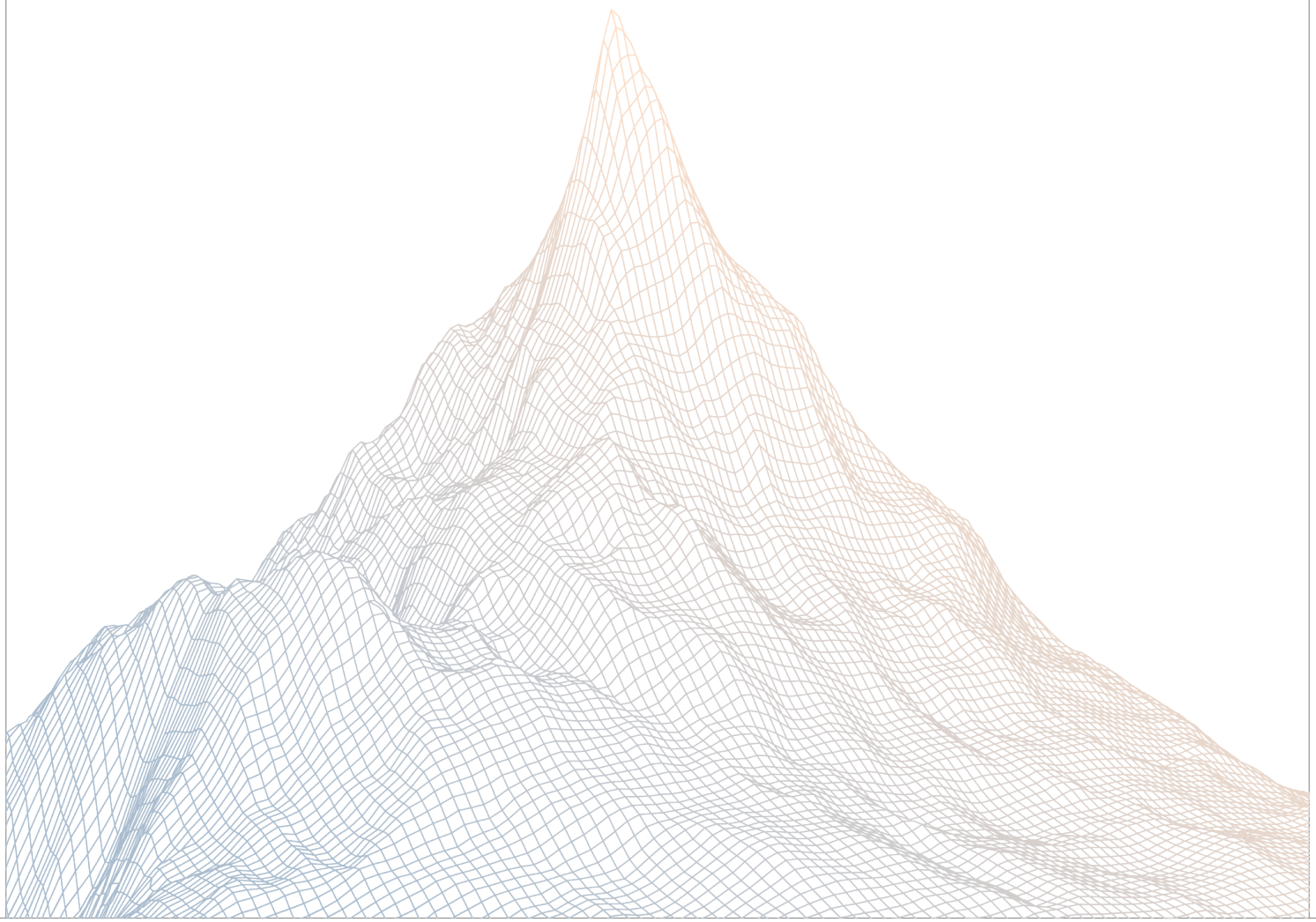
Zenith

GMX Solana Protocol

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: January 16th to February 26th, 2025

AUDITED BY: 0xTheC0der
Dadekuma
FrankCastle
Peakbolt
Owen Thurm

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

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

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

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

2.2 Scope

The engagement involved a review of the following targets:

Target	gmx-solana
Repository	https://github.com/gmsol-labs/gmx-solana
Commit Hash	`d91fedbed571a7b92e99695b124b0a62bb839128`

Files	`gmsol-store/src/lib.rs`
	`gmsol-store/src/constants/*`
	`gmsol-store/src/events/*`
	`gmsol-store/src/instructions/*`
	`gmsol-store/src/ops/*`
	`gmsol-store/src/states/*`
	`gmsol-store/src/utils/*`
	`gmsol-store/src/utils/internal/*`
	`gmsol-timelock/src/lib.rs`
	`gmsol-timelock/src/roles.rs`
	`gmsol-timelock/src/instructions/config.rs`
	`gmsol-timelock/src/instructions/executor.rs`
	`gmsol-timelock/src/instructions/instruction_buffer.rs`
	`gmsol-timelock/src/instructions/mod.rs`
	`gmsol-timelock/src/states/*`
	`gmsol-treasury/src/*`

2.3 Audit Timeline

January 16, 2025	Audit start
February 26, 2025	Audit end
March 7, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	9
Medium Risk	23
Low Risk	24
Informational	8
Total Issues	**64**

