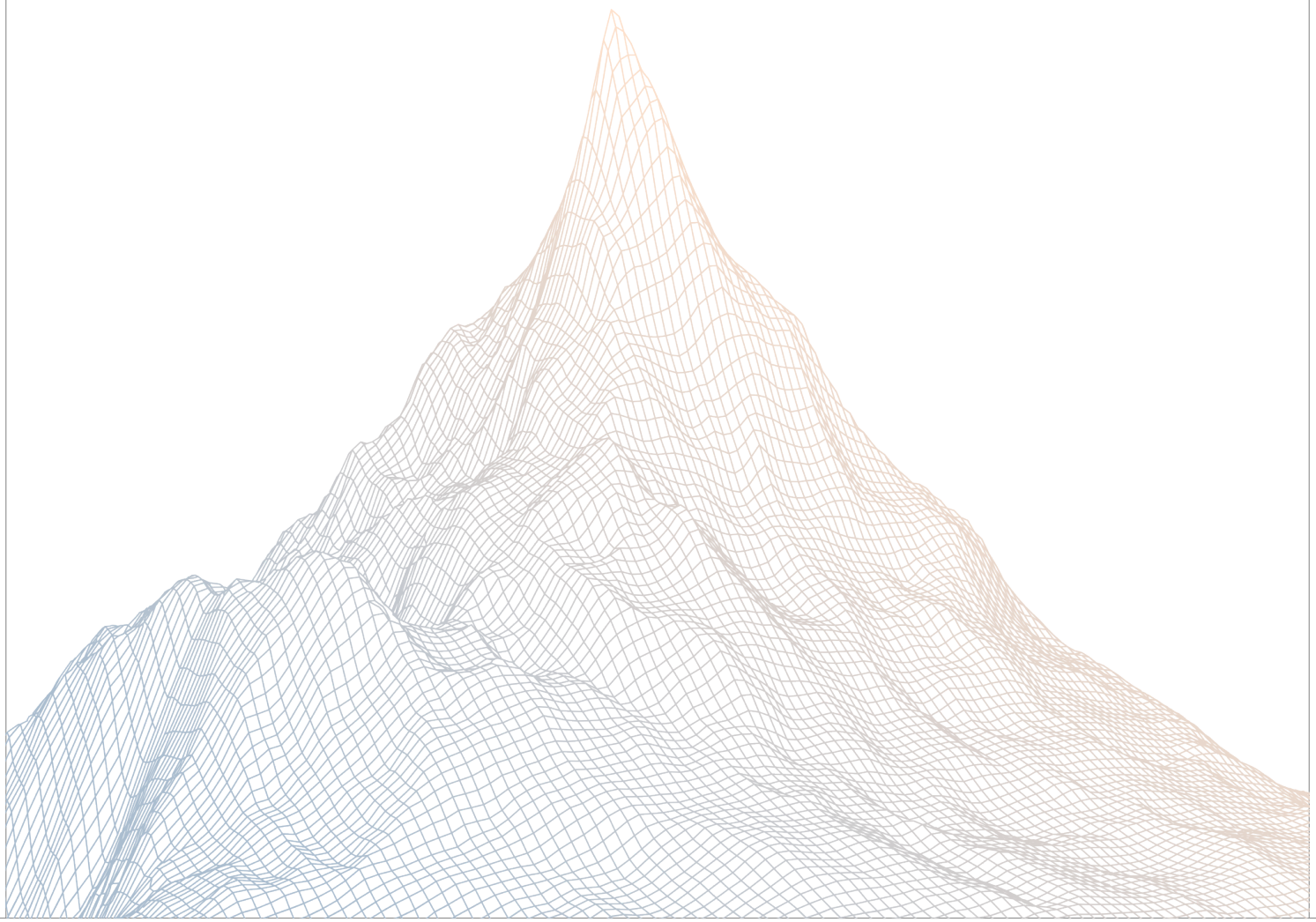
Zenith

GMX Solana

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: May 27th to June 10th, 2025

AUDITED BY: DadeKuma
lllllll
Mario Poneder

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About GMX Solana Protocol 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 7

4.1 Critical Risk 8

4.2 High Risk 10

4.3 Medium Risk 13

4.4 Low Risk 18

4.5 Informational 32

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	`ee31f0921d0f645de811b0fc93679fd2d7ba140a`
Files	`Changes in the latest source code version`

Target	GMX Solana Mitigation Review
Repository	https://github.com/gmsol-labs/gmx-solana
Commit Hash	`a5b38f3be9cc1acfe810bb9072e98c5ba8a12fbb`
Files	`Changes in the latest source code version`

2.3 Audit Timeline

May 27th, 2025	Audit start
June 10th, 2025	Audit end
June 19th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	2
Medium Risk	4
Low Risk	11
Informational	9
Total Issues	**27**

