SHERLOCK

Security Review For GMX-Solana

Collaborative Audit Prepared For:	GMX-Solana
Lead Security Expert(s):	IIIIIII
	gjaldon
	KupiaSec
	bin2chen
	J4X
Date Audited:	November 6th - December 4th
Final Commit:	d5ceeb8f28edfbce8cf4a363ef20f98cdd865708

Introduction

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

Scope

Repository: https://github.com/gmsol-labs/gmx-solana

Commit: 1fc1ed8b44a99fbdfdea503a0e636f0890550135

Contracts:

- crates/gmsol-model/Cargo.toml
- crates/gmsol-model/src/action/decrease_position/claimable.rs
- crates/gmsol-model/src/action/decrease_position/collateral_processor.rs
- crates/gmsol-model/src/action/decrease_position/mod.rs
- crates/gmsol-model/src/action/decrease_position/report.rs
- crates/gmsol-model/src/action/decrease_position/utils.rs
- crates/gmsol-model/src/action/deposit.rs
- crates/gmsol-model/src/action/distribute_position_impact.rs
- crates/gmsol-model/src/action/increase_position.rs
- crates/gmsol-model/src/action/mod.rs
- crates/gmsol-model/src/action/swap.rs
- crates/gmsol-model/src/action/update_borrowing_state.rs
- crates/gmsol-model/src/action/update_funding_state.rs
- crates/gmsol-model/src/action/withdraw.rs
- crates/gmsol-model/src/bank.rs
- crates/gmsol-model/src/clock.rs
- crates/gmsol-model/src/error.rs
- crates/gmsol-model/src/fixed.rs
- crates/gmsol-model/src/lib.rs
- crates/gmsol-model/src/market/base.rs
- crates/gmsol-model/src/market/borrowing.rs
- crates/gmsol-model/src/market/liquidity.rs
- crates/gmsol-model/src/market/mod.rs

- crates/gmsol-model/src/market/perp.rs
- crates/gmsol-model/src/market/position_impact.rs
- crates/gmsol-model/src/market/swap.rs
- crates/gmsol-model/src/num.rs
- crates/gmsol-model/src/params/fee.rs
- crates/gmsol-model/src/params/mod.rs
- crates/gmsol-model/src/params/position.rs
- crates/gmsol-model/src/params/price_impact.rs
- crates/gmsol-model/src/pool/balance.rs
- crates/gmsol-model/src/pool/delta.rs
- crates/gmsol-model/src/pool/mod.rs
- crates/gmsol-model/src/position.rs
- crates/gmsol-model/src/price.rs
- crates/gmsol-model/src/test.rs
- crates/gmsol-model/src/utils.rs
- crates/gmsol-utils/Cargo.toml
- crates/gmsol-utils/src/fixed_map.rs
- crates/gmsol-utils/src/init_space.rs
- crates/gmsol-utils/src/lib.rs
- crates/gmsol-utils/src/price/decimal.rs
- crates/gmsol-utils/src/price/mod.rs
- programs/gmsol-store/Cargo.toml
- programs/gmsol-store/Clippy.toml
- programs/gmsol-store/Xargo.toml
- programs/gmsol-store/src/constants/market.rs
- programs/gmsol-store/src/constants/mod.rs
- programs/gmsol-store/src/events.rs
- programs/gmsol-store/src/instructions/bug_fix.rs
- programs/gmsol-store/src/instructions/config.rs
- programs/gmsol-store/src/instructions/exchange/deposit.rs
- programs/gmsol-store/src/instructions/exchange/execute_deposit.rs

- programs/gmsol-store/src/instructions/exchange/execute_order.rs
- programs/gmsol-store/src/instructions/exchange/execute_shift.rs
- programs/gmsol-store/src/instructions/exchange/execute_withdrawal.rs
- programs/gmsol-store/src/instructions/exchange/mod.rs
- programs/gmsol-store/src/instructions/exchange/order.rs
- programs/gmsol-store/src/instructions/exchange/position_cut.rs
- programs/gmsol-store/src/instructions/exchange/shift.rs
- programs/gmsol-store/src/instructions/exchange/update_adl.rs
- programs/gmsol-store/src/instructions/exchange/withdrawal.rs
- programs/gmsol-store/src/instructions/feature.rs
- programs/gmsol-store/src/instructions/glv/deposit.rs
- programs/gmsol-store/src/instructions/glv/management.rs
- programs/gmsol-store/src/instructions/glv/mod.rs
- programs/gmsol-store/src/instructions/glv/shift.rs
- programs/gmsol-store/src/instructions/glv/withdrawal.rs
- programs/gmsol-store/src/instructions/gt.rs
- programs/gmsol-store/src/instructions/market.rs
- programs/gmsol-store/src/instructions/mod.rs
- programs/gmsol-store/src/instructions/oracle/mod.rs
- programs/gmsol-store/src/instructions/oracle/price_feeds.rs
- programs/gmsol-store/src/instructions/roles.rs
- programs/gmsol-store/src/instructions/store.rs
- programs/gmsol-store/src/instructions/token.rs
- programs/gmsol-store/src/instructions/token_config.rs
- programs/gmsol-store/src/instructions/user.rs
- programs/gmsol-store/src/lib.rs
- programs/gmsol-store/src/ops/deposit.rs
- programs/gmsol-store/src/ops/execution_fee.rs
- programs/gmsol-store/src/ops/glv.rs
- programs/gmsol-store/src/ops/market.rs
- programs/gmsol-store/src/ops/mod.rs

- programs/gmsol-store/src/ops/order.rs
- programs/gmsol-store/src/ops/shift.rs
- programs/gmsol-store/src/ops/withdrawal.rs
- programs/gmsol-store/src/states/common/action.rs
- programs/gmsol-store/src/states/common/mod.rs
- programs/gmsol-store/src/states/common/swap.rs
- programs/gmsol-store/src/states/common/token.rs
- programs/gmsol-store/src/states/common/token_with_feeds.rs
- programs/gmsol-store/src/states/deposit.rs
- programs/gmsol-store/src/states/feature.rs
- programs/gmsol-store/src/states/glv.rs
- programs/gmsol-store/src/states/gt.rs
- programs/gmsol-store/src/states/market/clock.rs
- programs/gmsol-store/src/states/market/config.rs
- programs/gmsol-store/src/states/market/mod.rs
- programs/gmsol-store/src/states/market/model.rs
- programs/gmsol-store/src/states/market/pool.rs
- programs/gmsol-store/src/states/market/revertible/buffer.rs
- programs/gmsol-store/src/states/market/revertible/liquidity_market.rs
- programs/gmsol-store/src/states/market/revertible/market.rs
- programs/gmsol-store/src/states/market/revertible/mod.rs
- programs/gmsol-store/src/states/market/revertible/revertible_position.rs
- programs/gmsol-store/src/states/market/revertible/swap_market.rs
- programs/gmsol-store/src/states/market/status.rs
- programs/gmsol-store/src/states/market/utils.rs
- programs/gmsol-store/src/states/mod.rs
- programs/gmsol-store/src/states/oracle/chainlink.rs
- programs/gmsol-store/src/states/oracle/mod.rs
- programs/gmsol-store/src/states/oracle/price_map.rs
- programs/gmsol-store/src/states/oracle/pyth.rs
- programs/gmsol-store/src/states/oracle/time.rs