3

Findings Summary

ID	Description	Status
H-1	Incorrect authority for token transfer in withdraw_from_treasury_vault()	Resolved
H-2	It's impossible to set the price feed data if any token uses a Chainlink oracle	Resolved
H-3	Time-locked instructions will fail to execute due to insufficient signing	Resolved
H-4	Wrong funding factor on markets with adaptive funding	Resolved
H-5	Missing feed ID verification in case of Switchboard provider	Resolved
H-6	Permanent Denial of Service in initialize_glv instruction hence all GLV functions Due to Pre-Created Associated Token Accounts.	Resolved
H-7	min_output parameter is not passed in for increase order	Resolved
H-8	Swaps Perturb Borrowing State	Resolved
H-9	Users can shift high-risk GM tokens by bypassing the maximum PNL check	Resolved
M-1	gt_set_exchange_time_window() could cause mis-pricing of GT exchange	Resolved
M-2	prepare_gt_exchange_vault() could fail due to seed collision	Resolved
M-3	unchecked_execute_instruction() fails to check approver is not revoked	Resolved
M-4	unchecked_revoke_role() will always fail due to wrong signer	Resolved
M-5	Incorrect access control for set_expected_price_provider()	Resolved
M-6	Missing verification of the fee receiver address during Store initialization can cause DoS of claiming treasury fees	Resolved
M-7	Pyth prices with high variance devalue the protocol	Acknowledged

ID	Description	Status
M-8	Order keeper can DoS others GLV Shift by calling close_glv_shift repeatedly	Resolved
M-9	Changing the fee receiver causes permanent DoS of claiming treasury fees	Resolved
M-10	close_empty_claimable_account and remove_glv_market should refund rent to the account creator	Resolved
M-11	State Inconsistency Due to Solana Rollback	Resolved
M-12	Funding calculation might not revert when total_open_interest is zero	Resolved
M-13	Switchboard check_and_get_price() should use min slot of the min/max prices	Resolved
M-14	Anyone can prevent market removal from the GLV	Resolved
M-15	Malicious TIMELOCK_KEEPER can set is_signer for another TIMELOCK_KEEPER's account to steal funds from it	Resolved
M-16	Imbalanced incentive for ADL creates unfair position closures and increases insolvency risk	Resolved
M-17	Malicious order keeper can perform risk-free trade using different price in the same tx	Resolved
M-18	Far future price timestamps can block feed updates	Resolved
M-19	GLV shifts could be blocked by a high spread	Resolved
M-20	GLV max market value can be exceeded	Resolved
M-21	ADL can be DoS by spamming update_adl_state()	Resolved
M-22	initialize_referral_code() can be griefed using transfer_referral_code()	Resolved
M-23	forPositiveImpact Neglects To Incentivize Markets Without Price Impact Configuration	Acknowledged

ID	Description	Status
L-1	final_short_token doesn't have any constraints on a close withdrawal request	Resolved
L-2	TIMELOCK_KEEPER can steal rent from others by calling execute_instruction	Resolved
L-3	Wrong INIT_SPACE of SwapExecuted	Resolved
L-4	Unexpected transfer of treasury fees	Resolved
L-5	Insufficient authority segregation on Store initialization	Resolved
L-6	Order keeper can DoS GLV Shift by executing dust amount shift	Resolved
L-7	CreateGlvShift can be DoS by another order keeper	Resolved
L-8	Missing constraining of final_short_token mint in Execute-Withdrawal context	Resolved
L-9	Incorrect use of the require_eq macro for Pubkey values	Resolved
L-10	Unsorted Referral Reward Factors Validation	Resolved
L-11	Limit orders can't specify an execution time	Resolved
L-12	ADL cannot be triggered when min_pnl_factor is reached	Resolved
L-13	Limit orders don't have a deadline	Acknowledged
L-14	Missed validation for Account Length in load_instruction	Resolved
L-15	Misleading core error UnknownOrDisabledToken	Resolved
L-16	Invalid future oracle price timestamps pass verification	Resolved

ID	Description	Status
L-17	Missing implementation of GT reserve	Resolved
L-18	Overrestrictive treasury_vault_config check on treasury withdrawals can leave old vaults inaccessible	Resolved
L-19	Missing validation for min_output in Order::update()	Resolved
L-20	Account seeds based on index of type u8 might limit long-term protocol operation	Resolved
L-21	There is no feature flag to disable deposits and withdrawals	Resolved
L-22	Missing rent exempt check on withdrawal creation	Resolved
L-23	Order keepers are wrongfully charged for GLV shifts	Resolved
L-24	Users overpay a position's rent on pure markets	Resolved
I-1	The project relies on vulnerable crate dependencies	Resolved
I-2	Inconsistency of market vault token account seeds	Resolved
I-3	Unused discount when minting GT	Resolved
I-4	Unused accounts in swap.rs	Resolved
I-5	Use of outdated Pyth Solana Receiver Rust SDK	Resolved
I-6	Incorrect error used for event_loader failure	Resolved
I-7	unchecked_insert_factor is callable by both CONFIG_KEEPER and MARKET_KEEPER	Resolved
I-8	Use of outdated Switchboard On-Demand libraries	Resolved

4

Findings

4.1 High Risk

A total of 9 high risk findings were identified.