3

Findings Summary

ID	Description	Status
C-1	Callback failures can be used to enable risk-free trades	Resolved
H-1	Changes to claimable_time_window can lead to collisions of claimable accounts	Resolved
H-2	Price adjustments/validation use better-than-market prices for some oracles	Resolved
M-1	Competitions can be manipulated through many small-volume trades	Resolved
M-2	update_order() never checks for v2 callbacks	Resolved
M-3	Order keepers can extract value through transaction ordering	Acknowledged
M-4	close_empty_claimable_account can be abused for DoS and rent theft by malicious order keepers	Acknowledged
L-1	Competition participants can game volume metric	Resolved
L-2	Mutable Config/Competition accounts in all order instructions will lead to contention	Acknowledged
L-3	Calls to initialize_config() can be front-run	Resolved
L-4	Lack of uniqueness validation of trade_event in OnExecuted callback	Acknowledged
L-5	Missing validation of association between trade_event and trader/participant in OnExecuted callback	Resolved
L-6	Improper use of require_neq macro for Pubkey comparison	Resolved
L-7	Competitions can expire instantly	Resolved
L-8	Zero max_deviation in case of low-decimal mid_price can cause unexpected price validation failures	Resolved
L-9	Malicious keepers can deliberately cancel protocol operations by abusing the throw_on_execution_error flag	Acknowledged
L-10	Precision loss in with_max_deviation_factor function due to division by RATIO_MULTIPLIER	Resolved
L-11	try_adjust_price() may return an incorrect result	Resolved

ID	Description	Status
I-1	No way to close and reclaim rent for ActionStats and Participant accounts	Resolved
I-2	InvalidTradeEvent is never used	Resolved
I-3	No way to deactivate a Competition	Resolved
I-4	Typos	Resolved
I-5	create_order()/update_order() do not emit event like the v2 versions do	Resolved
I-6	Chainlink oracle logic is unimplemented	Resolved
I-7	The GtBank struct has changed without version-based handling	Acknowledged
I-8	The ActionHeader struct has changed without version-based handling	Acknowledged
I-9	Unresolved TODO and FIXME comments	Resolved

4

Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Callback failures can be used to enable risk-free trades

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

Target

- programs/callback/src/interface.rs
- programs/store/src/states/common/action.rs

Description:

When creating an order, a user is allowed to specify a callback program to be executed by the store program via CPI when actions related to the order are performed. The current interface overrides anchor's default behavior of requiring that the program ID matches one of a pre-defined list of program IDs, and allows any arbitrary program to be used:

```
fn check_id(_id: &anchor_lang::prelude::Pubkey) → anchor_lang::Result<()> {
    Ok(())
}
```

Since one of the callback instructions is executed during order creation, the callback program at least guarantees that the program has that instruction and that instruction does not fail with the accounts passed to it by the store program. However, because it's an arbitrary program, the program may not even have matching instructions for the other callback functions, or it may contain code that can fail later CPIs, or may be upgraded to do so.

By providing a program that fails the `on_executed` instruction based on a flag that user controls, a malicious user can perform risk-free trades by submitting market orders and causing the execution to fail until the price moves significantly in their favor. Since the failure of a CPI unrecoverably fails the whole transaction, all attempts to execute the order by keepers will fail while the program is in this state. If the price has moved in their favor the trader can trigger the callback to stop failing, at which point a keeper would execute the order with oracle prices corresponding to right after the order was created. Once the

order is filled, the user can immediately submit a new market close order for a profit.

Another possible attack vector could be to create a very small position, and then to submit thousands of close orders with broken execution callbacks, which would indefinitely DOS keepers trying to execute the orders.

Recommendations:

Only allow specific, vetted, programs to be used by traders as callback programs, and don't allow them to be upgradeable.

GMX Solana: Resolved @af0875d...

Zenith: Verified

4.2 High Risk

A total of 2 high risk findings were identified.

[H-1] Changes to `claimable_time_window` can lead to collisions of claimable accounts

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

Target

- programs/store/src/lib.rs#L591-L594 (`insert_amount`)
- programs/store/src/instructions/config.rs#L27-L35 (`unchecked_insert_amount`)
- programs/store/src/states/store.rs#L234-L240 (`get_amount_mut`)
- programs/store/src/states/store.rs#L581 (`get_mut`, `AmountKey::ClaimableTimeWindow`)
- programs/store/src/states/store.rs#L293-L313 (`claimable_time_key`, `claimable_time_window_index`, `claimable_time_window`)
- crates/programs/src/utils/store.rs#L50-L71 (`claimable_time_key`, `claimable_time_window_index`, `claimable_time_window`)

Description:

The protocol allows the `claimable_time_window` to be changed by a config keeper via the `insert_amount` instruction using the `AmountKey::ClaimableTimeWindow`. However, this `claimable_time_window` is indirectly used as part of the account seed for PDA derivation in 19 instances throughout the protocol using the store's `claimable_time_key` function, which derives the key via division of the given timestamp by the `claimable_time_window`.

Consequently, any changes to the `claimable_time_window` will break access to existing claimable accounts or lead to collisions with others in the worst case due to the altered PDA seeds. In case no claimable account is initialized at the given PDA address after a `claimable_time_window` change, the affected instruction will be subject to DoS.

