9bd709413b95da8 to ad0c4e88f736722a5a52a071ff6bd37fb682cc8c. gmsol_liquidity_provider program (located in programs/liquidity-provider).`

2.3 Audit Timeline

September 23, 2025	Audit start
October 9, 2025	Audit end
October 14, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	1
Informational	8
Total Issues	**12**

3

Findings Summary

ID	Description	Status
M-1	Unstake allows to bypass claim_enabled gate by using negligible unstake_amount	Resolved
M-2	Full withdrawal in unstake_lp() can be blocked by donation attack	Resolved
M-3	update_fees_state() does not persist update due to missing commit()	Resolved
L-1	APY gradient buckets can be retroactively modified altering accrued rewards	Resolved
I-1	Permissionless liquidity provider initialization allows first caller to seize authority and parameters	Acknowledged
I-2	Typo in log message in get_market_token_value instruction	Resolved
I-3	PriceFeedPrice.is_market_open() does not correctly reflect market status	Acknowledged
I-4	Liquidity withdrawal will fail when the index token is delisted	Acknowledged
I-5	close_empty_position() fails to handle the case position.created_at == 0	Acknowledged
I-6	Silent saturation of GT reward amount	Resolved
I-7	Incorrect instruction docstring for update_fees_state	Resolved
I-8	Incorrect config/flag names in changelog	Resolved

4

Findings

4.1 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] Unstake allows to bypass `claim_enabled` gate by using negligible `unstake_amount`

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

Target

- programs/liquidity-provider/src/lib.rs#L373-L417

Description:

The `claim_gt` instruction enforces a `global_state.claim_enabled` check, but the `unstake_lp` instruction mints GT rewards unconditionally as part of its claim-like flow. When `global_state.claim_enabled` is false, users can still realize accrued rewards by calling `unstake_lp` with a very small `unstake_amount`, effectively performing a "claim" while continuing to remain staked.

Recommendations:

It is recommended to gate the reward minting in `unstake_lp` with a `global_state.claim_enabled` check when the `unstake_amount` does not constitute a full exit, or require it to at least exceed a minimum unstake amount.

GMX Solana: Resolved with PR-244.

Zenith: Resolved by requiring a full exit when `global_state.claim_enabled` is false.

[M-2] Full withdrawal in `unstake_lp()` can be blocked by donation attack

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

Target

- /programs/liquidity-provider/src/lib.rs#L450-L491

Description:

`unstake_lp()` will close the vault token account when `full_exit == true`.

However, the `close_account()` will fail when it has a non-zero balance.

That means an attacker can transfer dust amount of the LP token to the vault token account to prevent the users from fully exiting their stake position.

```
// Decide transfer amount based on full_exit
let amount_to_transfer = if full_exit {
    old_amount
} else {
    unstake_amount
};
if amount_to_transfer > 0 {
    let gs_seeds: &[&[u8] = &[GLOBAL_STATE_SEED, &[global_state.bump];
    let signer_seeds: &[&[&[u8] = &[gs_seeds];
    let cpi_accounts = TransferChecked {
        from: ctx.accounts.position_vault.to_account_info(),
        mint: ctx.accounts.lp_mint.to_account_info(),
        to: ctx.accounts.user_lp_token.to_account_info(),
        authority: ctx.accounts.global_state.to_account_info(),
    };
    let cpi_ctx = CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        cpi_accounts,
        signer_seeds,
    );
    token_if::transfer_checked(cpi_ctx, amount_to_transfer,
    ctx.accounts.lp_mint.decimals)?;
```

```
}


if full_exit {
    // Full unstake: zero fields, close vault, close position account
    {
        let position = &mut ctx.accounts.position;
        position.staked_amount = 0;
        position.staked_value_usd = 0;
    }
    // Close the vault token account to return rent to owner
    let gs_seeds: &[&[u8] = &[GLOBAL_STATE_SEED, &[global_state.bump];
    let signer_seeds: &[&[&[u8] = &[gs_seeds];
    let close_ctx = CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        CloseAccount {
            account: ctx.accounts.position_vault.to_account_info(),
            destination: ctx.accounts.owner.to_account_info(),
            authority: ctx.accounts.global_state.to_account_info(),
        },
        signer_seeds,
    );
    token_if::close_account(close_ctx)?;
```

Recommendations:

When `full_exit == true`, transfer out everything from the position vault token account to the user so that it can be closed.

GMX Solana: Resolved with PR-244.

Zenith: Verified.

[M-3] `update_fees_state()` does not persist update due to missing `commit()`

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

Target

- /programs/store/src/instructions/exchange/update_fees.rs#L73-L102

Description:

`update_fees_state()` is added to manually update the borrowing fee factor during non-trading hours and fix the issue reported by GMX. The issue was borrowing fee factor will not be updated if there are no liquidation or collateral increase during non-trading hours.

However, `update_fees_state()` fails to commit the updates in the RevertibleMarket as it is missing `market.commit()` and `virtual_inventories.commit()`.

This means `update_fees_state()` will be ineffective, causing the borrowing fee factor not to be updated.