[H-1] Incorrect authority for token transfer in `withdraw_from_treasury_vault()`

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

Target

- /programs/gmsol-treasury/src/instructions/treasury.rs#L544-L556

Description:

The `authority` for `WithdrawFromTreasuryVault` CPI context is incorrectly set as `self.config.to_account_info()`. It should instead be `self.treasury_vault_config.to_account_info()`.

This issue will cause `withdraw_from_treasury_vault()` to always fail.

```
impl<'info> WithdrawFromTreasuryVault<'info> {
    fn transfer_checked_ctx(&self) → CpiContext<'_, '_, '_, 'info,
    TransferChecked<'info>> {
        CpiContext::new(
            self.token_program.to_account_info(),
            TransferChecked {
                from: self.treasury_vault.to_account_info(),
                mint: self.token.to_account_info(),
                to: self.target.to_account_info(),
                authority: self.config.to_account_info(),
            },
        )
    }
}
```

Recommendations:

```
impl<'info> WithdrawFromTreasuryVault<'info> {
    fn transfer_checked_ctx(&self) -> CpiContext<'_, '_, '_, 'info,
    TransferChecked<'info>> {
        CpiContext::new(
            self.token_program.to_account_info(),
            TransferChecked {
                from: self.treasury_vault.to_account_info(),
                mint: self.token.to_account_info(),
                to: self.target.to_account_info(),
                authority: self.config.to_account_info(),
                authority: self.treasury_vault_config.to_account_info(),
            },
        )
    }
}
```

GMX Solana: Fixed in @ea50664122...

Zenith: Verified.

[H-2] It's impossible to set the price feed data if any token uses a Chainlink oracle

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

Target

- gmsol-store/src/states/oracle/chainlink.rs
- gmsol-store/src/states/oracle/price_map.rs

Description:

Chainlink price result is saved as both the min and max price:

```
    let price = Decimal::try_from_price(
        *answer as u128,
        decimals,
        token_config.token_decimals(),
        token_config.precision(),
    )
    .map_err(|_| error!(CoreError::InvalidPriceFeedPrice))?;
    Ok((
        round.slot,
        timestamp,
        Price {
→           min: price,
→           max: price,
        },
    ))
```

When the price is set in the `price_map` the transaction will fail if any token uses a Chainlink oracle:

```
    // Assume the remaining accounts are arranged in the following way:
    // [token_config, feed; tokens.len()] [..remaining]
    for (idx, token) in tokens.iter().enumerate() {
        let feed = &remaining_accounts[idx];
        let token_config = map.get(token).ok_or_else(||
    error!(CoreError::NotFound))?;
```

```
        require!(token_config.is_enabled(), CoreError::TokenConfigDisabled);

        let oracle_price = OraclePrice::parse_from_feed_account(
            validator.clock(),
            token_config,
            chainlink,
            feed,
        )?;

        validator.validate_one(
            token_config,
            &oracle_price.provider,
            oracle_price.oracle_ts,
            oracle_price.oracle_slot,
            &oracle_price.price,
        )?;
        self.primary
→           .set(token, oracle_price.price, token_config.is_synthetic())?;
    }
```

This is caused by a check that ensures that the max price must always be greater than the min price, which is always false in this case:

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

Recommendations:

Consider relaxing the price requirement while setting the `price_map`:

```
    require_gt!(price.max.value, price.min.value, CoreError::InvalidArgument);
```

```
require_gte!(price.max.value, price.min.value, CoreError::InvalidArgument)
    ;
```

GMX Solana: Resolved with @ea50664122a...

Zenith: Verified.

[H-3] Time-locked instructions will fail to execute due to insufficient signing

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

Target

- programs/gmsol-timelock/src/instructions/instruction_buffer.rs#L389-L395

Description:

Every time-locked instruction is buffered with its executor and all its accounts necessary for execution where some of them can be signers. Additionally, the executor is always set as a signer too, see here.

However, on execution of the instruction, it is only signed by the executor wallet's signer seeds and not by the executor account which is set as a signer. Additionally, it is not signed by any of the accounts that were specified as signers when the instruction was created.

Consequently, time-locked instructions will fail to execute blocking crucial protocol actions.

Recommendations:

It is recommended to revise the instruction signing mechanism:

1. Set the executor's wallet instead of the executor as signer in the instruction.
2. Ensure the invocation is signed by all accounts that were set as signers when the instruction was created.

GMX Solana: Fixed in @36112aa6868....

Zenith: Verified. Resolved by marking the executor wallet as signer instead of the executor.

[H-4] Wrong funding factor on markets with adaptive funding

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

Target

- gmsol-store/src/ops/market.rs
- gmsol-model/src/action/update_funding_state.rs

Description:

When a market updates its funding state, the calculation of the funding factor may be wrong on a market with adaptive funding (i.e. when `increase_factor_per_second > 0`).

The issue is that when the increase factor is positive, the function that calculates the funding factor returns zero even if it should not.

This would result in a zero funding factor as the `funding_value` will be multiplied by zero (which will be wrong when the open interest and/or short interest are not zero).

Recommendations:

Consider returning a zero funding factor only when the adaptive funding is not enabled:

```
    let funding_increase_factor_per_second = self.market.funding_fee_params()?.
      increase_factor_per_second();
if diff_value.is_zero(){
if diff_value.is_zero() && funding_increase_factor_per_second.is_zero() {
        return Ok((Zero::zero(), true, Zero::zero()));
    }
```

GMX Solana: Resolved with @ea50664122...

Zenith: Verified.