Usage instances of *`claimable_time_key`*:

- crates/gmsol/src/exchange/order.rs#L675
- crates/gmsol/src/exchange/order.rs#L690

- crates/gmsol/src/exchange/order.rs#L705
- crates/gmsol/src/exchange/position_cut.rs#L237
- crates/sdk/src/client/ops/exchange/order.rs#L749
- crates/sdk/src/client/ops/exchange/order.rs#L764
- crates/sdk/src/client/ops/exchange/order.rs#L779
- crates/sdk/src/client/ops/exchange/order.rs#L1784
- programs/store/src/instructions/token.rs#L95
- programs/store/src/instructions/token.rs#L163
- programs/store/src/instructions/exchange/execute_order.rs#L713
- programs/store/src/instructions/exchange/execute_order.rs#L728
- programs/store/src/instructions/exchange/execute_order.rs#L743
- programs/store/src/instructions/exchange/execute_order.rs#L151
- programs/store/src/instructions/exchange/execute_order.rs#L152
- programs/store/src/instructions/exchange/execute_order.rs#L154
- programs/store/src/instructions/exchange/position_cut.rs#L179
- programs/store/src/instructions/exchange/position_cut.rs#L194
- programs/store/src/instructions/exchange/position_cut.rs#L209

Recommendations:

It is recommended to prohibit changes to the `claimable_time_window` or resolve the PDA derivation's indirect dependency.

GMX Solana: Fixed in @036a7ee...

Zenith: Verified.

[H-2] Price adjustments/validation use better-than-market prices for some oracles

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

Target

- programs/store/src/states/oracle/mod.rs
- programs/store/src/states/oracle/validator.rs

Description:

Some oracles, such as the ones Pyth provides, return a price along with a confidence bounds, which can be used to generate fair, symmetrical, bid and ask prices. The new price adjustment and validation code symmetrically caps the size of these confidence bounds, to ensure prices with very large variance are rejected. However, some oracles, such as the ones that Switchboard provides, return actual min/max values, and the price it provides is a _median_ price rather than the midpoint between the two prices. This is to prevent outliers from drastically affecting the reported price (e.g. due to manipulation on one exchange).

The adjustment and validation code assumes that the mid-price is the fair market price, and bases its deviation logic on that price, which may be far from the actual market price. In such cases, the adjustment will result in a price that is either above or below the actual fair market price, leading to some traders being able to fill at prices and volume not available at any other venue. If price adjustments are disabled for the market, the validation code will reject orders for periods when it should have allowed them, since the price is a fair price.

Recommendations:

Use the reported price rather than the mid-price when doing adjustments and validations, and consider only adjusting the side that falls out of bounds, rather than both prices.

GMX Solana: Resolved @2815d72267...

Zenith: Verified

4.3 Medium Risk

A total of 4 medium risk findings were identified.

[M-1] Competitions can be manipulated through many small-volume trades

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

Target

- programs/competition/src/instructions/trade_callback.rs

Description:

The competition mechanism is designed to extend when a high trade volume is detected, allowing more users to participate. However, a malicious user could bypass this by executing multiple trades just below the `volume_threshold` in the same block where the competition ends.

Since these trades would not individually trigger the extension, the attacker could accumulate enough volume to win the competition instantly.

This behavior would enable the attacker to unfairly win the competition and extract MEV at the expense of regular users.

Recommendations:

Consider modifying the competition logic to check the cumulative trade volume per block rather than individual trades.

GMX Solana: Fixed in @0f899ac... and @24f82a3...

In addition, if a user wishes to extend the competition duration as much as possible, they can indeed split up large orders into multiple smaller ones. If that's not their intention, they're also free to execute a single large trade. We prefer to preserve this flexibility for users.

Zenith: Verified.

[M-2] `update_order()` never checks for v2 callbacks

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

Target

- programs/store/src/instructions/exchange/order.rs

Description:

One of the invariants that is supposed to hold is as follows:

> If an action has a callback configured, invoking the callback is mandatory for all events except for close_action initiated by the owner.

While the `update_order_v2()` instruction does properly check for and execute the configured callback, the `update_order()` function does not.

Recommendations:

Add a header check, similar to the one done for the close action.

GMX Solana: Resolved @cbdbc0a

Zenith: Verified

[M-3] Order keepers can extract value through transaction ordering

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

Target

- crates/model/src/params/fee.rs
- crates/model/src/params/fee.rs
- crates/model/src/action/swap.rs

Description:

Order keepers are mostly trusted entities, but the protocol aims to stay secure even in the presence of some untrusted keepers. One case where the protocol will have problems is when there is a keeper outage affecting only the trusted order keepers. When such an outage occurs, malicious keepers can extract value by executing orders not by their submission time, but by whichever ordering maximizes gains on their own positions, for orders that occur within the same token validity window (typically 30 seconds). Some re-ordering scenarios at their disposal are:

- Ensuring that the kink in the kink model is triggered
- Ensuring that either the funding rate skew changes or doesn't
- Maximizing either the negative or the positive price impact for specific trades

Recommendations:

It is difficult to solve this issue without having an extremely frequent trusted-order-keeper heartbeat mechanism. An alternative, followed by the EVM version of GMX, is to document such behavior as a known issue.

GMX Solana: Acknowledged. The README will be updated to include a description of the known issues with keepers.

[M-4] `close_empty_claimable_account` can be abused for DoS and rent theft by malicious order keepers

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

Target

- programs/store/src/lib.rs#L1746-L1752
- programs/store/src/instructions/token.rs#L142-L197

Description:

The `close_empty_claimable_account` instruction can be abused by a malicious order keeper to immediately close freshly created claimable accounts that were just initialized by other (legitimate) keepers via the `use_claimable_account` instruction. Since new claimable accounts are empty (hold no tokens), they are eligible for closure right after creation. This leads to two main issues:

1. **Denial of Service (DoS):** Execution of decrease orders and position cuts can be disrupted, as these operations rely on the existence of claimable accounts. Malicious keepers can preemptively close these accounts, causing subsequent instructions to fail.
2. **Rent theft:** A malicious keeper can close claimable accounts created by others and claim the rent refund, effectively stealing rent from legitimate keepers who paid to create the accounts.

