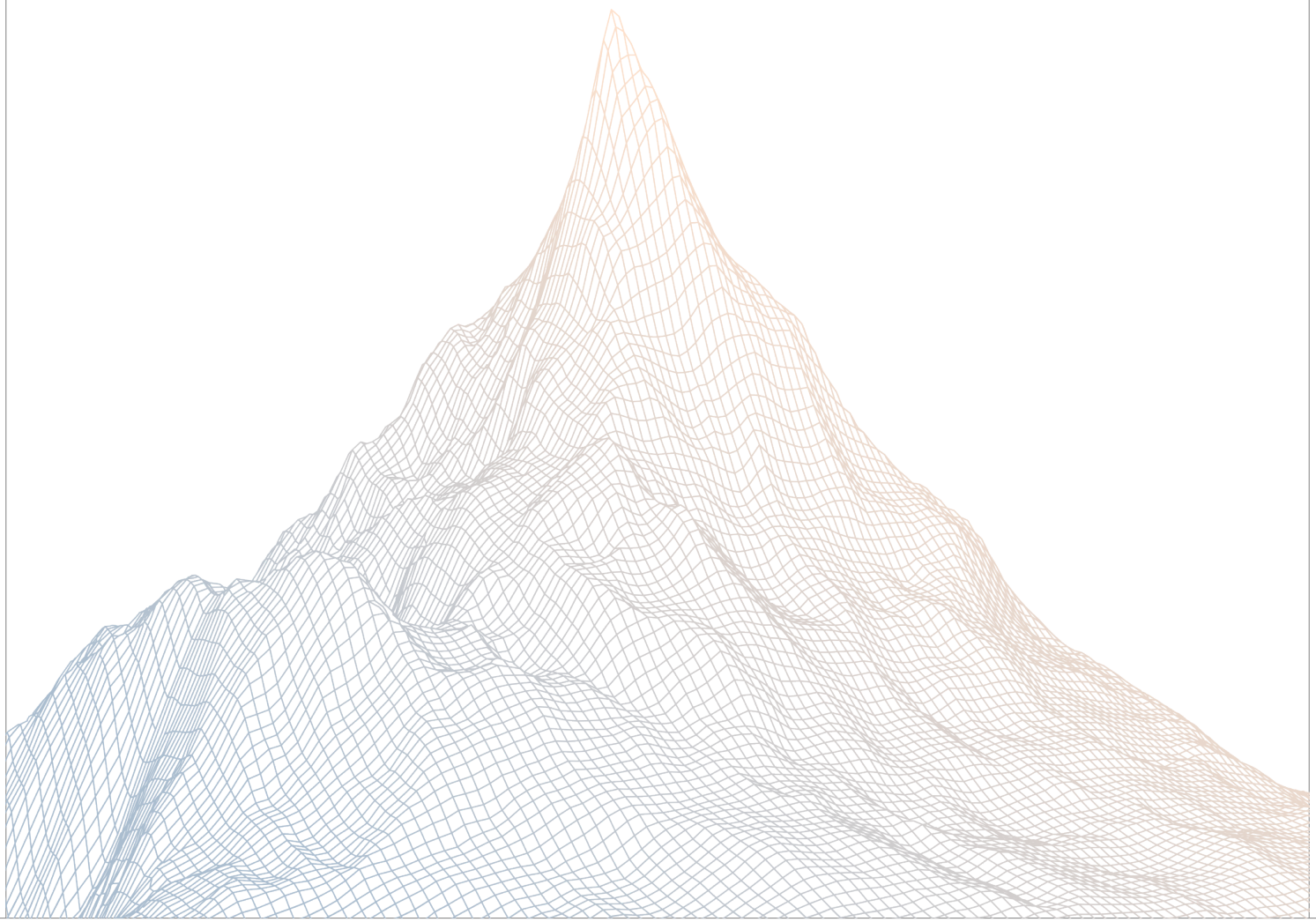
Zenith

GMX Solana

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: January 15th to January 20th, 2026

AUDITED BY: IIIIIII
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	gmsol-labs/gmx-solana#304
Repository	https://github.com/gmsol-labs/gmx-solana/pull/304
Commit Hash	`595a070d5fff2a00e33dcf46efa2223bc60df221`
Files	`Changes in PR-304`

Target	gmsol-labs/gmx-solana#305
Repository	https://github.com/gmsol-labs/gmx-solana/pull/305
Commit Hash	`20acb01b1cca547d0a19c56a03a076767f419167`
Files	`Changes in PR-305`