- programs/gmsol-store/src/states/oracle/validator.rs
- programs/gmsol-store/src/states/order.rs
- programs/gmsol-store/src/states/position.rs
- programs/gmsol-store/src/states/roles.rs
- programs/gmsol-store/src/states/shift.rs
- programs/gmsol-store/src/states/store.rs
- programs/gmsol-store/src/states/token_config.rs
- programs/gmsol-store/src/states/user.rs
- programs/gmsol-store/src/states/withdrawal.rs
- programs/gmsol-store/src/utils/chunk_by.rs
- programs/gmsol-store/src/utils/cpi.rs
- programs/gmsol-store/src/utils/de.rs
- programs/gmsol-store/src/utils/dynamic_access.rs
- programs/gmsol-store/src/utils/fixed_str.rs
- programs/gmsol-store/src/utils/internal/action.rs
- programs/gmsol-store/src/utils/internal/authentication.rs
- programs/gmsol-store/src/utils/internal/mod.rs
- programs/gmsol-store/src/utils/internal/transfer.rs
- programs/gmsol-store/src/utils/mod.rs
- programs/gmsol-store/src/utils/pubkey.rs
- programs/gmsol-store/src/utils/token.rs

Final Commit Hash

https://github.com/gmsol-labs/gmx-solana/commit/d5ceeb8f28edfbce8cf4a363ef20f98cdd865708

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
6	11	13

Issues Not Fixed or Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Borrowing factors are not checkpointed when liquidity is deposited/withdrawn

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/276

Summary

The fee for borrowing depends, in part, on how much liquidity is available in the liquidity pool. When changes are made to the pool's balances, there is no checkpoint made of the new value, which means positions will be under- or over-charged borrowing fees, depending on whether liquidity was withdrawn or provided.

Vulnerability Detail

When a position increase/decrease operation is performed, both the funding and borrowing factors are checkpointed, so positions can calculate how much they owe since the last update, and can save the current factor for the next time the position is modified.

The calculation of the borrowing factor is based, in part, on how much liquidity is available in the liquidity pool. If there are changes to the pool's value, a new factor should be calculated and stored, which currently is not done.

Impact

An attacker trying to systematically under-pay borrowing fees could monitor their market for liquidity and position changes, and any time there's a deposit to the liquidity pool that raises it to a level which is higher than it was during the last checkpoint (i.e. when some other trader changed their position last), they would create a new checkpoint by updating their position in some negligible way (e.g. adding/removing small amounts of collateral). In this way, the attacker can reduce the amount of time known to the system, where borrowing rates are high, at the expense of minimal transaction fees.

Conversely, an LP trying to maximize their borrow fees would trigger checkpoints any time the liquidity becomes lower than it was the last time there was a position change.

Code Snippet

There are no calls to `update_funding(...).execute()`:

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/deposit.rs#L280-L284 https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/withdraw.rs#L100-L103

Tool used

Manual Review

Recommendation

We recommend checkpointing funding and borrowing factors every time funds are deposited/withdrawn.

Discussion

IllIllI000

In the increase code outside of the model, the swapping of collateral happens prior to the checkpointing of funding and borrowing factors, so it's possible for someone to use swaps to unfairly alter the borrowing rate by including a swap. The increase/decrease code should change to pull out the funding/borrowing checkpoints from the model and the position impact pool should be distributed too, as has been done for the new deposit/withdraw code.

IllIllI000

Fix confirmed

Issue H-2: Flipped price maximization during position decrease makes price manipulation easier, and markets break down

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/269

Summary

Because entries and exits both use the maximized price rather than using flipped maximization directions, it's much easier to manipulate the Pyth oracle, using only a couple of exchanges, than it should be. The ability to enter an exit without having to pay costs in the form of a spread, without there being actual deep liquidity at those prices, will lead to markets breaking down.

Vulnerability Detail

When decreasing a position, the execution price chosen is based on the side of the market on which the position will exist, where longs are given the ask price, and shorts are given the bid price. When a position is created or increased, the chosen price is in the same direction, which means it's possible to enter and exit a position without having to cross the spread.

From the Pyth docs:

> Use a Spread: Pyth provides a confidence interval for each price update. Derivative protocols can use this confidence interval to determine the range in which the true price probably lies. By using the lower bound of the confidence interval, derivative protocols can protect themselves from price manipulation that drives the price down. By using the upper bound of the confidence interval, derivative protocols can protect themselves from price manipulation that drives the price up.

The docs for the Solidity version of GMX echo the same concern:

> Since the contracts use an oracle price which would be an average or median price of multiple reference exchanges. Without a price impact, it may be profitable to manipulate the prices on reference exchanges while executing orders on the contracts.

When entry and exit are allowed at the same price, rather than the confidence interval being used to protect against manipulation, the confidence interval makes manipulation *easier* to manipulate prices because the confidence interval is working in the favor of the attacker, rather than against them. The price impact no longer has to be large enough to cross the spread, since the spread has already been crossed.

Impact

An attacker can manipulate one or two exchanges in order to manipulate the confidence interval, and thus the spread, in order to get a more favorable entry or exit price. One specific case where this would be possible is outlined in an article from the Pyth team:

> The publishers publish distinct prices with non-overlapping confidence intervals. In this case, all votes of a single publisher will be adjacent in the sorted list and we can treat them as a single vote. Therefore, Pyth's aggregate price reduces to the median of the publisher's prices. This scenario is depicted in the 4th graph from the left above.

In other words, if there is dispersion among the prices of different exchanges, the attacker only needs to manipulate the price of one of the medians, in order to affect the spread and get a better entry/exit.

In addition to the market prices being highly manipulable, the markets themselves will break down. Because there is no spread when entering and exiting, users will be constantly submitting small orders for quickly entering then exiting if there's a small loss. This continuous flow of orders will tax the keeper system's resources. Rapid changes in trader positions will make funding rates extremely unstable, and will drive away institutional hedgers. Finally, the liquidity pools themselves will become taxed, because traders will be able to exit quickly without losing any collateral, and the payment of profits to the winners will raise the borrowing fees for all.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/utils.rs#L21

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L512

Tool used

Manual Review

Recommendation

```
-    let mut execution_price = index_price.pick_price(is_long).clone();
+    let mut execution_price = index_price.pick_price(!is_long).clone();
```

```
-                    .pick_price(self.position.is_long())
+                    .pick_price(!self.position.is_long())
```

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue H-3: Liquidations can be blocked by donating to position

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/266

Summary

The `close_position()` function processes liquidation by transferring `position.lamports - refund` from the `executor` to the `owner` before transferring the `position.lamports` to the `executor` during position closure. This logic can be exploited by manipulating `position.lamports` through donations, enabling attackers to block liquidations by requiring the `executor` to have exorbitant funds for liquidating.

Vulnerability Detail

The `close_position()` function is called when a position gets liquidated.

```
fn close_position(&self) -> Result<()> {
    let Some(position) = self.position else {
        return err!(CoreError::PositionIsRequired);
    };

    let balance = position.to_account_info().lamports();
    if balance < self.refund {
        msg!(
            "Warn: not enough balance to pay the executor, balance = {}, refund =
↪   {}",
            balance,
            self.refund,
        );
    }
    let refund = balance.saturating_sub(self.refund);

    if refund != 0 {
        system_program::transfer(
            CpiContext::new(
                self.system_program.clone(),
                system_program::Transfer {
                    from: self.executor.clone(),
                    to: self.owner.clone(),
                },
            ),
            refund,
        )?;
    }
    position.close(self.executor.clone())?;

    Ok(())
}
```

As one can see from the snippet, the function will first transfer `position.lamports - refund` from the `executor` to the `owner` . Afterward, the `position.lamports` gets transferred to the `executor` when closing the position. This requires that `executor.lamports >= position.lamports - refund`; otherwise, the transfer will always revert.