Recommendations:

It is recommended to restrict the ability to close claimable accounts to their creators (the keeper who initialized them). Alternatively, only allow claimable accounts to be closed if they are sufficiently old by checking their associated timestamp, preventing immediate closure after creation.

GMX Solana: Acknowledged.

Since claimable accounts are token accounts, there is no reliable way to track the creator. The current mitigation approach is for keepers to ensure that, in cases where the account remains empty, the creation and closure of the claimable account occur within the same transaction. This is feasible in practice and prevents rent theft in most situations.

In rare cases where a claimable account receives a balance, the user fully withdraws it, and a different keeper later closes the account, the rent may not return to the original creator. However, because such cases are infrequent, they can be handled off-chain by rewarding the creator. In an environment where most keepers are trusted, we consider this trade-off acceptable.

Regarding the DOS issue you mentioned, since claimable accounts are created using the idempotent instruction `use_claimable_account`, the keeper can simply insert this instruction before any instruction that requires a claimable account to avoid a DOS scenario.

4.4 Low Risk

A total of 11 low risk findings were identified.

[L-1] Competition participants can game volume metric

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

Target

- programs/competition/src/instructions/trade_callback.rs

Description:

The leaderboard of each competition is based off of the total volume across all orders, rather than being across buy/sell pairs:

```
        // Calculate volume as the absolute difference between after and
    before size_in_usd
        let volume = trade_event
            .after
            .size_in_usd
            .abs_diff(trade_event.before.size_in_usd);
...
        part.volume = part.volume.saturating_add(volume);
        part.last_updated_at = now;
```

This means that a trader that closes their position gets awarded twice the volume that a trader who leaves their position alone after opening it gets. If a trader is willing to incur position fees, the trader can rise to the top of the leaderboard by repeatedly opening and quickly closing very large positions (in order to reduce funding and borrowing fees). A malicious user could clear the leaderboard by creating five sock puppet accounts, and doing the same quick open/close pattern with each.

While it is possible to identify such users after the fact and to recover the true leaderboard, having to do so defeats the purpose of tracking it in the competition account.

Recommendations:

Only count volume associated with a position increase, and take into account the duration for which the position was open.

GMX Solana: Resolved @c25da04... Since the primary goal of this competition is to stimulate market trading volume, we do not plan to include rules related to position holding duration at this time.

Zenith: Verified

[L-2] Mutable `Config`/`Competition` accounts in all order instructions will lead to contention

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

Target

- programs/callback/src/instructions/action_callback.rs
- programs/competition/src/instructions/trade_callback.rs

Description:

The both the example callback program and the competition programs require (based on PDA constraints) the same mutable account to be passed to every action as the `Config`/`Competition`. This is not recommended because it reduces transaction throughput, due to the scheduler's priority algorithm, which can cause programs with high contention to have high transaction delays/failure rates, causing users to have to increase their priority fees.

A DEX with bursty transaction activity (e.g. during a bitcoin spike/crash) will end up having unnecessarily high transaction failure rates, leading to user dissatisfaction.

Recommendations:

Bin the `Config`/`Competition` into multiple PDAs based on the first few bits of each trader's address, and have separate instructions for periodic consolidation into `Config`/`Competition` results.

GMX Solana: Acknowledged. We may address the `Competition` contention when the GT rewards mechanism contention is addressed. The callback program is an example and we'd like to keep it simple.

[L-3] Calls to `initialize_config()` can be front-run

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

Target

- programs/callback/src/instructions/config.rs

Description:

The callback program's `Config` PDA uses hard-coded seeds as its derivation so once it's created, no others can be created by the same program. There are no authorization checks on the `initialize_config()` instruction, so once the program is deployed, a malicious user could front-run the initialization, potentially using a vulgar `prefix`.

Recommendations:

Require co-signing of the `initialize_config()` instruction by the callback program's keypair, which should only be known to the deployer:

```
pub struct InitializeConfig<'info> {
    /// Payer.
    #[account(mut)]
    pub payer: Signer<'info>,
    #[account(address = crate::ID)]
    pub init_authority: Signer<'info>,
    /// The [`Config`] account to initialize.
    #[account(
        init,
        payer = payer,
        space = 8 + Config::INIT_SPACE,
        seeds = [CONFIG_SEED],
        bump,
    )]
    pub config: Account<'info, Config>,
    /// The system program.
    pub system_program: Program<'info, System>,
}
```

GMX Solana: Resolved. For simplicity, we decided to remove `prefix` argument from the `initialize_config` instruction: @4151073...

Zenith: Verified

[L-4] Lack of uniqueness validation of `trade_event` in `OnExecuted` callback

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

Target

- programs/competition/src/instructions/trade_callback.rs#L115-L116

Description:

The `OnExecuted` context and the `invoke` function of the callback do not validate that the `trade_event` is unique and has not been used previously in the competition. This allows the same `trade_event` to be reused multiple times, potentially inflating a participant's volume or leaderboard ranking unfairly, undermining the integrity of the competition.

