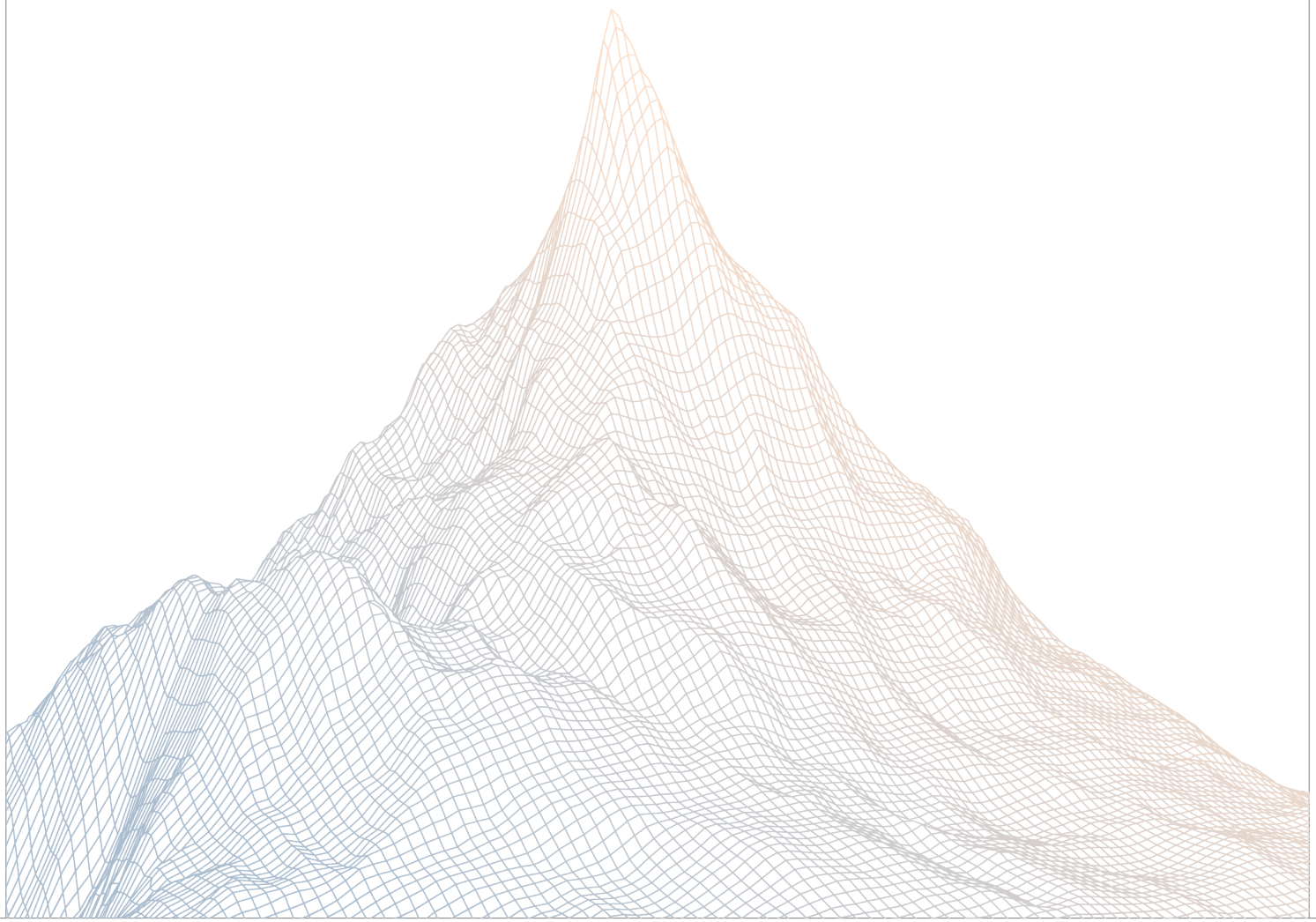
Zenith

GMX Solana

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: July 18th to July 28th, 2025

AUDITED BY: DadeKuma
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	`bf7b175859f776308de7f6c18b24ed0628fd64aa`
Files	`Diff up to bf7b175859f776308de7f6c18b24ed0628fd64aa`

Target	GMX Solana Mitigation Review
Repository	https://github.com/gmsol-labs/gmx-solana
Commit Hash	`3202f7ca1a01a076425af59d2bca7369fe9c156c`
Files	`Changes in the latest source code version`

2.3 Audit Timeline

July 18, 2025	Audit start
July 28, 2025	Audit end
July 31, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	5
Low Risk	3
Informational	0
Total Issues	**9**