This leads to an issue, as the `position.lamports` could be manipulated by donating funds to the position. As a result, an attacker could execute the following plan:

1. Donate a lot of funds to his position
2. Position goes negative and become liquidatable
3. People try liquidating but don't have enough funds
4. Once someone liquidates the attacker, all his donated funds will still be transferred back to him

This allows an attacker to at least block liquidations for a significant time, resulting in losses to the protocol.

Impact

The issue allows users to DOS liquidations of their positions. If the attacker has a lot of sol (more than other market participants and the protocol treasury) he could DOS indefinitely. This will result in protocol losses as positions can't be liquidated sufficiently.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/order.rs#L953-L984

Tool used

Manual Review

Recommendation

The issue can be mitigated by reordering the transfer and the close so that the executor first receives all funds.

Discussion

orion-047

Will Fix

J4X-98

Fix confimed

Issue H-4: unchecked_sync_es_factor() if current gt_amount==0 and does not synchronize user es_factor, resulting in es being stolen

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/263

Summary

`unchecked_sync_es_factor()` is used to synchronize the `user.gt.es_factor = global.es_factor`. However, the current implementation returns early if `gt_amount==0` and does not synchronize this value, resulting in a large `diff_factor` that steals the `es`

Vulnerability Detail

Before the user's `user.gt.amount` is modified, we must first synchronize `user.gt.es_factor = global.es_factor` This is an important invariant, because `es` is calculated based on the difference `es = user.gt.amount * (global.es_factor - user.gt.es_factor)` The main way to calculate the es that can be obtained is through the `unchecked_sync_es_factor()` method, while synchronizing `user.gt.es_factor = store.gt.es_factor`.

```
    pub(crate) fn unchecked_sync_es_factor(&mut self, user: &mut UserHeader) ->
 ↪  Result<()> {
        use gmsol_model::utils::apply_factor;

        let gt_amount = u128::from(user.gt.amount());

        if gt_amount == 0 {
@>          return Ok(());
        }


        let current_factor = self.es_factor;
        let user_factor = user.gt.es_factor;
        require_gte!(current_factor, user_factor, CoreError::Internal);

        if current_factor == user_factor {
            return Ok(());
        }

        let diff_factor = current_factor
            .checked_sub(user_factor)
            .ok_or_else(|| error!(CoreError::ValueOverflow))?;
        let amount = apply_factor::<_, { constants::MARKET_DECIMALS }>(&gt_amount,
↪   &diff_factor)
            .ok_or_else(|| error!(CoreError::ValueOverflow))?;

        let amount: u64 = amount.try_into()?;

        let next_es_amount = user
            .gt
            .es_amount
            .checked_add(amount)
            .ok_or_else(|| error!(CoreError::TokenAmountOverflow))?;

        let next_es_supply = self
            .es_supply
            .checked_add(amount)
            .ok_or_else(|| error!(CoreError::TokenAmountOverflow))?;

        /* The following steps should be infallible. */

        user.gt.es_amount = next_es_amount;
        user.gt.es_factor = current_factor;
        self.es_supply = next_es_supply;

        Ok(())
    }
```

Since `gt_amount == 0` is returned early it does not synchronize `user.gt.es_factor`, resulting in the user having a large `diff_factor`. For example: 1.now, alice.gt.amount = 0 2.then call gt.mint_to(alice,5) after method execution , returning early could result in `alice.gt.amount = 5` then `alice.gt.es_factor = 0` After that alice has es = 5 * (store.gt.es_factor - 0)

Impact

es being stolen

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/states/gt.rs#L229

Tool used

Manual Review

Recommendation

```
    pub(crate) fn unchecked_sync_es_factor(&mut self, user: &mut UserHeader) ->
 ↪  Result<()> {
....

        if gt_amount == 0 {
+            user.gt.es_factor = self.es_factor;
            return Ok(());
```

Discussion

bin2chen66

Fix confirmed

https://github.com/sherlock-audit/2024-11-gmx/pull/284/files

when `gt\_amount == 0` , `user.gt.es\_factor` has synchronized the current `self.es\_factor`.

Issue H-5: perform_glv_deposit() missing slippage check min_glv_token_amount

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/262

Summary

The user executing `create_glv_deposit()` specifies the minimum number of glv_tokens received:`params.min_glv_token_amount`. However, when `order_keeper` executes `execute_glv_deposit()`, it doesn't check that the actual `glv_token` received must not be less than this value.

Vulnerability Detail

When `execute_glv_deposit()` is executed, we deposit the `long/short token` into the `market` to generate the `market token` and calculate the `market_token_value`. We also calculate the `glv_token` by calculating the `pool_value` and `glv_supply` of the `glv`. The `market_token_value/pool_value` is affected in many ways. So we will specify `GlvDeposit.min_glv_token_amount` and `revert` if the final amount received is less than this value to avoid attacks and losses. Currently `execute_glv_deposit()` doesn't do this check.

```
    fn perform_glv_deposit(self) -> Result<()> {
...
                let glv_amount = usd_to_market_token_amount(
                    received_value,
                    glv_value,
                    u128::from(glv_supply),
                    constants::MARKET_USD_TO_AMOUNT_DIVISOR,
                )
                .ok_or_else(||
 ↪  error!(CoreError::FailedToCalculateGlvAmountToMint))?;
                u64::try_from(glv_amount).map_err(|_|
 ↪  error!(CoreError::TokenAmountOverflow))?
            };

            op.commit();

            glv_amount
@>        //audit missing check slippage
 ↪  glv_amount>=deposit.params.min_glv_token_amount
        };

        // Invertible operations after the commitment.
        {
            // Complete the market tokens transfer.
            self.transfer_market_tokens_in();

            // Mint GLV token to the receiver.
            self.mint_glv_tokens(glv_token_amount);
        }

        Ok(())
    }
```

Impact

Lack of slippage checking can lead to common attacks such as price manipulation/sandwiches/front-run etc.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/glv.rs#L405

Tool used

Manual Review

Recommendation

```
    fn perform_glv_deposit(self) -> Result<()> {
...
            op.commit();
+           require_gte!(
+                glv_amount,
+                deposit.params.min_glv_token_amount,
+               CoreError::NotEnoughTokenAmount
+           );
            glv_amount
        };

        // Invertible operations after the commitment.
        {
            // Complete the market tokens transfer.
            self.transfer_market_tokens_in();

            // Mint GLV token to the receiver.
            self.mint_glv_tokens(glv_token_amount);
        }
        Ok(())
    }
```

Discussion

bin2chen66

Fix confirmed

https://github.com/sherlock-audit/2024-11-gmx/pull/284/files

Add method `validate\_output\_amount()` for validating `glv\_amount` This issue has been resolved

Issue H-6: Potential seed collision allows users to prevent liquidations

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/252

Summary

The report highlights a vulnerability in the `unchecked_process_position_cut()` function, which enables the `ORDER_KEEPER` to liquidate user positions. The issue arises from a potential seed collision in the initialization of the `order` object. The `nonce` parameter, which is user-controlled, can be exploited to create an order that conflicts with the `position_cut` order. This allows malicious users to front-run liquidation attempts and block the `ORDER_KEEPER` from completing the liquidation process.

Vulnerability Detail

The `unchecked_process_position_cut()` allows the `ORDER_KEEPER` to liquidate positions. However, when looking at the function context, we can spot a problem when looking at the seed of the initialized `order` object.

```
#[account(
    init,
    space = 8 + Order::INIT_SPACE,
    payer = authority,
    seeds = [Order::SEED, store.key().as_ref(), owner.key().as_ref(), &nonce],
    bump,
)]
pub order: AccountLoader<'info, Order>
```

The `Order::SEED`, `store.key().as_ref()`, `owner.key().as_ref()` parts are all fixed, and `nonce`, which is passed by the arguments, can be chosen at will. As both the normal orders a user can create and the position cut order are of the type `Order` a seed collision could occur if a position cur and a normal order use the same nonce.