Note that this issue is partially mitigated since the signer is restricted to the callback authority of the `store` program. Nevertheless, the issue persists from the perspective of the `competition` program.

Recommendations:

It is recommended to add a mechanism to track processed `trade_event` IDs within the `competition` account to ensure uniqueness.

GMX Solana: Acknowledged.

To address this issue within the competition program, we would need to introduce an unbounded vector in the competition struct to store all trade events, which could result in significant overhead.

In that case, unless a more efficient solution is identified, we are inclined to add a comment to explicitly state the assumptions that the caller program must adhere to.

[L-5] Missing validation of association between `trade_event` and `trader/participant` in `OnExecuted` callback

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

Target

- programs/competition/src/instructions/trade_callback.rs#L115-L116

Description:

The `OnExecuted` context and the `invoke` function of the callback do not validate that the `trade_event` is associated with the `trader/participant`. This allows any `trade_event` to be used, potentially leading to incorrect updates to the participant's volume and leaderboard rankings. Without proper validation, malicious or unintended `trade_event` data could be used, compromising the integrity of the competition.

Note that this issue is partially mitigated since the signer is restricted to the callback authority of the `store` program. Nevertheless, the issue persists from the perspective of the `competition` program.

Recommendations:

It is recommended to add explicit validation to ensure that the `trade_event` is associated with the `trader/participant`.

GMX Solana: Fixed in @63a8b63...

Zenith: Verified.

[L-6] Improper use of `require_neq` macro for `Pubkey` comparison

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

Target

- programs/competition/src/instructions/participant_init.rs#L47

Description:

The `require_neq!` macro is being used to compare two `Pubkey` values (`trader` and `default_pubkey`). However, this macro is designed for general value comparisons and does not account for the specific semantics of `Pubkey` types. Anchor provides a dedicated macro, `require_keys_neq!`, which is specifically designed for comparing `Pubkey` values and ensures proper handling of `Pubkey` semantics.

Recommendations:

It is recommended to replace the `require_neq!` macro with `require_keys_neq!` to ensure proper handling of `Pubkey` comparisons:

```
require_keys_neq!(trader, default_pubkey);
```

GMX Solana: Fixed in @63a8b63...

Zenith: Verified.

[L-7] Competitions can expire instantly

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

Target

- programs/competition/src/instructions/competition_init.rs

Description:

The competition mechanism allows `start_time` to be set in the past, which could lead to unintended behavior (e.g., instant expiration). While `start_time < end_time` is enforced, the check does not ensure `start_time` is at least the current block timestamp.

Recommendations:

Consider ensuring that the start date is greater than the current timestamp.

GMX Solana: Fixed in @04bf173...

Zenith: Verified.

[L-8] Zero `max_deviation` in case of low-decimal `mid_price` can cause unexpected price validation failures

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

Target

- programs/store/src/states/oracle/validator.rs#L71-L92

Description:

The intermediate `max_deviation` value in the price validator can become zero if `mid_price * max_deviation_factor < 10^MARKET_DECIMALS` and `mid_price` has a low `decimal_multiplier`. This occurs due to integer division truncation in `apply_factor`. When `max_deviation` is zero, the subsequent rounding up in `with_unit_price` does not compensate for this, resulting in a zero deviation threshold. This can cause the deviation checks to fail unexpectedly, as only exact prices would be accepted.

Recommendations:

It is recommended to temporarily increase the precision of the deviation calculations to reduce the risk of truncation to zero, or enforce a minimum value for `max_deviation` (e.g., `max_deviation ⩾ 1`) before using it in deviation checks.

GMX Solana: Fixed in @8787324...

Zenith: Verified.

[L-9] Malicious keepers can deliberately cancel protocol operations by abusing the `throw_on_execution_error` flag

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

Target

- programs/store/src/lib.rs#L1836-L1874
- programs/store/src/lib.rs#L1940-L1978
- programs/store/src/lib.rs#L2268-L2336
- programs/store/src/lib.rs#L2338-L2402
- programs/store/src/lib.rs#L2404-L2460
- programs/store/src/lib.rs#L2462-L2517
- programs/store/src/lib.rs#L2727-L2772
- programs/store/src/lib.rs#L3425-L3489
- programs/store/src/lib.rs#L3580-L3640
- programs/store/src/lib.rs#L3729-L3791

Description:

A single malicious order keeper can deliberately cause deposits, withdrawals, order actions, shifts, and GLV actions to be cancelled by invoking the relevant instruction with the `throw_on_execution_error` flag set to `false` when an execution error is expected (e.g., due to temporary market volatility or other external conditions).

While these operations might only be temporarily unexecutable and could succeed if retried later, setting `throw_on_execution_error` to `false` causes the protocol to cancel the operation immediately. This forces affected users to recreate their operations, resulting in a minor DoS vector and impeded user experience.

Recommendations:

It is recommended to restrict the ability to set `throw_on_execution_error` to `false` to trusted or multisig keepers, or consider implementing a grace period mechanism before allowing cancellation of user operations.

GMX Solana: Acknowledged.

In practice, the keeper defaults to executing all actions with `throw_on_execution_error` set

to `false`.

This is intentional. We treat errors affected by `throw_on_execution_error` as non-temporary. While some of them may not strictly be non-temporary, our current strategy is to allow as many cancellations as possible and then adjust the scope of non-temporary errors based on real user experience. For certain errors that we do consider temporary, they will always throw unconditionally when triggered, such as most price oracle errors and trigger price-related errors.