3

Findings Summary

ID	Description	Status
H-1	Markets are vulnerable to systemic negative price impact	Resolved
M-1	Virtual inventory impact should be excluded for GLV shift between markets in the same VI	Resolved
M-2	create_virtual_inventory_for_positions() prevents creation of new virtual inventory account	Resolved
M-3	An imbalance between VI for swaps and pool liquidity may cause a DoS	Acknowledged
M-4	Lack of validation of expires_at may result in stale prices	Resolved
M-5	from_chainlink_report() could fail due to underflow error in for last_update_diff	Resolved
L-1	join_virtual_inventory_for_swaps fails to verify pool token decimals	Resolved
L-2	Zero price impact is not capped	Resolved
L-3	is_market_open() should verify last_update_timestamp regardless of last_update_diff_nanos	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] Markets are vulnerable to systemic negative price impact

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

Target

- crates/model/src/action/increase_position.rs
- crates/model/src/position.rs#L583
- crates/model/src/position.rs#L534

Description:

All markets sharing the same token are currently vulnerable to systemic negative price impact. This occurs because the price impact calculation does not consider any cross-market liquidity, leading to cascading imbalances and a higher-than-expected global negative price impact.

Suppose the following scenario:

1. Two balanced markets exist: `solUsdc` and `solUsdt`.
2. Users execute multiple sizable increase orders on `solUsdc`, causing a pool imbalance and a negative price impact.
3. Naturally, the same actions will be repeated within the `solUsdt` market. As the calculation of the price impact is independent between markets, it will result in the same negative price impact as point 2.

The issue is that the negative price impact calculated in point 3 should be much higher to prevent an imbalance cascade between multiple markets.

Note: this behaviour is also not consistent with the Solidity version, where point 3 would have indeed resulted in a higher price impact.

Recommendations:

Consider keeping track of cross-market liquidity/imbalances when calculating the price impact for a specific market.

GMX: Resolved with @d32bb0aab992...

Zenith: Verified.

4.2 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] Virtual inventory impact should be excluded for GLV shift between markets in the same VI

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

Target

- market.rs#L548-L613

Description:

Markets within a GLV are likely to be in the same virtual inventory for swap as they share the common Long/Short tokens. That means a GLV shift would not result in any net change in the virtual inventory, which is why GMXv2 will ignore virtual price impact for GLV Shifts.

However, in Solana, the GLV shift will still calculate and utilize the virtual price impacts for the withdrawal and deposit actions.

This could prevent GLV Shift from executing if the negative virtual price impact (from withdrawal/deposit action) is significantly worse than the positive price impact (from the other action). This will cause the net price impact to exceed the max price impact factor and thus fail to execute as enforced by `validate_shift_price_impact()`.

Even though the markets are in the same virtual inventory, It is possible for a GLV shift's negative virtual price impact to be worse than the positive price impact, because the positive virtual price impact is ignore if the normal price impact is positive.

Recommendations:

Consider excluding virtual inventory price impact for GLV shift, if the markets are in the same virtual inventory.

GMX: Resolved with @39d286d0d2c...

Zenith: Verified. Resolved by ignoring virtual price impact for deposit when the GLV shift is performed for markets in the same virtual inventory, as there is no net change in virtual

inventory. This also prevents charging the same but opposite signed virtual price impact again for deposit (which would offset the virtual price impact for withdrawal).

[M-2] `create_virtual_inventory_for_positions()` prevents creation of new virtual inventory account

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

Target

- virtual_inventory.rs#L265-L269
- virtual_inventory.rs#L314

Description:

The virtual inventory account for positions is seeded based on the `index_token.key()`. This allows `create_virtual_inventory_for_positions` to enforcesa canonical virtual inventory account for every index token.

However, this will be problematic if the virtual inventory account is disabled, which prevents it from being enabled or closed. And `create_virtual_inventory_for_positions()` will not be able to create another virtual inventory account for the same index token.

```
pub struct CreateVirtualInventoryForPositions<'info> {
    /// Authority.
    #[account(mut)]
    pub authority: Signer<'info>,
    /// Store account.
    pub store: AccountLoader<'info, Store>,
    /// Index token address.
    /// CHECK: only the address of this account is used.
    pub index_token: UncheckedAccount<'info>,
    /// The virtual inventory account to create.
    #[account(
        init,
        payer = authority,
        space = 8 + VirtualInventory::INIT_SPACE,
        seeds = [
            VIRTUAL_INVENTORY_FOR_POSITIONS_SEED,
            store.key().as_ref(),
            index_token.key.as_ref(),
        ],
        bump,
```

```
    )]
```

Recommendations:

This can be resolved by using an unique virtual inventory index instead of `index_token.key()`.

GMX: Resolved with @04bd1298d397...

Zenith: Verified. Resolved by decrementing `ref_count` in virtual_inventory so that it can be safely closed when it reaches zero. This will allow a new virtual inventory to be created.