Recommendations:

Add `market.commit()` and `virtual_inventories.commit()` to persist the changes to the market state in `update_fees_state()`.

GMX Solana: Resolved with PR-243

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] APY gradient buckets can be retroactively modified altering accrued rewards

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

Target

- programs/liquidity-provider/src/lib.rs#L110-L173

Description:

The APY gradient used for reward calculation is read directly from the current `global_state.apy_gradient` at claim/unstake time, not from values in effect during the staking period. The authority can update past buckets via the `update_apy_gradient_sparse` or `update_apy_gradient_range` instruction, which immediately affects rewards for existing positions retroactively through the `compute_time_weighted_apy` and `compute_reward_with_cpi` functions.

Retroactive changes to APY buckets immediately change rewards that users have accrued but not yet claimed. Consequently, users cannot rely on APY at stake time and outcomes depend on potential future changes. This affects both `claim_gt` and `unstake_lp` flows, since both call the `compute_reward_with_cpi` function.

Recommendations:

It is recommended to mitigate this using a snapshot approach, alternatively clearly state in the documentation that the current design applies the “current” APY gradients at claim time.

GMX Solana: Resolved with PR-244.

Zenith: Resolved by adding clarifying comments.

4.3 Informational

A total of 8 informational findings were identified.

[I-1] Permissionless liquidity provider initialization allows first caller to seize authority and parameters

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

Target

- programs/liquidity-provider/src/lib.rs#L65-L89
- programs/liquidity-provider/src/lib.rs#L764-L779

Description:

The `initialize` instruction of the liquidity provider program can be called by any signer immediately after deployment and before the intended deployer initializes it. This allows anyone to back-run the deployment transaction and set themselves as `authority` and arbitrarily choose critical parameters such as `min_stake_value`, `initial_apy`, and `pricing_staleness_seconds`.

In case this happens, the program needs to be redeployed successfully, and downstream references in the SDK need to be updated to the new liquidity provider program ID.

Recommendations:

It is recommended to require the program's keypair to co-sign the `initialize` instruction, preventing anyone except the deployer from calling it directly. Add the following to the `Initialize` account context:

```
#[account(address = crate::ID)]
pub program: Signer<'info>
```

GMX Solana: Acknowledged. Considering that fixing this issue would cause significant inconvenience for automated testing and the likelihood of it occurring is quite low, we've decided not to address it.

Zenith: Acknowledged.

[I-2] Typo in log message in `get_market_token_value` instruction

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

Target

- programs/store/src/instructions/market.rs#L956

Description:

There is a misspelled word in a `msg!` log within the `GetMarketTokenValue::evaluate` flow. The message currently reads "evluation was not performed" instead of "evaluation was not performed."

Recommendations:

It is recommended to fix the typo by changing the message to "evaluation was not performed".

GMX Solana: Resolved with PR-248.

Zenith: Verified.

[I-3] `PriceFeedPrice.is_market_open()` does not correctly reflect market status

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

Target

- /programs/store/src/states/oracle/feed.rs#L152
- /crates/utils/src/price/feed_price.rs#L114-L150

Description:

In this PR, changes are introduced to have different parameters (min_collateral_factor, etc) and operations (liquidation) that are/can be used when market is closed.

`PriceFeed::check_and_get_price()` will get the market status from `PriceFeedPrice.is_market_open()`, to determine if the market is open or closed.

However, a binary market status (open/closed) does not reflect the actual market state. Based on Chainlink documentation, it is interpreted that there are actually 3 market states as follows,

- Market closed
- Market open and trading allowed
- Market open and trading disallowed (when lastUpdateTimestamp is not current timestamp)

That means `PriceFeedPrice.is_market_open()` will wrongly set the market as closed even though it is in the state "market open and trading disallowed".

In that situation, the wrong market close parameters will be used and operations such as liquidation can still be performed.

Recommendations:

This issue can be resolved by fixing `PriceFeedPrice.is_market_open()` to instead return the 3 market states (as described above) and correctly handles them.

GMX Solana: For the case where the market is open but trading is disallowed, we currently treat it as market closed, unless there is a clear and practical reason to apply different parameters between the two states.

Zenith: Acknowledged by client. For current design they will regard the state "Market open and trading disallowed" as "Market closed" as there is no reasons now to differentiate them. This includes the scenarios sudden oracle failures and trading halts.

[I-4] Liquidity withdrawal will fail when the index token is delisted

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

Target

- /programs/store/src/states/oracle/price_map.rs#L52
- /crates/model/src/price.rs#L93-L100

Description:

Based on the Chainlink documentation, RWA that are delisted will have their price reset to zero.