[L-10] Precision loss in `with_max_deviation_factor` function due to division by `RATIO_MULTIPLIER`

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

Target

- crates/utils/src/token_config.rs#L319-L332

Description:

The `FeedConfig::with_max_deviation_factor` function divides the provided `max_deviation_factor` (`u128`) by `RATIO_MULTIPLIER` (`10^12`) to store it as a `u32` ratio (`max_deviation_ratio`). This conversion can silently truncate significant digits if the input factor is not an exact multiple of `RATIO_MULTIPLIER`, resulting in a loss of precision. When the factor is later reconstructed (multiplied by `RATIO_MULTIPLIER`), the original value may not be accurately restored, potentially allowing lower deviations than intended and causing unexpected validation failures.

Recommendations:

It is recommended to add explicit checks or warnings if the provided value is not an exact multiple of `RATIO_MULTIPLIER`, and document this precision limitation clearly. Where possible, store the full `u128` factor directly or reject values that would lose precision to avoid silent truncation.

GMX Solana: Fixed in @7ed45c2... Note that only the worst-case scenario (`max_deviation_ratio` becoming zero after division) is mitigated here.

Zenith: Verified.

[L-11] `try_adjust_price()` may return an incorrect result

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

Target

- programs/store/src/states/oracle/mod.rs

Description:

The `try_adjust_price()` function intends to adjust the price and say whether there was an adjustment or not. Presumably the intention here is to not update if there is an error during the adjustment, but there are multiple internal function calls that may error, causing `try_adjust_price_with_max_deviation_factor()` to return `false` even though `price.max` was updated.

Recommendations:

Convert `try_adjust_price_with_max_deviation_factor()` to atomically update.

GMX Solana: Resolved @59f4dfd...

Zenith: Verified

4.5 Informational

A total of 9 informational findings were identified.

[I-1] No way to close and reclaim rent for `ActionStats` and `Participant` accounts

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

Target

- programs/competition/src/instructions/participant_init.rs

Description:

The `ActionStats` and `Participant` accounts must be created by traders prior to being passed as accounts for the various callback instructions. They cost rent to store, but once the trader is done with them, there is no way to close the accounts and reclaim the rent.

Recommendations:

Provide instructions for closing the accounts

GMX Solana: Resolved @5ce0fc7..., @d6e20f1..., @6add407..., @695237f...

Zenith: Verified

[I-2] `InvalidTradeEvent` is never used

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

Target

- programs/competition/src/error.rs
- programs/competition/src/instructions/trade_callback.rs

Description:

The `InvalidTradeEvent` error is never used:

```
#[msg("Invalid trade event")]
InvalidTradeEvent,
```

Recommendations:

```
let trade_event = trade_event.load()?;
let trade_event = trade_event.load()
    .map_err(|_| CompetitionError::InvalidTradeEvent)?;
```

GMX Solana: Resolved @63a8b63...

Zenith: Verified

[I-3] No way to deactivate a `Competition`

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

Target

- programs/competition/src/instructions/competition_init.rs
- programs/competition/src/instructions/trade_callback.rs
- programs/competition/src/instructions/trade_callback.rs

Description:

The `Competition` account contains an `is_active` field that is unconditionally set to `true` in the `initialize_competition()` instruction, and thereafter its value never changes. However, the field still costs rent, and the other instructions still expend CUs to check that it's unchanged:

```
fn validate_competition(&self) -> Result<()> {
    let now = Clock::get()?.unix_timestamp;
    let comp = &self.competition;
    require!(comp.is_active, CompetitionError::CompetitionNotActive);
```

```
impl OnExecuted<'_> {
    /// Core entry point called by the store program.
    pub(crate) fn invoke(
...
        if !comp.is_active {
            msg!("competition: the competition is not active");
            return Ok(()); }
```

Recommendations:

Create an instruction for changing the state of the flag, or remove the flag since it is currently redundant, given anchor's use of discriminators

GMX Solana: Resolved @edc9dd9... @0f899ac...

Zenith: Verified

[I-4] Typos

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

Target

- programs/callback/**/*.rs
- programs/store/**/*.rs

Description:

The following typos were identified:

`pending -> action_stats`

```
pub const ACTION_STATS_SEED: &[u8] = b"pending";
```

`procotol -> protocol`

```
//! This crate defines the callback interface for the GMX-Solana procotol
```

`collatearl -> collateral`

```
/// Initial collatearl token vault.
```

`requried -> required`

```
/// Only requried by increase and swap orders.
```

`hanlde_updated -> handle_updated`

```
fn hanlde_updated(&self) -> Result<()> {
```

`claimble -> claimable`

```
//! - [`close_empty_claimable_account`]: Close a empty claimble account.
```