Target	gmsol-labs/gmx-solana#306
Repository	https://github.com/gmsol-labs/gmx-solana/pull/306
Commit Hash	`939c1ab4d548586df3886f3dc5d90fdbc3d9037e`
Files	`Changes in PR-306`

Target	GMX Solana Mitigation Review
Repository	https://github.com/gmsol-labs/gmx-solana
Commit Hash	`6a5e6b24d72fcd216f8314db4ef651c575e7fa72`

Files	`Changes in PR-304`
	`Changes in PR-305`
	`Changes in PR-306`

2.3 Audit Timeline

January 15, 2026	Audit start
January 20, 2026	Audit end
January 27, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	3
Informational	2
Total Issues	**5**

3

Findings Summary

ID	Description	Status
L-1	Insufficient precision may be used if the price suddenly drops	Acknowledged
L-2	Malicious keeper can cancel valid Increase orders to earn fees	Resolved
L-3	Inconsistent doc and implementation for update_price_feed_with_chainlink_idempotent()	Resolved
I-1	Misleading comments	Resolved
I-2	Missing error message	Resolved

4

Findings

4.1 Low Risk

A total of 3 low risk findings were identified.

[L-1] Insufficient precision may be used if the price suddenly drops

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

Target

- crates/utils/src/oracle.rs
- crates/utils/src/price/decimal.rs

Description:

The precision used when consuming the various oracles' prices is solely determined by the keeper-configured token configuration, which includes the token's decimals as well as the desired precision. While market keepers are expected to track price changes and update the configuration's precision, it is possible for a sudden price change to occur before the keepers can update the configuration, or for the bid and ask to require different precision values. There currently is a change to prevent large jumps in price from overflowing the internal decimal representation, but there is no such protection on the down side to prevent excessive precision loss.

For example, if the price of ETH is ~$450k with a precision of 2 but then drops to $4k, a more correct precision would be 3, but the configuration might not be updated quickly enough, or this may be due to the oracle's confidence interval:

```
#[test]
fn precision_stuck_after_price_drop() {
    let token_decimals = 17;
    let decimals = 18;

    // Large ETH price forces low precision
    let high_price = 450_000u128 * 10u128.pow(18);
    let low_precision = 2;
```

```
    let d_high = Decimal::try_from_price(high_price, decimals,
    token_decimals, low_precision).unwrap();
    assert_eq!(d_high.to_unit_price(), 450000000);

    // price drops, but precision is not yet increased
    //let dropped_price = 4_321u128 * 10u128.pow(18);
    let dropped_price = 4_3216789u128 * 10u128.pow(18 - 4);
    let d_stuck = Decimal::try_from_price(dropped_price, decimals,
    token_decimals, low_precision).unwrap();

    // used with only 2 decimals, even though more are possible
    //assert_eq!(d_stuck.value, 4_321 * 1_00);
    assert_eq!(d_stuck.value, 4_32167);

    // what we *could* have used with higher precision
    let high_precision = 3;
    let d_better = Decimal::try_from_price(dropped_price, decimals,
    token_decimals, high_precision).unwrap();

    //assert_eq!(d_better.value, 4_321 * 1_000);
    assert_eq!(d_better.value, 4_321678);

    // could have used more precision, but didn't
    assert!(d_better.value > d_stuck.value * 10);
    assert_eq!(d_stuck.to_unit_price(),  4321670);
    assert_eq!(d_better.to_unit_price(), 4321678);
}
```

Recommendations:

Introduce a function to check whether there was space for more precision, and reject prices where there was space for two more decimal places of precision:

```
impl Decimal {
    /// Returns true if this decimal could represent more precision for
    `token_decimals`
    /// by up to `places` additional decimal digits without overflowing
    `u32`.
    pub fn has_unused_precision_capacity(&self, token_decimals: u8, places:
    u8) → bool {
        let used_precision = Decimal::MAX_DECIMALS
            .checked_sub(token_decimals)
            .checked_sub(self.decimal_multiplier);

        token_decimals + used_precision + places ⩽ Decimal::MAX_DECIMALS
```

```
            && 10u128.checked_pow(places as u32)
                .map_or(false, |factor| (self.value
    as u128).checked_mul(factor) ≤ u32::MAX as u128)
    }
}
```

GMX: Acknowledged. A larger price storage type (such as u64) will be considered in the future to further reduce the likelihood of this kind of issue occurring.On the other hand, in scenarios involving sharp price drops, rejecting prices due to precision loss could prevent liquidations from being executed properly, which may introduce a greater risk than minor precision loss itself.