As a result, a user can monitor if he `ORDER_KEEPER` tries to liquidate him, front-run the `ORDER_KEEPER`'s creation, and create a different order with the same nonce. In that case, the `ORDER_KEEPER` call to `unchecked_process_position_cut()` would fail, and they would have to retry. The user could keep doing that indefinitely, blocking the `ORDER_KEEPER` from liquidating him.

The frontrunning could happen in the two following scenarios:

1. The `position_cut`-TX is submitted using a mempool, which can be monitored
2. The owner of position is the operator of the node that receives the `position_cut`-TX and can read out the nonce there

Impact

The issue allows a user to block an admin from liquidating his position using the `position_cut()` function.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/instructions/exchange/position_cut.rs#L69-L76

Tool used

Manual Review

Recommendation

The issue can be mitigated by using the `ORDER_KEEPER`'s address instead of the `owner`'s address in the seed of the `Order` object.

```
/// The order to be created.
#[account(
    init,
    space = 8 + Order::INIT_SPACE,
    payer = authority,
    seeds = [Order::SEED, store.key().as_ref(), authority.key().as_ref(), &nonce],
    bump,
)]
pub order: AccountLoader<'info, Order>,
```

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue M-1: Attackers can prevent ADL by manipulating liquidity pool balances

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/275

Summary

Swaps of secondary output tokens at the end of decrease action processing can be used to fail ADL orders

Vulnerability Detail

At the end of decrease action processing, secondary output tokens are unconditionally swapped to output tokens. It is possible for this swap to fail due to market validation checks, such as the ones for max pool size, reserve coverage, and PnL-to-pool-value ratios, failing.

Impact

A whale can open a large long or short position using the collateral token of the other side, so that PnL is paid using secondary token, and needs to be swapped. If the whale is about to be ADL'd, they can prevent it by manipulating liquidity pool balances, such as by depositing liquidity up to the maximum pool amount for their position's collateral, so that any attempt to swap the PnL/secondary token to their output/collateral token will fail due to the max pool amount validation.

A whale need not initiate such a condition - it may occur organically in various markets. Having a large position that cannot be ADL'd will lead to an insolvent market and to capped PnL for other market participants.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L274-L279

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/swap.rs#L266-L283

Tool used

Manual Review

Recommendation

We recommend altering the swap code to support rolling back the operation, so that it can be undone/skipped if there are failures.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue M-2: The disabling of the swap feature can be bypassed via increase/decrease orders

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/274

Summary

The disabling of the swap feature can be bypassed via an increase order with minimum required position size, and a decrease order with a swap path

Vulnerability Detail

The checking of whether a particular order type feature is enabled only happens during order creation and execution. The final steps of a `MarketDecrease` order involve swapping to the PnL token, then executing the swaps provided in the swap path, which is exactly what is done for a `MarketSwap` order.

Impact

If an admin attempts to disable swaps in order to stave off some sort of attack, the attacker can create a normal increase order with the minimum position size, and the desired amount of collateral to swap. Once that order is executed, they can create, and have executed, a market decrease order with the desired swap path, since decrease orders unconditionally swap.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L272-L279

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/order.rs#L1392

Tool used

Manual Review

Recommendation

We recommend forbidding non-empty swap paths and swaps at the end of collateral processing, if either `DomainDisabledFlag` swap flag is set.

Discussion

orion-047

Won't Fix

Issue M-3: Unfair liquidations when pending position fees are large

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/273

Summary

Users with large pending position fees will not be able to top up their collateral levels to stave off liquidation

Vulnerability Detail

When performing a position increase, the full amount of position fees (order, funding, and borrowing) owed must be paid. The current code subtracts these fees from the added collateral, but errors rather than taking any remaining amounts owed from the position's existing collateral, even if there is plenty. The decrease code properly subtracts fees from collateral, but doesn't allow the addition of more collateral, and may itself be temporarily disabled.

Impact

If the user is unable to provide the full position fee's cost during the increase, in addition to having provided their position's collateral which itself was sufficient to cover the pending fees up to that point, they'll eventually be liquidated.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L442-L454

Tool used

Manual Review

Recommendation

We recommend allowing the remaining costs to be subtracted from the position's collateral, as is done in the Solidity version of GMX.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue M-4: Insolvent closes not allowed for ADL operations

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/272

Summary

Positions may have enough PnL to not be liquidated, but not enough collateral to cover costs when an ADL operation needs to be performed, leading to a position that can't be reduced or closed until price impacts decrease.

Vulnerability Detail

The code that allows insolvent closes does not allow ADL operations to be insolvently closed, even if the position is fully closed.

There may be very large positions that have a lot of pending profits, but an even larger amount of pending borrowing and funding fees. In such cases the position cannot be liquidated because the net profit plus collateral is enough to pass the solvency checks, and the position cannot be ADL'd because the negative price impact cost for ADL is larger than the price impact cost considered by the liquidation checks, and the larger impact would lead to not all costs being covered.

Impact

The ADL feature will not fulfill its purpose of maintaining solvency when index token price growth outstrips collateral token price growth. LPs will not be able to deposit/withdraw, and all user PnL will be unfairly capped.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L113-L115

Tool used

Manual Review

Recommendation

We recommend modifying `DecreasePosition::try_new()` and its callers by replacing `is_insolvent_close_allowed` with `is_adl_order`, and updating the condition to be:

```
                 initial_collateral_withdrawal_amount:
 ↪  collateral_withdrawal_amount.clone(),
-                is_insolvent_close_allowed: is_insolvent_close_allowed
-                    && (size_delta_usd == *position.size_in_usd())
-                    && is_liquidation_order,
+                is_insolvent_close_allowed:
+                    (size_delta_usd == *position.size_in_usd())
+                    && (is_liquidation_order || is_adl_order),
                 is_liquidation_order,
```

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue M-5: Only Oracle Controller authentication checks for the wrong role

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/271

Summary

The Oracle Controller is authorized to call instructions such as clear_all_prices() and set_prices_from_feed(). However, the `only_oracle_controller()` authentication checks that the signer has the `GT_CONTROLLER` role.

Vulnerability Detail

The implementation for only_oracle_controller() checks for the `GT_CONTROLLER` role instead of the `ORACLE_CONTROLLER` role.

```
/// Check that the `authority` has the
↪  [`ORACLE_CONTROLLER`](`RoleKey::ORACLE_CONTROLLER`) role.
fn only_oracle_controller(ctx: &Context<Self>) -> Result<()> {
     Self::only(ctx, RoleKey::GT_CONTROLLER) // @audit-issue GT_CONTROLLER instead
↪  of ORACLE_CONTROLLER
}
```

Impact

Only GT Controllers can modify the oracles instead of the Oracle Controllers.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/utils/internal/authentication.rs#L47-L50

Tool used

Manual Review

Recommendation

Consider replacing `GT_CONTROLLER` with `ORACLE_CONTROLLER` in `only_oracle_controller()`.

Discussion

orion-047

Will Fix

gjaldon

Fix confirmed. The issue was addressed here.

Issue M-6: Delegated amount is ignored when preparing a claimable account

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/270

Summary

The Order Keeper prepares a claimable account to receive tokens for claiming when executing orders and give the owner access to these tokens. An error in the code prevents delegating the claimable amount to the owner.