ConfigurateGt -> ConfigureGt

```
ctx: Context<ConfigurateGt>,
```

Unintialized -> Uninitialized

```
///   - Unintialized ATA account of `market_token` owned by `glv`
```

defualt -> default

```
/// Non-defualt store is not allowed.
```

prcie -> price

```
/// Max prcie's timestamp exceeded.
```

diabled -> disabled

```
/// Token config is diabled.
```

emtpy - > empty

```
#[msg("emtpy withdrawal")]
```

togather -> together

```
/// but the token amounts are not merged togather.
```

derving -> deriving

```
doc = "Set the creator of this order. CHECK: It must be the address derving
    the order account",
```

liqudiation -> liquidation

```
// Validate the liqudiation is a fully close.
```

discrimator -> discriminator

```
// Make sure the discrimator is written to the account data.
```

vaule -> value

```
/// Returns the previous vaule.
```

Extentsion -> Extension

```
/// Extentsion trait for [`Action`].
```

MAREKT_TOKEN_MINT_SEED -> MARKET_TOKEN_MINT_SEED

```
pub const MAREKT_TOKEN_MINT_SEED: &[u8] = b"market_token_mint";
```

cliamable -> claimable

```
/// Short token amount for cliamable account of user.
```

Accetpable -> Acceptable

```
/// Accetpable price (in unit price).
```

receving -> receiving

```
/// The escrow account for receving market tokens.
```

receving -> receiving

```
/// The ATA of the owner for receving market tokens.
```

collatearl -> collateral

```
/// Initial collatearl token vault.
```

deifinition -> definition

```
/// The accounts deifinition for the
    [`execute_withdrawal`](crate::gmsol_store::execute_withdrawal)
```

`receving -> receiving`

```
/// The escrow account for receving market tokens to burn.
```

`withdrawed -> withdrawn`

```
/// The escrow account for receiving withdrawed final long tokens.
```

`withdrawed -> withdrawn`

```
/// The escrow account for receiving withdrawed final short tokens.
```

`collatearl -> collateral`

```
/// Initial collatearl token vault.
```

`instrcution -> instruction`

```
/// CHECK: only ORDER_KEEPER is allowed to use this instrcution.
```

`owenr -> owner`

```
/// The owenr of the shift.
```

`receving -> receiving`

```
/// The escrow account for receving market tokens to burn.
```

`withdrawed -> withdrawn`

```
/// The escrow account for receiving withdrawed final long tokens.
```

`withdrawed -> withdrawn`

```
/// The escrow account for receiving withdrawed final short tokens.
```

`recevier -> receiver`

```
/// The recevier of the deposit.
```

sigenr -> signer

```
sigenr: &ActionSigner,
```

thte -> the

```
/// Update thte config of GLV.
```

defintion -> definition

```
/// The accounts defintion for
    [`create_glv_withdrawal`](crate::create_glv_withdrawal) instruction.
```

defintion -> definition

```
/// The accounts defintion for
    [`close_glv_withdrawal`](crate::gmsol_store::close_glv_withdrawal)
    instruction.
```

wihtdrawal -> withdrawal

```
/// Market token wihtdrawal vault.
```

defintions -> definitions

```
/// The accounts defintions for the
    [`initialize_gt`](crate::gmsol_store::initialize_gt) instruction.
```

defintions -> definitions

```
/// The accounts defintions for GT configuration instructions.
```

ConfigurateGt -> ConfigureGt

```
pub struct ConfigurateGt<'info> {
```

udpate -> update

```
/// - Only MARKET_KEEPER can udpate the config of market.
```

set_is_gt_minting_enbaled -> set_is_gt_minting_enabled

```
.set_is_gt_minting_enbaled(enable);
```

magament -> management

```
/// Instructions for config magament.
```

funtionality -> functionality

```
/// Instructions for the exchange funtionality.
```

adjustemnt -> adjustment

```
/// Get timestamp adjustemnt of the given token.
```

claimble -> claimable

```
/// The claimble account.
```

otken -> token

```
/// Initial short otken amount to deposit.
```

perfrom_deposit -> perform_deposit

```
fn perfrom_deposit(self) -> Result<()> {
```

otken -> token

```
/// Initial short otken amount to deposit.
```

maount -> amount

```
/// Minimum acceptable maount of market tokens to be minted.
```

invertbile -> invertible

```
/// This is an invertbile operation that will panic on error.
```

valut -> vault

```
/// Operation for transferring funds into market valut.
```

vaule -> value

```
/// Returns the previous vaule.
```

Extentsion -> Extension

```
/// Extentsion trait for [`Action`].
```

loders -> loaders

```
/// Create a new [`SwapMarkets`] from loders.
```

set_is_gt_minting_enbaled -> set_is_gt_minting_enabled

```
pub fn set_is_gt_minting_enbaled(&mut self, enabled: bool) -> bool {
```

avlue -> value

```
/// Pool avlue without pnl for short.
```

instrcutions -> instructions

```
/// the oracle through instrcutions.
```

decimal_multipler -> decimal_multiplier

```
decimal_multipler: u8,
```

Fisrt -> First

```
/// Fisrt Deposit Receiver Seed.
```

vaule -> value

```
/// Get min shift vaule.
```

uninitalized -> uninitialized

```
.expect("uninitalized GLV Deposit account")
```

uninitalized -> uninitialized

```
.expect("uninitalized GLV Withdrawal account")
```

storted -> sorted

```
// Ranks must be storted.
```

storted -> sorted

```
// Factors must be storted.
```

Intiailized -> Initialized

```
self.flags.get_flag(GtExchangeVaultFlag::Intiailized)
```

Comfirmed -> Confirmed

```
self.flags.get_flag(GtExchangeVaultFlag::Comfirmed)
```

cliamable -> claimable

```
/// Short token amount for cliamable account of user.
```