[H-5] Missing feed ID verification in case of Switchboard provider

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

Target

- programs/gmsol-store/src/states/oracle/mod.rs#L384-L386

Description:

In case of Switchboard being the price provider, the provided feed account is never verified against the stored feed ID of the token configuration, neither in the `parse_from_feed_account` function nor within the subsequent call to `Switchboard::check_and_get_price`.

Consequently, when `set_prices_from_remaining_accounts` is invoked, any valid Switchboard price feed, given in `remaining_accounts`, could be used for any token having Switchboard as the configured provider which can lead to severe mispricing.

For reference, the Chainlink and Pyth cases perform such a feed verification.

Recommendations:

It is recommended to also implement a feed ID verification in case of Switchboard being the price provider

GMX Solana: Fixed in @3f24e5a254...

Zenith: Verified.

[H-6] Permanent Denial of Service in initialize_glv instruction hence all GLV functions Due to Pre-Created Associated Token Accounts.

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

Target

- management.rs (initialize_vaults)

Description:

In the `initialize_vaults` function, the store program uses the `create` instruction to generate the associated token accounts (ATAs) for vaults with the `market_token`:

```
create(CpiContext::new(
    self.associated_token_program.to_account_info(),
    Create {
        payer: self.authority.to_account_info(),
        associated_token: vault.clone(),
        authority: self.glv.to_account_info(),
        mint: market_token.clone(),
        system_program: self.system_program.to_account_info(),
        token_program: self.market_token_program.to_account_info(),
    },
))?;
```

However, since ATAs can be created by anyone before the `glv` account is initialized, an attacker can preemptively create at least one of the market vaults before the `InitializeGlv` instruction executes.

This results in a **denial of service (DoS) attack** because the `InitializeGlv` instruction requires at least one market to initialize. If the attacker creates the vault beforehand, the function call to `create` will fail since it returns an error if the account already exists:

```
/// Creates an associated token account for the given wallet address and
/// token mint. Returns an error if the account exists.
```

As a result, the entire GLV initialization process becomes **permanently blocked**, rendering

the GLV functionality unusable at almost no cost to the attacker.

Recommendations:

To mitigate this issue, use the `create_idempotent` instruction instead of `create`. Unlike `create`, `create_idempotent` will create the ATA **only if it doesn't already exist**, avoiding the DoS issue while still ensuring proper vault initialization.

Example implementation:

```
pub fn create_idempotent<'info>(
    ctx: CpiContext<'_, '_, '_, 'info, CreateIdempotent<'info>>,
) → Result<()> {
    let ix = spl_associated_token_account::instruction::
        create_associated_token_account_idempotent(
        ctx.accounts.payer.key,
        ctx.accounts.authority.key,
        ctx.accounts.mint.key,
        ctx.accounts.token_program.key,
    );
    anchor_lang::solana_program::program::invoke_signed(
        &ix,
        &[
            ctx.accounts.payer,
            ctx.accounts.associated_token,
            ctx.accounts.authority,
            ctx.accounts.mint,
            ctx.accounts.system_program,
            ctx.accounts.token_program,
        ],
        ctx.signer_seeds,
    )
    .map_err(Into::into)
}
```

According to the documentation:

```
/// Creates an associated token account for the given wallet address and
/// token mint, if it doesn't already exist.  Returns an error if the
/// account exists, but with a different owner.
```

By switching to `create_idempotent`, the program ensures that vaults are created only when necessary while avoiding errors if an attacker preemptively creates them.

GMX Solana: Resolved with @25a0d36c5b...

Zenith: Verified.

[H-7] `min_output` parameter is not passed in for increase order

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

Target

- order.rs#L280-L297

Description:

For increase position orders, it is possible to swap from an initial collateral token to the position market collateral token. During execution, slippage check is in place as the `min_output` is validated in `execute_increase_position()`.

However when increase orders are created in `CreateIncreaseOrderOperation`, the `min_output` value is not passed into `params.init_increase()` to set it in the `order` account.

This will cause `min_output` to be always zero when the order is executed, causing the slippage check to always pass. That will lead to users to have their initial collateral swapped at an unexpected rate despite indicating a `min_output` amount, causing them to incur losses when the final output amount is below their expectation.

```
impl<'a, 'info> CreateIncreaseOrderOperation<'a, 'info> {
    pub(crate) fn execute(self) -> Result<()> {
        self.common.validate()?;
        self.validate_params_excluding_swap()?;

        let collateral_token = if self.common.params.is_collateral_long {
            self.common.market.load()?.meta().long_token_mint
        } else {
            self.common.market.load()?.meta().short_token_mint
        };

        self.common.init_with(|create, tokens, params| {
            tokens
                .initial_collateral
                .init(self.initial_collateral_token);
            tokens.long_token.init(self.long_token);
            tokens.short_token.init(self.short_token);
            params.init_increase(
```

```
                //@audit missing min_output parameter
                create.is_long,
                create.kind,
                self.position.key(),
                collateral_token,
                create.initial_collateral_delta_amount,
                create.size_delta_value,
                create.trigger_price,
                create.acceptable_price,
            )?;
            Ok((self.initial_collateral_token.mint, collateral_token))
        })?;
```

Recommendations:

Set the `min_output` value in the order account during order creation.

GMX Solana: Fixed in @828c0ac48fa....

Zenith: Verified.