Vulnerability Detail

In unchecked_use_claimable_account(), the delegated amount is ignored and the owner is always delegated 0.

```
pub(crate) fn unchecked_use_claimable_account(
    ctx: Context<UseClaimableAccount>,
    _timestamp: i64,
    amount: u64,       // <========== The delegated amount
) -> Result<()> {
    if ctx.accounts.account.delegate.is_none() ||
 ↪  ctx.accounts.account.delegated_amount != amount {
        anchor_spl::token::approve(
            CpiContext::new_with_signer(
                ctx.accounts.token_program.to_account_info(),
                anchor_spl::token::Approve {
                    to: ctx.accounts.account.to_account_info(),
                    delegate: ctx.accounts.owner.to_account_info(),
                    authority: ctx.accounts.store.to_account_info(),
                },
                &[&ctx.accounts.store.load()?.signer_seeds()],
            ),
            0,  // @audit-issue The delegated amount isn't used and 0 is always
 ↪  passed
        )?;
    }
    Ok(())
}
```

Impact

The funds in the claimable account will be stuck until the contract is upgraded to fix the issue.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/instructions/token.rs#L113-L126

Tool used

Manual Review

Recommendation

Consider replacing `0` with `amount`.

Discussion

orion-047

Will Fix

gjaldon

Fix confirmed. The issue was addressed here.

Issue M-7: Capped PnL calculations always minimize the value of pool tokens

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/268

Summary

PnL is affected by the bid/ask spread, which may lead to incorrect pool token valuations during deposit/withdrawal, and incorrect PnL during position reduction.

Vulnerability Detail

The code that values a GM pool's tokens properly minimizes/maximizes the pool's collateral tokens' value, but when it comes to subtracting the net owed capped PnL, the PnL value of the index tokens is always capped by, at minimum, the minimized collateral value. Similarly, the calculation of a position's PnL during position reduction, also caps the amount using the minimized collateral value.

Impact

In cases where the bid price drops down to almost zero (e.g. during periods of extreme market uncertainty), the price used to cap the PnL will also be almost zero, which means the PnL value owed by a pool will be close to zero. GM token holders that withdraw during such periods will be able to withdraw many more tokens than they should be able to, at the expense of all the other LPs that remain until normal market conditions.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/base.rs#L203-L209

Tool used

Manual Review

Recommendation

We recommend modifying `cap_pnl()` to take in a `maximize` parameter, and passing it to the call to `pool_value_without_pnl_for_one_side()`.

Discussion

orion-047

Will Fix

IIIIIII000

In the Solidity code, `poolUsd` is passed, as the penultimate argument, to the `getCappedPnl()`, and in the original Rust code, `pool\_value` is calculated in `cap\_pnl()` rather than being passed in. Based on the three `poolUsd` values in the Solidity code, I believe the fix should be changed as follows: `position.rs::pnl\_value()::cap\_pnl(false -> !self.is\_long())` and `liquidity.rs::pool\_value()::cap\_pnl(true -> maximize)` (for both instances in that function)

orion-047

Regarding the invocation of `cap\_pnl` within `LiquidityMarketExt::pool\_value`, I believe you are right.

As for `Position::pnl\_value`, I believe the correct implementation here is still to minimize the pool value.

However, the implementation of `cap\_pnl` is indeed not good enough. We plan to move it to the new `pub(crate) MarketUtils` trait and have it accept the `pool\_value` as a parameter, similar to the Solidity version, instead of calculating it internally.

IIIIIII000

Fix confirmed

Issue M-8: GLV tokens are misvalued when there are pending position impact pool distributions

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/267

Summary

The position impact pool pending distributions are improperly accounted for when valuing market tokens, leading to GLV tokens being misvalued duing deposit/withdrawal.

Vulnerability Detail

A GM pool's value, and thus its GM token's value's, calculation involves offsetting the pool's USD value by the USD value of the pending position impact pool, but it mistakenly offsets it by the pending amount instead. While GM deposits and withdrawals are unaffected, since they commit the distribution prior to their operations via calls to `distribute_position_impact()`, GLV deposits and withdrawals use the pending value without committing first.

Impact

Any user with GLV tokens for a GLV pool that contains a market with a large pending distribution, can withdraw their share of GM tokens at an inflated valuation, at the expense of the other token holders. Conversely, any user attempting to deposit into such a GLV pool will receive fewer tokens than they should.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/liquidity.rs#L131-L139

Tool used

Manual Review

Recommendation

```
        // Deduct impact pool value.
        let impact_pool_value = {
```

```
+            use num_traits::CheckedMul;
             let duration =
 ↪  self.passed_in_seconds_for_position_impact_distribution()?;
-            self.pending_position_impact_pool_distribution_amount(duration)?
-                .1
+            let impact_pool_amount =
 ↪  self.pending_position_impact_pool_distribution_amount(duration)?
+                .1;
+            let price = prices.index_token_price.pick_price(!maximize);
+            impact_pool_amount.checked_mul(price)
+                .ok_or(crate::Error::Computation("calculating impact pool value"))?
         };
         pool_value = pool_value
             .checked_sub(&impact_pool_value)
```

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue M-9: Incorrect exclusion amount for pure markets during validation

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/265

Summary

`validate_market_balance_for_the_given_token` function of `ValidateMarketBalances` does validation of one token for a market after excluding a specific amount, but it does not exclude correct amount for pure markets which will cause issues during the validation.

Vulnerability Detail

```
fn validate_market_balance_for_the_given_token(
    &self,
    token: &Pubkey,
    excluded: u64,
) -> Result<()> {
    let side = self.market_meta().to_token_side(token)?;
    let balance = self
@>      .balance_excluding(token, &excluded)
        .map_err(ModelError::from)?
        .into();
    self.validate_token_balance_for_one_side(&balance, side)
        .map_err(ModelError::from)?;
    Ok(())
}
```

Here's the code snippet of the function. It gets final balance as `totalTokenAmount - excluded`. This calculation is correct for normal markets, but not for pure markets.

In pure markets, long and short tokens are same so their balances are mixed together and assumed it contains half long tokens and half short tokens. So in other parts of the codebase, excluding amount is split between long and short tokens when it's pure markets, which means it should apply same for this case.

Impact

Actions that validate markets using this function might fail because of excluding doubled amounts for pure markets. For now, the function is used in claiming fees from a market, thus accruing fees from a market might fail because of over excluding.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/states/market/utils.rs#L83

Tool used

Manual Review

Recommendation

Excluding amounts should be halved for pure markets.

```
fn validate_market_balance_for_the_given_token(
    &self,
    token: &Pubkey,
    excluded: u64,
) -> Result<()> {
    let side = self.market_meta().to_token_side(token)?;
    let balance = self
-       .balance_excluding(token, &excluded)
+       .balance_excluding(token, self.is_pure() ? &(excluded / 2) : &excluded)
        .map_err(ModelError::from)?
        .into();
    self.validate_token_balance_for_one_side(&balance, side)
        .map_err(ModelError::from)?;
    Ok(())
}
```

Discussion

orion-047

Will Fix

KupiaSecAdmin

Fix confirmed.

Issue M-10: transfer_tokens_out() using wrong short token market

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/264

Summary

`transfer_tokens_out()` when transferring a `short token` uses `find_and_unpack_last_market(is_primary=true)` incorrectly, resulting in the wrong `market` being selected.

Vulnerability Detail

In `execute_glv_withdrawal()` execution was successful We use `transfer_tokens_out()` to transfer `tokens` to `escrow`.