[M-3] An imbalance between VI for swaps and pool liquidity may cause a DoS

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

Target

- crates/model/src/market/base.rs

Description:

The virtual inventory implementation contains a discrepancy in delta updates for swaps. Unlike GMX's standard implementation which uses saturating arithmetic, the current version uses checked summation that reverts when underflow occurs.

Instead of saturating the lower bound towards zero, in the current implementation this would cause a revert.

This creates a potential DoS vulnerability in GMX Solana when an imbalance exists between virtual inventory and actual pool liquidity, and the VI value becomes lower than available pool liquidity.

As a result, subsequent attempts to call the function fail due to the arithmetic rounding behavior.

Recommendations:

Consider saturating the lower bound towards zero instead of reverting.

GMX: We believe this is not actually an issue that requires any changes. The VI for swaps and the liquidity pools are designed to remain fully synchronized, so a VI for swaps underflow could only occur if the last liquidity pool underflows, or if some other component fails to maintain synchronization between the VI for swaps and the liquidity pools. In other words, if the situation you described were to occur, it would indicate the presence of a bug elsewhere rather than a problem with the current logic itself.

If it turns out that such a bug does exist and leads to a VI for swaps underflow, we could consider temporarily disabling the affected VI until the issue is fully resolved.

Zenith: It seems that there are a few differences between the two implementations: GMX uses the bounded/saturating delta for many operations like `applyDeltaToSwapImpactPool`,

`applyDeltaToPositionImpactPool`, `applyDeltaToVirtualInventoryForSwaps`, while GMX-sol always uses a checked delta. So in _theory_, the issue should never happen.

To be on the safe side, the issue will be acknowledged, and it case it somehow happens, it's sufficient to temporarily disable the VI to prevent the DoS.

[M-4] Lack of validation of `expires_at` may result in stale prices

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

Target

- chainlink-datastreams/src/report.rs

Description:

According to the datastream SDK documentation, the `expires_at` field indicates that: `/// - expires_at: Latest timestamp where the report can be verified onchain.`

The datastream verification process fails to validate the `expires_at` timestamp, as the field is never used other than being assigned to the report struct, potentially allowing expired price reports to be accepted.

Recommendations:

Consider returning an error when `expires_at` has already passed.

GMX: Although we believe the verifier program should validate `expires_at`, an additional check has still been added. Resolved with @661a5f9a55...

Zenith: Verified.

[M-5] `from_chainlink_report()` could fail due to underflow error in for `last_update_diff`

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

Target

- crates/chainlink-datastreams/src/gmsol.rs#L53-L57

Description:

In `from_chainlink_report()`, it will handle the case when `report.last_update_timestamp()` is present in the report.

Specifically it will calculate the price's age by obtaining `last_update_diff = observations_timestamp_ns.checked_sub(last_update_timestamp_ns)`.

However, as `observations_timestamp_ns` is scaled to nanoseconds, while `last_update_timestamp_ns` is already in nanoseconds precision, the `checked_sub()` could result in a underflow error despite being valid.

For example,

1. Suppose`last_update_timestamp_ns = 1_999_999_999` while `observations_timestamp_ns = 1 * 1_000_000_000 = 1_000_000_000` after scaling. This could be possible if `observations_timestamp` is truncated and rounded down.
2. Now the calculation `observations_timestamp_ns.checked_sub(last_update_timestamp_ns)` will underflow as `1_000_000_000 - 1_999_999_999` is negative.

That means `from_chainlink_report()` will fail whenever the underflow occurs, preventing the instructions from obtaining price data.

```
let observations_timestamp_ns = u64::from(observations_timestamp)
    .checked_mul(NANOS_PER_SECOND)
    .ok_or(crate::Error::Overflow(
        "observations_timestamp is too large",
    ))?;
let last_update_diff = observations_timestamp_ns
    .checked_sub(last_update_timestamp_ns)
```

```
    .ok_or(crate::Error::InvalidRange(
        "last_update_timestamp larger than observations_timestamp",
    ))?;
```

Recommendations:

Consider scaling down `last_update_timestamp_ns` to seconds precision and truncate subseconds to obtain the `last_update_diff` in seconds instead.

GMX: Resolved with @4a0561bd1141...

Zenith: Verified. Resolved by allowing `last_update_timestamp_ns` to exceed `observations_timestamp_ns` by < 1 second to account for the rounding down of `observation_timestamp`.

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] `join_virtual_inventory_for_swaps` fails to verify pool token decimals

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

Target

- virtual_inventory.rs#L207-L220

Description:

`join_virtual_inventory_for_swaps` allows the Market Keeper to add a market to a specific virtual inventory.

However, it fails to verify that the decimals of the market's long/short token mints matches the other markets in the virtual inventory.