[H-8] Swaps Perturb Borrowing State

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

Description:

During the execution of swaps the borrowing state is not updated for markets which are not the starting market in the swap path. The swap will change the backing balance of both the long and short sides and thus change the borrowing fee rate that should be paid by traders. However the borrowing fees are not charged for the previous time period from [lastBorrowingUpdate, currentTimeOfSwap] and thus the borrowing fees which have accrued in the past have been changed.

A malicious actor could potentially trigger such a swap to intentionally decrease the amount of borrowing fees that they presently owe for their position. Or a malicious actor could trigger such a swap to intentionally cause borrowing fees to rise immediately and unexpectedly, causing positions to be liquidated.

Recommendations:

Consider updating the borrowing state of all markets in a swap path prior to the swap occurring.

GMX Solana: Fixed in @2775da2636...

Zenith: Verified.

[H-9] Users can shift high-risk GM tokens by bypassing the maximum PNL check

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

Target

- gmsol-store/src/ops/glv.rs

Description:

When a GLV deposit is executed, `is_market_deposit_required` is `true` only when the user provides either a short or long amount of tokens. In `unchecked_deposit`, there is a check to ensure that the max PNL was not exceeded during the deposit.

The issue is that the PNL check can be skipped if the user deposits only market tokens; a user could make a deposit when the open interest exceeds the max PNL that was set (and thus when new deposits should be disabled) to withdraw from a high risk market through a low risk market (which is normally prohibited through other paths, such as direct shifts or withdraw-deposit).

Recommendations:

Consider checking that the max PNL is not exceeded when the user deposits market tokens:

```
let mt = self.market_token_mint.clone();
let mut market = RevertibleLiquidityMarketOperation::new(
    &self.store,
    self.oracle,
    &self.market,
    self.market_token_mint,
    self.token_program.clone(),
    Some(&deposit.swap),
    self.remaining_accounts,
    self.event_emitter,
)?;

let mut op = market.op()?;
```

```
if market_token_amount ≠ 0 {
    let prices = self.oracle.market_prices(op.market())?;
    self.market.load()?.as_liquidity_market(&mt).validate_max_pnl(
        &prices,
        PnlFactorKind::MaxAfterWithdrawal,
        PnlFactorKind::MaxAfterWithdrawal,
    ).map_err(|_| error!(CoreError::PnlFactorExceeded))?;
}
```

```
if deposit.is_market_deposit_required() {
    let executed = op.unchecked_deposit(
        &deposit.header().receiver(),
        &self.market_token_vault,
        &deposit.params.deposit,
        (
            deposit.tokens.initial_long_token.token(),
            deposit.tokens.initial_short_token.token(),
        ),
        None,
    )?;

    market_token_amount = market_token_amount
        .checked_add(executed.output)
        .ok_or_else(|| error!(CoreError::TokenAmountOverflow))?;

    op = executed.with_output(());
}
```

GMX Solana: Fixed in @2a6676ld65...

Zenith: Verified.

4.2 Medium Risk

A total of 23 medium risk findings were identified.

[M-1] `gt_set_exchange_time_window()` could cause mis-pricing of GT exchange

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

Target

- /programs/gmsol-store/src/instructions/gt.rs#L124-L134

Description:

`gt_set_exchange_time_window()` can be used to change the `time_window` that determines the `time_window_index` for the `gt_exchange_vault`.

However, that will prevent the depositing of token into `gt_bank` using `deposit_treasury_vault()` to increase `gt_bank` balance for buyback, due to the constraints that requires `store.load()?.gt().exchange_time_window() as i64 == gt_exchange_vault.load()?.time_window()`.

Due to the issue, `gt_bank` could have insufficient buy back value, causing the users who had requested GT exchange to receive a low price for their GT.

Recommendations:

`gt_set_exchange_time_window()` should check there are no unconfirmed `gt_exchange_vault` before updating the `exchange_time_window`.

That means `gt_set_exchange_time_window()` should be called after all `gt_exchange_vault` are confirmed and before the next `prepare_gt_exchange_vault()` is called.

GMX Solana: Fixed in @ea5066412...

Zenith: Verified. Resolved by making `gt_set_exchange_time_window()` a test-only function and removed it from deployment build.

[M-2] `prepare_gt_exchange_vault()` could fail due to seed collision

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

Target

- /programs/gmsol-store/src/instructions/gt.rs#L148

Description:

The gt_exchange_vault account is created using `prepare_gt_exchange_vault()`, based on the seed `[GtExchangeVault::SEED, store.key().as_ref(), &time_window_index.to_be_bytes()]`.

However, as `time_window` can be changed using `set_exchange_time_window()`, it is possible for two different gt_exchange_vaults with different time_window to have the same `time_window_index`.

That will cause the gt_exchange_vault addresses to clash and prevent the creation new gt_exchange_vault that has the same `time_window_index` but different `time_window`.

```
/// The accounts definition for [`pre-
    pare_gt_exchange_vault`](crate::gmsol_store::prepare_gt_exchange_vault)
    instruction.
#[derive(Accounts)]
#[instruction(time_window_index: i64)]
pub struct PrepareGtExchangeVault<'info> {
    #[account(mut)]
    pub payer: Signer<'info>,
    #[account(constraint = store.load()?.gt().is_initialized() @
    CoreError::PreconditionsAreNotMet)]
    pub store: AccountLoader<'info, Store>,
    #[account(
        init_if_needed,
        space = 8 + GtExchangeVault::INIT_SPACE,
        payer = payer,
        seeds = [GtExchangeVault::SEED, store.key().as_ref(),
    &time_window_index.to_be_bytes()],
        bump,
```

```
    )]
    pub vault: AccountLoader<'info, GtExchangeVault>,
    pub system_program: Program<'info, System>,
}
```

Recommendations:

Add `time_window` to the seed for `GtExchangeVault`.