Recevier -> Receiver

```
/// Recevier Kind.
```

validatoin -> validation

```
// so validatoin is not required. In fact, we should prohibit the creation
    of limit swap orders
```

accetable -> acceptable

```
/// Get accetable price (unit price).
```

Unitialized -> Uninitialized

```
/// - `kind` is `Unitialized`.
```

`enbaled -> enabled`

```
/// Is enbaled.
```

`msut -> must`

```
self.from_market_token.account().expect("msut exist")
```

`msut -> must`

```
self.to_market_token.account().expect("msut exist")
```

`mutablely -> mutably`

```
// Check the account for mutablely access.
```

`try_deserailize_unchecked -> try_deserialize_unchecked`

```
let header =
    Arc::new(de::try_deserailize_unchecked::<TokenMapHeader>(buf)?);
```

`initilaized -> initialized`

```
/// Return `false` if the transfer is required but the ATA is not
    initilaized.
```

Recommendations:

We recommend fixing the typos.

GMX Solana: Resolved @1b3ad47... @3fca231...

Zenith: Verified

[I-5] `create_order()/update_order()` do not emit event like the v2 versions do

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

Target

- programs/store/src/instructions/exchange/order.rs
- programs/store/src/instructions/exchange/order.rs
- programs/store/src/instructions/exchange/order.rs

Description:

Most of the order-related instructions already emitted events when state changed, and the emission code was added without needing to create new versions of the instructions. With the addition of the callback code, the `update_order_v2()` instruction had code added to it to emit unconditionally. The `update_order()` instruction is missing the same functionality, leading to an undocumented inconsistency between the two versions. The `create_order()` instruction has a similar discrepancy.

Recommendations:

Document the difference in code comments, since the emission can't be added without changing the accounts required by existing callers of the instructions.

GMX Solana: Resolved @e5d4979...

Zenith: Verified

[I-6] Chainlink oracle logic is unimplemented

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

Target

- programs/store/src/states/oracle/chainlink.rs

Description:

The entire logic for Chainlink's oracle was commented out, so any tokens that use this oracle will not work correctly (if it's specified in the token config), resulting in an unexpected revert.

Recommendations:

Consider reimplementing the oracle logic before deploying the program. Alternatively, consider removing the entire Chainlink logic.

GMX Solana: Fixed in @89b1a2d...

Zenith: Verified.

[I-7] The `GtBank` struct has changed without version-based handling

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

Target

- programs/treasury/src/states/gt_bank.rs#L15-L28 (new)
- programs/gmsol-treasury/src/states/gt_bank.rs#L14-L26 (old)

Description:

The `GtBank` struct has changed since the last audit. Notably, the size of the `reserved` field was reduced, and a new field was added: `receiver_vault_out`. However, the `version` field present in the struct is never used throughout the codebase to distinguish between different struct versions.

Recommendations:

Although the current changes seemingly integrate smoothly, it is still recommended to implement logic throughout the codebase to check and handle the `version` field, ensuring correct parsing and handling of different `ActionHeader` versions.

GMX Solana: Acknowledged.

The `version` field is reserved for future breaking changes where we want to keep the account discriminator unchanged. For non-breaking structural updates, we generally don't see a need to change the `version`. We ensure new fields have safe default values of zeros, so reserved bytes can be treated as those fields and won't cause breaking changes. In SDKs, such changes typically result in a major version bump to clearly indicate potential breaking changes.

[I-8] The `ActionHeader` struct has changed without version-based handling

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

Target

- programs/store/src/states/common/action.rs#L21-L61 (new)
- programs/gmsol-store/src/states/common/action.rs#L17-L49 (old)

Description:

The `ActionHeader` struct has changed since the last audit. Notably, the sizes of the `padding_0` and `reserved` fields were reduced, and new fields were added: `callback_kind`, `callback_version`, `callback_program_id`, `callback_config`, and `callback_action_stats`. However, the `version` field present in the struct is never used throughout the codebase to distinguish between different struct versions.

Recommendations:

Although the current changes seemingly integrate smoothly, it is still recommended to implement logic throughout the codebase to check and handle the `version` field, ensuring correct parsing and handling of different `ActionHeader` versions.

GMX Solana: Acknowledged.

The `version` field is reserved for future breaking changes where we want to keep the account discriminator unchanged. For non-breaking structural updates, we generally don't see a need to change the `version`. We ensure new fields have safe default values of zeros, so reserved bytes can be treated as those fields and won't cause breaking changes. In SDKs, such changes typically result in a major version bump to clearly indicate potential breaking changes.

[I-9] Unresolved TODO and FIXME comments

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

Description:

The following instances of unresolved TODO and FIXME comments were discovered in the codebase:

TODO

1. crates/gmsol/src/exchange/treasury.rs#L11

   implement this.

FIXME

1. crates/gmsol/src/exchange/treasury.rs#L59

   read program id from the market.

2. crates/gmsol/src/store/token_config.rs#L75

   reduce the number of args.

3. crates/sdk/src/client/ops/market.rs#L537

   read program id from the market.

4. crates/sdk/src/client/ops/token_config.rs#L31

   reduce the number of args.

Recommendations:

We recommended resolving, removing, or acknowledging (keeping as normal comments) the above instances.

GMX Solana: Fixed in @0928580...

Zenith: Verified.