`unchecked_execute_glv_withdrawal()->transfer_tokens_out()`

```
    fn transfer_tokens_out(
        &self,
        remaining_accounts: &'info [AccountInfo<'info>],
        final_long_token_amount: u64,
        final_short_token_amount: u64,
    ) -> Result<()> {
...
        if final_short_token_amount != 0 {
            let market = self
                .glv_withdrawal
                .load()?
                .swap
@>              .find_and_unpack_last_market(store, true, remaining_accounts)?
                .unwrap_or(self.market.clone());
            let vault = &self.final_short_token_vault;
            let escrow = &self.final_short_token_escrow;
            let token = &self.final_short_token;
            builder
                .market(&market)
                .to(escrow.to_account_info())
                .vault(vault.to_account_info())
                .amount(final_short_token_amount)
                .decimals(token.decimals)
                .token_mint(token.to_account_info())
                .build()
                .execute()?;
        }
        Ok(())
    }
```

From the code above we know that `short_token` uses `find_and_unpack_last_market(is_primary=true)`. This will fetch the `market` of the `long token`.

Note: execute_withdrawal.rs#L297 similarity

Impact

The final transfer-out fails, even if it is resubmitted, it will be invalid, and the user have to re-select the path with a relatively low exchange rate.cause a certain loss of token

if two markets have the same long_token, finally, it will be transferred out successfully, but it will lead to wrong accounting.

```
`market_right`.state.pools.primary.long_token_amount -> decrease
`market_right`.state.other.long_token_balance -> not change


`market_error`.state.pools.primary.long_token_amount -> not change
`market_error`.state.other.long_token_balance -> decrease
```

Incorrect accounting in the market will lead to the inability to get back the funds.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/instructions/glv/withdrawal.rs#L729

Tool used

Manual Review

Recommendation

```
        if final_short_token_amount != 0 {
            let market = self
                .glv_withdrawal
                .load()?
                .swap
-               .find_and_unpack_last_market(store, true, remaining_accounts)?
+               .find_and_unpack_last_market(store, false, remaining_accounts)?
```

Discussion

bin2chen66

Fix confirmed

https://github.com/sherlock-audit/2024-11-gmx/pull/284/files

1. `glv/withdrawal.rs`
2. `exchange/execute\_withdrawal.rs`

`transfer\_tokens\_out()` All changed, if `short\_token`, use `is\_primary=false`.

Issue M-11: When OracleTimestampsAreLargerThanRequired occurs, GlvDeposit should not be executed.

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/261

Summary

When `OracleTimestampsAreLargerThanRequired` occurs, `glv_deposit` should become canceled `ActionState.Cancelled`. However, the loss of `return` causes it to be possible to continue with `perform_glv_deposit()`.

Vulnerability Detail

when `create_glv_deposit()` we write `GlvDeposit.header.updated_at = clock.unix_timestamp` We expect `execute_glv_deposit()` with `oracle.max_oracle_ts` greater than `GlvDeposit.header.updated_at + store.amount.request_expiration` can no longer perform this `GlvDeposit`

```
    pub(crate) fn unchecked_execute(self) -> Result<bool> {
        let throw_on_execution_error = self.throw_on_execution_error;
        match self.validate_oracle() {
            Ok(()) => {}
            Err(CoreError::OracleTimestampsAreLargerThanRequired) if
 ↪  !throw_on_execution_error => {
                msg!(
                    "GLV Deposit expired at {}",
                    self.oracle_updated_before()
                        .ok()
                        .flatten()
                        .expect("must have an expiration time"),
                );
@>              //audit missing return Ok(false);
            }
            Err(err) => {
                return Err(error!(err));
            }
        }
        match self.perform_glv_deposit() {
            Ok(()) => Ok(true),
            Err(err) if !throw_on_execution_error => {
                msg!("Execute GLV deposit error: {}", err);
                Ok(false)
            }
            Err(err) => Err(err),
        }
```

From the code above we know that `OracleTimestampsAreLargerThanRequired` occurs and does not return, causing the execution of `perform_glv_deposit()` to continue

Note:`ExecuteGlvWithdrawalOperation/ExecuteGlvShiftOperation.unchecked_execute()` Similar

Impact

In times of sharp price changes, overdue deposits or withdrawals will affect the amount of token users receive

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/glv.rs#L267

Tool used

Manual Review

Recommendation

```
    pub(crate) fn unchecked_execute(self) -> Result<bool> {
        let throw_on_execution_error = self.throw_on_execution_error;
        match self.validate_oracle() {
            Ok(()) => {}
            Err(CoreError::OracleTimestampsAreLargerThanRequired) if
↪   !throw_on_execution_error => {
                msg!(
                    "GLV Deposit expired at {}",
                    self.oracle_updated_before()
                        .ok()
                        .flatten()
                        .expect("must have an expiration time"),
                );
+               return Ok(false);
            }
```

Discussion

bin2chen66

Fix confirmed

https://github.com/sherlock-audit/2024-11-gmx/pull/284/files

1. `ExecuteGlvDepositOperation.unchecked\_execute()`
2. `ExecuteGlvWithdrawalOperation.unchecked\_execute()`
3. `ExecuteGlvShiftOperation.unchecked\_execute()`

All have been added, if `OracleTimestampsAreLargerThanRequired` occurs, return `Ok(false)`

Issue L-1: `DecreasePositionReport::pnl` is missing from the debug output

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/283

Summary

`DecreasePositionReport::pnl` is missing from the debug output

Vulnerability Detail

`DecreasePositionReport::pnl` is not present in the output generated by `DecreasePositionReport::fmt()`

Impact

Not including all `struct` fields makes debug output less useful in debugging potential issues.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/report.rs#L25-L29

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/report.rs#L57-L58

Tool used

Manual Review

Recommendation

We recommend adding `pnl` to the debug output.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue L-2: Unused account `CloseOrder::gt_token_program`

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/282

Summary

The account `CloseOrder::gt_token_program` is never used

Vulnerability Detail

The `close_order()` instruction requires a `gt_token_program` `TokenInterface` account to be provided, but the account is never used.

Impact

The unused account has computational unit costs associated with its anchor validations, and each additional account in an instruction limits the number of instructions that can fit into a single transaction.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/instructions/exchange/order.rs#L588-L589

Tool used

Manual Review

Recommendation

We recommend removing the account from the instruction.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue L-3: GMX Model divergences

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/281

Summary

The Rust/Solana implementation of the GMX model is different in some small ways, from the Solidity implementation.

Vulnerability Detail

There are small differences in both behavior and output available to consumers.

Impact

While it's perfectly valid for the two implementations to differ, including if they were to differ greatly, any difference makes writing integrations that interact with both, difficult, and adds extra overhead during subsequent security reviews. The differences below do not, in and of themselves, rise to the level of a security issue.

Code Snippet

These are the differences we've identified:

- The Solidity version looks up the token balance before doing the comparison, rather than using the pool's storage value. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/base.rs#L350-L355
- The Solidity version implements a kinked borrowing factor model. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/borrowing.rs#L83
- The Solidity version skips the application of the delta if the delta is zero, and therefore doesn't emit. With the reviewed code, the callers are logging unconditionally https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/distribute_position_impact.rs#L52-L53
- The Solidity version implements a 'virtual inventory' to track impacts across similar tokens. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/swap.rs#L201
- The Solidity version attempts to pay the excess beyond the cap of the side that was capped, using the other side https://github.com/sherlock-audit/2024-11-gmx/blob