[L-2] Malicious keeper can cancel valid Increase orders to earn fees

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

Target

- /programs/store/src/instructions/exchange/execute_order.rs#L200
- order.rs#L1048-L1051

Description:

The `event` account for `execute_increase_or_swap_order_v2` is marked `Option`, as it is only required for Increase orders and not for swap orders.

```
pub struct ExecuteIncreaseOrSwapOrderV2<'info> {
...
/// Trade event buffer.
    #[account(mut, has_one = store, has_one = authority)]
    pub event: Option<AccountLoader<'info, TradeData>>,
```

However, `execute_increase_or_swap_order_v2` does not ensure that the keeper (caller) provides the `event` account for Increase orders.

If the keeper provides `None`, it will cause `ExecuteOrderOperation.perform_execution()` to fail and throw `CoreError::EventBufferNotProvided`.

And by setting `throw_on_execution_error = false`, it will mark the order as cancelled and keeper receives the execution fee despite being a valid order.

```
fn perform_execution(
...
            let event_loader = self
                .event
                .as_ref()
                .ok_or(error!(CoreError::EventBufferNotProvided))?;
```

Recommendations:

Update `execute_increase_or_swap_order_v2` to validate that `ctx.accounts.event.is_some()` for Increase order.

GMX Solana: Resolved with `@d76267940b ...`

Zenith: Verified. Resolved by setting `should_throw_error = true` when the `event` account is `None`, which will propagate the error and revert the transaction.

[L-3] Inconsistent doc and implementation for `update_price_feed_with_chainlink_idempotent()`

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

Target

- programs/store/src/lib.rs#L1265
- programs/store/src/states/oracle/feed.rs#L94-L96

Description:

In the doc comments, it states the following for `update_price_feed_with_chainlink_idempotent`.

> Returns `Ok(false)` if the price report is valid but not newer than the last update.

If the price report is 'not newer', it translate to `!(price_ts > last_price_ts)`, which is equivalent to `(price_ts ⩽ last_price_ts)`.

However, the implementation checks for `(price_ts < last_price_ts)`, which is inconsistent with the docs.

```
if idempotent && price_ts < last_price_ts {
    return Ok(false);
}
```

Recommendations:

Consider updating the doc or the implementation to be consistent for both.

GMX Solana: Resolved with `@c08a4b2b14 ...`

Zenith: Verified. Resolved by updating the doc.

4.2 Informational

A total of 2 informational findings were identified.

[I-1] Misleading comments

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

Target

- programs/store/src/lib.rs
- programs/store/src/states/position.rs

Description:

The following comments are misleading, which adds extra cognitive load to reviews:

- The rustdocs for `update_price_feed_with_chainlink_idempotent()` state that the function returns `false` if the "price report is valid but not newer than the last update". However, not all of the checks are performed once the report is determined to be stale, including checks to verify that the market is not crossed. If future code relies on a non-error to indicate a valid report, it will improperly use an invalid report. All checks should be performed, and only the updates to storage should be skipped:

```
/// - Returns `Ok(true)` if the price feed is updated.
/// - Returns `Ok(false)` if the price report is valid but not newer than the
    last update.
```

- Recent changes to `validated_meta_with_options()` changed what conditions lead to invalid positions. The error should read "invalid, closed, or disabled market":

```
let meta = market
    .validated_meta_with_options(&self.store, allow_closed)
    .map_err(|_| gmsol_model::Error::InvalidPosition("invalid or disabled
    market"))?;
```

Recommendations:

Update the comments to reflect the current behavior, or change the code to match the behavior in the comment.

GMX: Resolved with `@b0788c355d ...` and `@6fede805d1 ...`

Zenith: Verified.

[I-2] Missing error message

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

Target

- programs/store/src/lib.rs

Description:

The `FailedToCalculateMarketTokenAmountToBurn` error code is the only `CoreError` that is missing the use of the `#[msg()]` attribute macro. This may lead to miss-handling of such errors by clients expecting all errors to have error messages.

Recommendations:

```
/// Failed to calculate market token amount to burn.
#[msg("failed to calculate market token amount to burn")]
FailedToCalculateMarketTokenAmountToBurn,
```

GMX: Resolved with `@1dcd885d10 ...`

Zenith: Verified.