GMX Solana: Fixed in @ea50664122...

Zenith: Verified.

[M-3] `unchecked_execute_instruction()` fails to check approver is not revoked

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

Target

- instruction_buffer.rs#L385

Description:

`unchecked_execute_instruction()` performs a validation using `is_executable()` to ensure that the instruction has been approved and past the timelock.

However, it fails to check that the approver is not revoked of the role. This allows revoked approvers (which are timelocked) to approve malicious instruction while their revocation are still in timelock.

```
pub(crate) fn unchecked_execute_instruction(ctx:
    Context<ExecuteInstruction>) -> Result<()> {
    let remaining_accounts = ctx.remaining_accounts;

    let instruction = ctx.accounts.instruction.load_instruction()?;

    let delay = ctx.accounts.timelock_config.load()?.delay();
    require!(
        instruction.header().is_executable(delay)?,
        CoreError::PreconditionsAreNotMet
    );

    let signer = ExecutorWalletSigner::new(ctx.accounts.executor.key(),
    ctx.bumps.wallet);

    invoke_signed(
        &instruction.to_instruction(),
        remaining_accounts,
        &[&signer.as_seeds()],
    )?;
    Ok(())
}
```

Recommendations:

Check that the approver of the instruction still has the required role when calling `unchecked_execute_instruction()`.

GMX Solana: Fixed in @ea5066412..

Zenith: Verified.

[M-4] `unchecked_revoke_role()` will always fail due to wrong signer

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

Target

- revoke_role.rs#L99

Description:

`unchecked_revoke_role()` is invoking the `revoke_role()` CPI using the executor wallet signer.

However, the CPI context in `revoke_role_ctx()` incorrect set authority to `self.executor.to_account_info()` instead of `self.wallet.to_account_info()`.

```
impl<'info> RevokeRole<'info> {
    fn revoke_role_ctx(&self) -> CpiContext<'_, '_, '_, 'info,
    StoreRevokeRole<'info>> {
        CpiContext::new(
            self.store_program.to_account_info(),
            StoreRevokeRole {
                //@audit this should be self.wallet as it is invoking with
    executor wallet signer
                authority: self.executor.to_account_info(),
                store: self.store.to_account_info(),
            },
        )
    }
}

/// Revoke a role. This instruction will bypass the timelock check.
/// # CHECK
/// Only [`TIMELOCKED_ADMIN`](roles::TIMELOCKED_ADMIN) can use.
pub(crate) fn unchecked_revoke_role(ctx: Context<RevokeRole>, role: String)
    -> Result<()> {
    require!(
        !NOT_BYPASSABLE_ROLES.contains(&role.as_str()),
        CoreError::InvalidArgument
```

```
    );
    let signer = ExecutorWalletSigner::new(ctx.accounts.executor.key(),
    ctx.bumps.wallet);
    let cpi_ctx = ctx.accounts.revoke_role_ctx();
    revoke_role(
        cpi_ctx.with_signer(&[&signer.as_seeds()]),
        ctx.accounts.user.key(),
        role,
    )?;
    Ok(())
}
```

Recommendations:

```
impl<'info> RevokeRole<'info> {
    fn revoke_role_ctx(&self) -> CpiContext<'_, '_, '_, 'info,
    StoreRevokeRole<'info>> {
        CpiContext::new(
            self.store_program.to_account_info(),
            StoreRevokeRole {
                authority: self.executor.to_account_info(),
                authority: self.wallet.to_account_info(),
                store: self.store.to_account_info(),
            },
        )
    }
}
```

GMX Solana: Fixed in @ea50664122...

Zenith: Verified.

[M-5] Incorrect access control for `set_expected_price_provider()`

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

Target

- lib.rs#L97

Description:

`set_expected_price_provider()` is incorrectly restricted to `TIMELOCKED_ADMIN` instead of `TIMELOCKED_MARKET_KEEPER`.

This prevents the actual users in `TIMELOCKED_MARKET_KEEPER` from calling `set_expected_price_provider()`.

```
    /// Set expected price provider.
>>> #[access_control(CpiAuthenticate::only(&ctx, roles::TIMELOCKED_ADMIN))]
    pub fn set_expected_price_provider(
        ctx: Context<SetExpectedPriceProvider>,
        new_expected_price_provider: u8,
    ) -> Result<()> {
        instructions::unchecked_set_expected_price_provider(
            ctx,
            new_expected_price_provider
                .try_into()
                .map_err(|_| error!(CoreError::InvalidProviderKindIndex))?,
        )
    }
```

The comment in `unchecked_set_expected_price_provider()` states that it is meant for `TIMELOCKED_MARKET_KEEPER`.

```
/// Revoke a role. This instruction will bypass the timelock check.
/// # CHECK
>>>/// Only
    [`TIMELOCKED_MARKET_KEEPER`](crate::roles::TIMELOCKED_MARKET_KEEPER) can
    use.
```

```
pub(crate) fn unchecked_set_expected_price_provider(
    ctx: Context<SetExpectedPriceProvider>,
    new_expected_price_provider: PriceProviderKind,
) → Result<()> {
```