/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/swap.rs#L211-L228

- The Solidity version errors if the in amount becomes zero. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/swap.rs#L235-L237
- The Solidity version returns a signed value, since PnL might be larger than the collateral, which lets callers see how close ADL is to completion, by seeing how negative the price is. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/liquidity.rs#L127-L129
- The Solidity version uses one as the price if there is no supply. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/liquidity.rs#L152-L154
- In the Solidity version, for liquidations, the factor is applied to the position size, rather than the delta size.
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/position.rs#L564-L573
- In cases where no more collateral remains and should_validate_min_collateral_usd is true, the Solidity code returns `"min collateral"`, whereas the current code returns `LiquidatableReason::NotPositive`.
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/position.rs#L472-L482
- The Solidity code does a sanity check that the market and collateral are also valid, subsequent to any execution.
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/position.rs#L398
- The reviewed code uses the `max_factor_per_second` as a factor and applies it to the `diff_value_to_open_interest_factor`, but in the Solidity version, the max caps the result of the application, rather than being used as a replacement factor to apply. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/update_funding_state.rs#L226-L235
- The Solidity version uses -1 or 1 in the case where the decrease is equal to the existing factor, so that it's possible for `is_skew_the_same_direction_as_funding` to be true during the next update, rather than forcing the rate change to be an increase. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/update_funding_state.rs#L282-L287
- The Solidity version does the operations in a different order, and thus overflows under different conditions. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/utils.rs#L38-L45

- The Solidity version has a step argument for saying at what step things stopped. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/collateral_processor.rs#L576
- The Solidity version tracks the original size and has it in the result, rather than the capped delta https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L106
- The Solidity version only changes the original size for two of the OrderTypes https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L106
- The Solidity version has fees for liquidation https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L411-L412
- The Solidity version performs the swap before fees are assessed, so that positive impact can potentially pay for fees that may otherwise be unpaid https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L445
- The solidity version allows swap kinds other than `NoSwap`, and allows orders to specify the kind they want during order creation. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L546
- The solidity version emits during a swap, whereas the current code does not add the swap report to the `DecreasePositionReport`. https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L566

Tool used

Manual Review

Recommendation

We recommend either changing the Rust implementation to match the Solidity one, or to clearly document the differences and reasons that they exist.

Discussion

orion-047

Will Fix

IIIIIII000

- The first item is missing a comment
- For the third item, it still logs unconditionally, so there should be a comment describing the difference
- Refactoring for the tenth item introduced a bug: `should\_validate\_min\_collateral\_usd.then(|| params.min\_collateral\_factor())`, should be `should\_validate\_min\_collateral\_usd.then(|| params.min\_collateral\_value())`,

IIIIIII000

Fix confirmed

Issue L-4: Liquidations and ADLs may be DOSed if claimable accounts or token escrows are frozen

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/280

Summary

Liquidations and ADLs may be DOSed if claimable or token escrow accounts are frozen by a token's freeze authority.

Vulnerability Detail

On Solana, tokens that have a freeze authority can use that authority to prevent tokens from being transferred to/from any user's address. Though extremely unlikely, it's possible that a freeze authority freezes either a claimable account (holds funding fee payments) or a token escrow account (holds output tokens when a position in profit is cut).

Impact

Any operation that interacts with these types of accounts will be prevented from completing them. In the case of ADL or liquidations, the positions won't be able to be cut until the keeper creates new claimable/escrow accounts, and retries the operation.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/order.rs#L472-L517
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/order.rs#L442-L457

Tool used

Manual Review

Recommendation

We recommend documenting that keepers must be cognizant of the possibility of such a failure, and have automated procedures to work around it quickly.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue L-5: Overflows during collateral sufficiency checks trigger insufficient collateral errors

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/279

Summary

Overflows during collateral sufficiency checks trigger insufficient collateral errors rather than errors indicating that an overflow occurred.

Vulnerability Detail

During the calculation of sufficient collateral, a configuration factor is applied. If this application overflows, the code ends up triggering an insufficient collateral error rather than an error indicating that an overflow occurred.

Impact

A user that encounters the error will try to submit their order again with a larger amount of collateral, only to get another error saying that more is required.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/position.rs#L257-L263

Tool used

Manual Review

Recommendation

```
-        let Some(min_collateral_value_for_leverage) =
+        let min_collateral_value_for_leverage =
             crate::utils::apply_factor(&delta.next_size_in_usd,
 ↪   &min_collateral_factor)
-        else {
-            return Ok(WillCollateralBeSufficient::Insufficient(
-                remaining_collateral_value,
-              ));
-          };
+              .ok_or(crate::Error::Computation(
+                  "overflow calculating min collateral value for leverage",
+              ))?;
```

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue L-6: Inconsistent overflow and output behavior when using various factor exponents

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/278

Summary

The code that calculates exponents behaves inconsistently depending on whether the exponent is a whole unit or not.

Vulnerability Detail

1. The output has more decimals (and thus better precision) when the exponent is a whole unit:

```
#[cfg(feature = "u128")]
#[test]
fn pow_u128_truncate() {
    let x = U128D20::from_inner(u64::MAX as u128);

    let exp = U128D20::from_inner(U128D20::ONE.0 - 1);
    let ans = x.checked_pow(&exp).unwrap();
    assert_eq!(ans, U128D20::from_inner(18446744100000000000));

    let exp = U128D20::from_inner(U128D20::ONE.0);
    let ans = x.checked_pow(&exp).unwrap();
    assert_eq!(ans, U128D20::from_inner(18446744073709551615)); // more decimals

    let exp = U128D20::from_inner(U128D20::ONE.0 + 1);
    let ans = x.checked_pow(&exp).unwrap();
    assert_eq!(ans, U128D20::from_inner(18446744000000000000));
}
```

2. Different exponents have vastly different overflow characteristics, depending on whether they're whole units or not:

```
#[cfg(feature = "u128")]
#[test]
fn pow_u128_algo() {
    let x = U128D20::from_inner(10u128 * U128D20::ONE.0);
    let exp = U128D20::from_inner(18u128 * U128D20::ONE.0);
    let ans = x.checked_pow(&exp).unwrap();
    assert_eq!(ans, U128D20::from_inner(100000000000000000000000000000000000000));
    //                      u128::MAX = 340282366920938463463374607431768211455
    let exp = U128D20::from_inner(6u128 * U128D20::ONE.0 - 1);
    let ans = x.checked_pow(&exp);
    assert_eq!(ans, None); // should not have errored with smaller exponent
    let exp = U128D20::from_inner(4u128 * U128D20::ONE.0 - 1);
    let ans = x.checked_pow(&exp).unwrap();
    assert_eq!(ans, U128D20::from_inner(999999520588100000000000)); // 18-4=12
 ↪  orders of magnitude
}
```

Impact

The calculation of borrow and funding fees, and of price impact all rely on the `checked_pow()` function to calculate their values. If a non-unit exponent factor is used, the calculations of these fees will overflow much sooner, leading to positions being stuck until the factor is changed to a unit exponent.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/fixed.rs#L56

Tool used

Manual Review

Recommendation

The sponsor has confirmed with their risk parameter team that using only unit exponents is acceptable, and that is our recommendation.

Discussion

orion-047

Won't Fix

Issue L-7: Pure markets lose one token sub-unit to rounding, when an odd amount of tokens exists in the pool

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/277

Summary

Pure markets lose one token sub-unit to rounding, when an odd amount of tokens exists in the pool, where the total pool amount is divided in two in order to spread the tokens between both the long and short sides.

Vulnerability Detail

Pure markets divide their balance of tokens by two and use the rounded down amount for both the long and short side amounts, not taking into account potential loss of precision.

Impact

For pure markets, if the long token balance is odd, the last user to withdraw will be unable to claim the last token sub-unit, since both sides round down the balance. The borrow fees may be slightly incorrect if the value of the smallest token sub-unit is large (e.g. BTC).

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/states/market/revertible/market.rs#L99-L101

Tool used

Manual Review

Recommendation

We recommend choosing one side to round up the division for, and continue rounding down for the other.