However, there are two price validations that forbids zero price

1. `PriceMap::set()` will call `SmallPrices::from_price()`, which does not allow `price.min.value == 0`.

```
pub(crate) fn from_price(
    price: &gmsol_utils::Price,
    is_synthetic: bool,
    is_open: bool,
) -> Result<Self> {
    // Validate price data.
    require_eq!(
        price.min.decimal_multiplier,
        price.max.decimal_multiplier,
        CoreError::InvalidArgument
    );
    require_neq!(price.min.value, 0, CoreError::InvalidArgument);
```

2. `Prices::validate()` will call `Price::is_valid()`, which does not allow `index_token_price` to be zero.

```
impl<T> Price<T>
```

```
where
    T: num_traits::Zero + CheckedAdd + CheckedDiv + num_traits::One,
{
    fn is_valid(&self) -> bool {
        !self.min.is_zero() && !self.max.is_zero()
    && self.checked_mid().is_some()
    }
}
```

This issue will cause liquidity withdrawal to fail after delisting. That is because there are no mechanism to force users to withdraw their liquidity before the market is delisted.

For deposit and increase/decrease positions, it is expected that the protocol will tighten the collateral ratio and discourage new increase/deposit to force users to wind down positions and stop providing liquidity ahead of the delisting.

Recommendations:

Update the `SmallPrices::from_price()` and `price::is_valid()` to allow zero price for index tokens.

GMX Solana: But non-zero prices are a fundamental assumption of the model, allowing a zero price to participate in withdrawal calculations is currently unacceptable.

In fact, we believe this issue does not need to be addressed at the program level. It is equivalent to the corresponding RWA market having an incorrect price.

In such cases, our recommended standard solution for deployers is to replace the price provider of the index token. If necessary, they may switch to a fixed-price feed created with Switchboard and conduct an orderly withdrawal of liquidity, with trading functions disabled.

Zenith: Acknowledged as this is handled manually by replacing the price provider of the index token.

[I-5] `close_empty_position()` fails to handle the case `position.created_at == 0`

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

Target

- programs/store/src/instructions/exchange/order.rs#L1146-L1168

Description:

`close_empty_position()` allows positions to be closed manually by the owner only when the position is older than the configured `MinPositionAgeForManualClose`, as determined by `position.created_at`.

However, `position.created_at` will be zero for all existing positions, which means that the existing positions will fulfill the minimum age condition and can be closed manually by the owners.

```
impl CloseEmptyPosition<'_> {
    pub(crate) fn invoke(ctx: Context<Self>) -> Result<()> {
        ctx.accounts.validate()
    }

    fn validate(&self) -> Result<()> {
        let now = Clock::get()?.unix_timestamp;
        let position = self.position.load()?;
        require!(position.state.is_empty(),
    CoreError::PreconditionsAreNotMet);
        let min_age = *self
            .store
            .load()?
            .validate_not_restarted()?
            .get_amount_by_key(AmountKey::MinPositionAgeForManualClose)
            .ok_or_else(|| error!(CoreError::Internal))?;
        let min_age = min_age
            .try_into()
            .map_err(|_| error!(CoreError::ValueOverflow))?;
        let closable_after = position.created_at.saturating_add(min_age);
        require_gte!(now, closable_after,
```

```
    CoreError::PreconditionsAreNotMet);
        Ok(())
    }
}
```

Recommendations:

Update `close_empty_position()` to handle the case where `position.created_at == 0`.

GMX Solana: The "timelock" mechanism for closing empty positions is intended to protect keepers from sustained attacks. A limited number of empty position accounts that can be closed immediately do not pose such a threat. Once the position accounts are recreated, the same attack cannot be performed again.

Zenith: Acknowledged by client as the small amount of existing positions that can be manually closed would not DoS the keeper (and exhaust the network fee).

[I-6] Silent saturation of GT reward amount

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

Target

- programs/liquidity-provider/src/lib.rs#L648

Description:

The `calculate_gt_reward_amount` function in the `liquidity_provider` program silently saturates GT reward amounts that exceed `u64::MAX` without any warning or error logging:

```
Ok(gt_raw.min(u64::MAX as u128) as u64)
```

When `gt_raw` exceeds `u64::MAX`, the function returns the maximum `u64` value instead of the calculated amount.

Recommendations:

It is recommended to add logging when saturation occurs to allow for proper monitoring if this unlikely edge case ever arises.

GMX Solana: Resolved with PR-236.

Zenith: Verified.

[I-7] Incorrect instruction docstring for `update_fees_state`

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

Target

- programs/store/src/lib.rs#L2558

Description:

In the PR adding the `update_fees_state` instruction, the instruction's documentation/comment in `lib.rs` (and reflected in the IDL/docs) incorrectly states: "Update the closed state for the market." This appears to be a copy/paste error from `update_closed_state`. The mismatch can mislead integrators and codegen consumers, and it may surface in SDKs that display instruction docs from the IDL.

Recommendations:

It is recommended to update the `update_fees_state` instruction docstring and regenerate the IDL so it reflects the corrected docs.

GMX Solana: Resolved with PR-247

Zenith: Verified.

[I-8] Incorrect config/flag names in changelog

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

Target

- CHANGELOG.md#L30-L45

Description:

The CHANGELOG uses wrong names in the "Unreleased" section:

- `min_collatereal_factor_for_liquidation` → `min_collateral_factor_for_liquidation`
- `enable_market_closed_params` → `use_market_closed_params`

Recommendations:

It is recommended to replace all occurrences of the above config/flag names with their corrected versions.

GMX Solana: Resolved with PR-246.

Zenith: Verified.