Recommendations:

Change the allowed role to `TIMELOCKED_MARKET_KEEPER` as follows,

```
    /// Set expected price provider.
    #[access_control(CpiAuthenticate::only(&ctx, roles::TIMELOCKED_ADMIN))]
    #[access_control(CpiAuthenticate::only(&ctx, roles::TIMELOCKED_MARKET_
        KEEPER))]
    pub fn set_expected_price_provider(
        ctx: Context<SetExpectedPriceProvider>,
        new_expected_price_provider: u8,
    ) -> Result<()> {
        instructions::unchecked_set_expected_price_provider(
            ctx,
            new_expected_price_provider
                .try_into()
                .map_err(|_| error!(CoreError::InvalidProviderKindIndex))?,
        )
    }
```

GMX Solana: Fixed in @ea50664122...

Zenith: Verified.

[M-6] Missing verification of the fee receiver address during `Store` initialization can cause DoS of claiming treasury fees

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

Target

- programs/gmsol-store/src/states/store.rs#L98
- programs/gmsol-store/src/states/store.rs#L390-L39

Description:

When initializing the `Store`, the `authority` can be any address which is not subject to further constraints and is subsequently utilized for three distinct purposes:

- Store authority
- Treasury authority (fee receiver)
- Address authority (holding key)

However, the treasury's fee receiver address needs to be derived from a specific seed to facilitate invocation of `transfer_receiver` and `claim_fees`:

```
#[account(
    seeds = [constants::RECEIVER_SEED, config.key().as_ref()],
    bump,
)]
pub receiver: SystemAccount<'info>,
```

If the unverified `authority` address during `Store` initialization is not derived as such, the above instructions are subject to DoS.

Recommendations:

Is recommended to revise the the initialization of the treasury's fee receiver address during `Store` initialization.

GMX Solana: Fixed in @e639cc1bd9...

Zenith: Verified. Resolved by facilitating initialization of the treasury fee receiver via a

separate parameter as well as by implementing a two-step approach for transferring the receiver.

[M-7] Pyth prices with high variance devalue the protocol

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

Target

- gmsol-store/src/states/oracle/pyth.rs

Description:

While setting the price feeds data, the Pyth Oracle doesn't perform any checks to ensure that the price is valid by considering its confidence level:

```
pub fn pyth_price_with_confidence_to_price(
    price: i64,
    confidence: u64,
    exponent: i32,
    token_config: &TokenConfig,
) → Result<Price> {
    let mid_price: u64 = price
        .try_into()
        .map_err(|_| error!(CoreError::InvalidPriceFeedPrice))?;
    let min_price = mid_price
        .checked_sub(confidence)
        .ok_or_else(|| error!(CoreError::InvalidPriceFeedPrice))?;
    let max_price = mid_price
        .checked_add(confidence)
        .ok_or_else(|| error!(CoreError::InvalidPriceFeedPrice))?;
    Ok(Price {
        min: pyth_price_value_to_decimal(min_price, exponent,
    token_config)?,
        max: pyth_price_value_to_decimal(max_price, exponent,
    token_config)?,
    })
}
```

The issue is that if the confidence level results in a very high price variance, it is still considered valid, and the transaction will not revert.

This impacts the protocol value, devaluing it, as it's calculated as the

sum of the minimum prices.

Recommendations:

Consider checking the price/confidence ratio, and return an error if the variance is too high.

GMX Solana: Acknowledged. Yes, we're aware of the issue, so we won't be using Pyth as an oracle until we have a proper solution. We'll add a note to clarify.

Zenith: A comment was added, warning that there currently isn't any validation on price volatility.

[M-8] Order keeper can DoS others GLV Shift by calling `close_glv_shift` repeatedly

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

Target

- /programs/gmsol-store/src/instructions/glv/shift.rs#L234

Description:

An GLV shift can be closed by any order keepers using `close_glv_shift` so that the corresponding PDA can be closed and rent refunded when the GLV shift is cancelled or completed.

However, as it returns true for `skip_completion_check_for_keeper`, it will not perform the check that ensures that the GLV shift can only be closed when it is cancelled or completed.

That means a malicious order keeper can intentionally close GLV shifts to prevent any rebalancing of the GLV. This will cause the GLV to be unbalanced over time.

```
fn skip_completion_check_for_keeper(&self) -> bool {
    true
}
...
/// Preprocess.
fn preprocess(&self) -> Result<ShouldContinueWhenATAsAreMissing> {
    if *self.authority().key == self.action().load()?.header().owner {
        Ok(true)
    } else {
        self.only_role(self.expected_keeper_role())?;
        {
            let action = self.action().load()?;
            if self.skip_completion_check_for_keeper()
                ||
    action.header().action_state()?.is_completed_or_cancelled()
            {
                Ok(false)
            } else {
                err!(CoreError::PermissionDenied)
```

```
            }
        }
    }
}
```

Recommendations:

Set the creator as the owner during `create_glv_shift` and during `close_giv_shift` check that `action.header().action_state()?.is_completed_or_cancelled() == true` in the case of non-owner order keepers.

GMX Solana: Fixed in @ea50664122...

Zenith: Verified. Resolved as by updating `skip_completion_check_for_keeper()` to only allow creator (funder) of GLV shift to close the GLV shift.