Discussion

orion-047

Will Fix

IIIIIII000

Fix confirmed

Issue L-8: Incorrect rent assumption on `Action`

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/258

Summary

The `validate_balance()` function ensures the Action account retains enough rent after deducting execution_lamports. However, it fails to account for the 8 bytes used by the account discriminator, leading to an underestimated rent calculation.

Vulnerability Detail

The `validate_balance()` function verifies that even after using up the `execution_lamports`, the `Action` account has enough rent to prevent it from being reaped.

```
/// Validate balance.
fn validate_balance(account: &AccountLoader<Self>, execution_lamports: u64) ->
↪  Result<()>
where
    Self: ZeroCopy + Owner + InitSpace,
{
    require_gte!(
        execution_lamports,
        Self::MIN_EXECUTION_LAMPORTS,
        CoreError::NotEnoughExecutionFee
    );
    let balance = account.get_lamports().saturating_sub(execution_lamports);
    let rent = Rent::get()?;
    require!(
        rent.is_exempt(balance, Self::INIT_SPACE),
        CoreError::NotEnoughExecutionFee
    );
    Ok(())
}
```

However, this function misses that an account uses 8 bytes for its discriminator in addition to its `INIT_SPACE`. As a result, the calculated minimum rent is off by 8 bytes.

Impact

The issue could result in actions being closed prematurely due to a lack of rent.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/states/common/action.rs#L276-L279

Tool used

Manual Review

Recommendation

We recommend adapting the code to also account for the account discriminator.

```
require!(
    rent.is_exempt(balance, 8 + Self::INIT_SPACE),
    CoreError::NotEnoughExecutionFee
);
```

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue L-9: `only_controller()` function is never used

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/257

Summary

The `only_controller()` function, which provides access control for the `CONTROLLER` role, is defined but never used, contributing to unnecessary code complexity.

Vulnerability Detail

The `only_controller()` function provides access control for the `CONTROLLER` role.

```
/// Check that the `authority` has the [`CONTROLLER`](`RoleKey::CONTROLLER`) role.
fn only_controller(ctx: &Context<Self>) -> Result<()> {
    Self::only(ctx, RoleKey::CONTROLLER)
}
```

However, this function is never used.

Impact

The issue results in unnecessary code complexity.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/utils/cpi.rs#L63-L66

Tool used

Manual Review

Recommendation

We recommend removing the unused function.

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue L-10: Roles can be set multiple times

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/256

Summary

The `enable_role()` function allows roles to be enabled but lacks a check to throw an error when the role is already set, resulting in unnecessary execution costs.

Vulnerability Detail

The `enable_role()` function allows roles to be enabled.

```
pub fn enable_role(&mut self, role: &str) -> Result<()> {
    match self.roles.get_mut(role) {
        Some(metadata) => {
            require_eq!(metadata.name()?, role, CoreError::InvalidArgument);
            metadata.enable();
        }
        None => {
            let index = self
                .roles
                .len()
                .try_into()
                .map_err(|_| error!(CoreError::ExceedMaxLengthLimit))?;
            self.roles
                .insert_with_options(role, RoleMetadata::new(role, index)?, true)?;
        }
    }
    Ok(())
}
```

However, the function lacks a check that throws an error if the role is already set.

Impact

The issue results in unnecessary execution costs.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/states/roles.rs#L154-L171

Tool used

Manual Review

Recommendation

We recommend throwing an error if a role is already set.

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue L-11: Incorrect documentation of `check_admin()`

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/255

Summary

The report highlights a discrepancy between the documentation and the behavior of the `check_admin()` function. While the documentation states that the function will revert if the `authority` is not the store's admin, the code only returns `false` in such cases without throwing an error.

Vulnerability Detail

The documentation of the `check_admin()` function states that it will revert if the `authority` is not the store's admin.

```
/// Check that the signer is the admin of the given store, throw error if
/// the check fails.
///
/// # Accounts
/// *[See the documentation for the accounts.](CheckRole).*
///
/// # Errors
/// - The [`authority`](CheckRole::authority) must be a signer and be
///   the `ADMIN` of the store.
/// - The [`store`](CheckRole::store) must have been initialized
///   and owned by the store program.
pub fn check_admin(ctx: Context<CheckRole>) -> Result<bool> {
    instructions::check_admin(ctx)
}
```

However, when looking at the wrapped `instructions::check_admin` function, we can see that it will only return false in that case but not throw an error.

```
/// Verify that the `user` is an admin of the given `store`.
pub(crate) fn check_admin(ctx: Context<CheckRole>) -> Result<bool> {
    Ok(ctx
        .accounts
        .store
        .load()?
        .is_authority(ctx.accounts.authority.key))
}
```

```
/// Check if the given pubkey is the authority of the store.
pub fn is_authority(&self, authority: &Pubkey) -> bool {
    self.authority == *authority
}
```

Impact

The issue results in incorrect documentation.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/lib.rs#L325-L326

Tool used

Manual Review

Recommendation

We recommend updating the documentation to reflect that the function will not error if the user is not an admin.

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue L-12: Incorrect documentation of `update_adl_state()`

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/254

Summary

The report identifies a documentation inconsistency in the `update_adl_state()` function. While the documentation states that the function can only be called by the store's `CONTROLLER`, the code enforces that it can only be called by the `ORDER_KEEPER`.

Vulnerability Detail

The documentation for the `update_adl_state()` function states that it can only be called by the store's `CONTROLLER`.

```
/// - The [`authority`](UpdateAdlState::authority) must be a signer and a
/// CONTROLLER of the store.
```

However, this is incorrect as it can only be called by the `ORDER_KEEPER`.

```
#[access_control(internal::Authenticate::only_order_keeper(&ctx))]
```

Impact

The issue results in incorrectly documented access restrictions.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/lib.rs#L1643-L1648

Tool used

Manual Review

Recommendation

We recommend updating the comment to state the correct access restriction.

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Issue L-13: Typos in codebase

Source: https://github.com/sherlock-audit/2024-11-gmx/issues/253

Summary

Vulnerability Detail

The codebase includes multiple typos. This leads to a lower perceived quality of code.

Impact

Typos lead to a lower quality of code and, thus, potential damage to the team's credibility.

Code Snippet

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/collateral_processor.rs#L183
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/collateral_processor.rs#L217
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/collateral_processor.rs#L443
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/collateral_processor.rs#L460
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L36
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L66
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/mod.rs#L378
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/report.rs#L13
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/decrease_position/utils.rs#L50

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L53
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L54
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L70
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L74
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L108
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L111
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/increase_position.rs#L150
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/swap.rs#L132
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/action/update_funding_state.rs#L223
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/fixed.rs#L155
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/base.rs#L36
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/base.rs#L385
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/base.rs#L444
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/borrowing.rs#L89
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/borrowing.rs#L94
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/liquidity.rs#L164
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/market/perp.rs#L272
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/num.rs#L132
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/num.rs#L209
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/pool/mod.rs#L80
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-model/src/position.rs#L889
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-utils/src/price/decimal.rs#L129
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-utils/src/price/decimal.rs#L130

https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-utils/src/fixed_map.rs#L63
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/crates/gmsol-utils/src/fixed_map.rs#L86
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/lib.rs#L915
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/lib.rs#L1407
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/instructions/exchange/order.rs#L484
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/market.rs#L302
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/shift.rs#L216
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/ops/order.rs#L785
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/utils/internal/action.rs#L124
https://github.com/sherlock-audit/2024-11-gmx/blob/ee60a905137a4eeedcb87fccfaba86e8277bbb42/gmx-solana/programs/gmsol-store/src/constants/mod.rs#L41

Tool used

Manual Review

Recommendation

We recommend fixing the typos.

Discussion

orion-047

Will Fix

J4X-98

Fix confirmed

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.