If the decimals are different, `join_virtual_inventory_for_swaps` will incorrectly adjust the balance in the virtual inventory.

Recommendations:

Though the Market Keeper could verify manually, it is recommended to add this as an on-chain validation to prevent any accidental change.

GMX: Resolved with @fb706ecec69...

Zenith: Resolved by verifying the long/short token decimals in `join_virtual_inventory_for_swaps()`.

[L-2] Zero price impact is not capped

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

Target

- crates/model/src/market/perp.rs

Description:

The current implementation differs from GMX's in how price impact caps are applied. While GMX applies the cap for non-negative price impact (>= 0), the current implementation only caps positive price impact (> 0).

This creates a discrepancy in handling the zero edge case. Although this doesn't currently cause any impact due to the specific arithmetic operations involved, it is recommended to re-align the two implementations.

Recommendations:

Consider the following change:

```
if impact.is_positive()
if !impact.is_negative()
```

GMX: Resolved with @a22b5c21fc5...

Zenith: Verified.

[L-3] `is_market_open()` should verify `last_update_timestamp` regardless of `last_update_diff_nanos`

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

Target

- /crates/utils/src/price/feed_price.rs#L106-L123

Description:

in `is_market_open()`, it will check that the market is open by checking that the `current_timestamp - last_update_timestamp ≤ heartbeat_duration`. Basically it will reject old prices that were from the previous session before market closes, even when market status is open.

However, it only perform this check when `last_update_diff_nanos > 0` and always return true when `last_update_diff_nanos == 0`.

This is incorrect because when `last_update_diff_nanos == 0`, it only means that the `observations_timestamp - last_update_timestamp == 0`. That does not mean the price is not from the previous session.

Fortunately, a subsequent stale-ness check in `check_and_get_price()` will still reject prices from the previous session. Though it is recommended to handle `last_update_diff_nanos == 0` case to make it semantically correct for `is_market_open()`.

```
/// Returns whether the market is open.
pub fn is_market_open(&self, current_timestamp: i64, heartbeat_duration:
    u32) -> bool {
    if !self.flags.get_flag(PriceFlag::Open) {
        return false;
    }

    let last_update_diff_nanos = i64::from(self.last_update_diff_nanos);

    if last_update_diff_nanos > 0 {
        // The use of `saturating_sub` here is valid because:
        //   - In the case of overflow, the function returns `false`,
        //     and since `current_timestamp ≥ ts + i64::MAX`, it must
        //     also hold that `current_timestamp ≥ ts + heartbeat_duration`,
```

```
    //     and thus `current_timestamp > last_update +
heartbeat_duration`.
    //   - In the case of underflow, the function returns `true`,
    //     and since `current_timestamp ⩽ ts + i64::MIN`, it follows that
    //     `current_timestamp ⩽ ts - last_update_diff_secs`, and thus
    //     `current_timestamp - last_update ⩽ heartbeat_duration`.
    // Therefore, we only need to check the case where no overflow or
underflow occurs.
    let current_diff = current_timestamp.saturating_sub(self.ts);
    let heartbeat_duration = heartbeat_duration.into();
    if current_diff ⩾ heartbeat_duration {
        return false;
    }

    last_update_diff_nanos
        ⩽ heartbeat_duration
            // The use of `saturating_sub` is valid because:
            //   - Underflow is impossible because of the check above,
and in the case of
            //     overflow, the function returns `true`, and since
            //     `heartbeat_duration ⩾ current_diff + i64::MAX`, it
must also hold that
            //     `heartbeat_duration ⩾ current_diff +
last_update_diff_secs`, and thus
            //     `current_timestamp - last_update ⩽
heartbeat_duration`.
            .saturating_sub(current_diff)
            // The use of `saturating_mul` is valid because:
            //   - Underflow is impossible because `heartbeat_duration >
current_diff`,
            //     and in the case of overflow, the function returns
`true`, and since
            //     `(heartbeat_duration - current_diff) *
NANOS_PER_SECOND ⩾ i64::MAX`, it must
            //     hold that `(heartbeat_duration - current_diff) *
NANOS_PER_SECOND ⩾ last_update_diff_nanos`.
            .saturating_mul(NANOS_PER_SECOND)
} else {
    true
}
```

Recommendations:

Perform the `last_update_timestamp` check even when `last_update_diff_nanos == 0`.

GMX: Resolved with @4a0561bd1141...

Zenith: Verified. Resolved by removing `if last_update_diff_nanos > 0 {` condition to handle all case, including when `last_update_diff_nanos == 0`. An additional flag `LastUpdateDiffEnabled` is used to determine whether the `last_update_diff_nanos` is set and enabled.

Also updated to `if current_diff > heartbeat_duration {` to make it consistent with staleness check in feed.rs#L157.