[M-9] Changing the fee receiver causes permanent DoS of claiming treasury fees

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

Target

- programs/gmsol-treasury/src/instructions/store.rs#L35-L39
- programs/gmsol-treasury/src/instructions/store.rs#L105-L109

Description:

The `transfer_receiver` instruction allows the `TREASURY_OWNER` to change the fee receiver to any address. However, the `receiver` account address is fixed by a seed. Therefore, the `transfer_receiver` instruction will not allow changing the receiver again, since only the new receiver is allowed to change the receiver (the receiver is set as `authority` here).

The same applies to the `claim_fees` instruction where the `receiver` account address is also fixed by the same seed. Once the receiver is changed by `transfer_receiver`, any invocation of `claim_fees` will fail since its account context can only accept the original receiver while the subsequent invocation solely allows the new receiver (the receiver is set as `authority` here).

Please note that direct invocation of the `claim_fees_from_market` or `set_receiver` instructions as a workaround is not possible, since the expected signer is the `receiver` account which is a PDA and not an on-curve address, therefore only CPI from within the program is possible.

Recommendations:

It is suggested to remove the seed-based `receiver` account address restriction in the above instances since the receiver is subsequently checked against the stored receiver.

Please note that there are further instances of this seed-based `receiver` account address restriction which do not facilitate a subsequent check against the stored receiver:

- *programs/gmsol-treasury/src/instructions/swap.rs#L70-L75*
- *programs/gmsol-treasury/src/instructions/swap.rs#L226-L23*
- *programs/gmsol-treasury/src/instructions/treasury.rs#L308-L312*

These instances are insensitive to a receiver change and can only operate with the original receiver account.

GMX Solana:

In @e639cc1bd9..., we implemented a two-step approach for transferring the receiver, which requires the following steps:

1. `transfer_receiver`: The current receiver signs to set the next receiver (this can be canceled by calling the instruction again and resetting the next receiver to the current receiver).

2. `accept_receiver`: The next receiver signs to complete the update of the receiver address.

In the `initialize_config` instruction of the Treasury Program, if the current receiver of the store is not the designated receiver, it will first call `accept_receiver` to complete the transfer of the receiver.

Zenith: Verified. Mitigated by implementing a two-step approach for transferring the receiver. (intended for migration to another treasury program)

[M-10] `close_empty_claimable_account` and `remove_glv_market` should refund rent to the account creator

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

Target

- /programs/gmsol-store/src/instructions/token.rs#L176-L197
- /programs/gmsol-store/src/instructions/glv/management.rs#L414-L418

Description:

Both `close_empty_claimable_account` and `remove_glv_market` will perform account closure.

However, they refund the rent to the executor instead of the account creator, which is not tracked. That means any `ORDER_KEEPER` or `MARKET_KEEPER` that call those functions will be able to retrieve the rent, which prevents refunds to the actual account creator.

```
pub(crate) fn unchecked_close_empty_claimable_account(
    ctx: Context<CloseEmptyClaimableAccount>,
    _timestamp: i64,
) → Result<()> {
    if must_be_uninitialized(&ctx.accounts.account) {
        return Ok(());
    }
    let account = ctx.accounts.account.to_account_info();
    let amount = anchor_spl::token::accessor::amount(&account)?;
    if amount == 0 {
        anchor_spl::token::close_account(CpiContext::new_with_signer(
            ctx.accounts.token_program.to_account_info(),
            anchor_spl::token::CloseAccount {
                account: ctx.accounts.account.to_account_info(),
                destination: ctx.accounts.authority.to_account_info(),
                authority: ctx.accounts.store.to_account_info(),
            },
            &[&ctx.accounts.store.load()?.signer_seeds()],
        ))?;
    }
    Ok(())
```

```
}
```

```
pub(crate) fn unchecked_remove_glv_market(ctx: Context<RemoveGlvMarket>) →
    Result<()> {
   ....
    {
        let glv = ctx.accounts.glv.load()?;
        let signer = glv.signer_seeds();
        close_account(
            CpiContext::new(
                ctx.accounts.token_program.to_account_info(),
                CloseAccount {
                    account: ctx.accounts.vault.to_account_info(),
                    destination: ctx.accounts.authority.to_account_info(),
                    authority: ctx.accounts.glv.to_account_info(),
                },
            )
            .with_signer(&[&signer]),
        )?;
    }
```

Recommendations:

Consider tracking the account creator to allow correct refund in `close_empty_claimable_account` and `remove_glv_market`.

GMX Solana: Fixed in @ea50664122...

Zenith: Verified. Resolved by setting the rent receiver in unchecked_remove_glv_market to the store wallet.

[M-11] State Inconsistency Due to Solana Rollback

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

Target

- treasury.rs#L10-L20
- config.rs#L61-L70

Description:

Multiple configuration-modifying functions in the protocol are vulnerable to state inconsistencies in the event of a Solana rollback.

Key Risks:

1. **Setting Config Parameters**:
   - Global configuration parameters could become outdated during a rollback.
   - The protocol may operate with old, invalid settings, leading to potential system malfunctions or vulnerabilities.

Recommendations:

1. **Detect Outdated Configurations**
   - Utilize the `LastRestartSlot` sysvar to check the validity of configurations.
   - Automatically pause the protocol if configurations are outdated.
   - Require admin intervention before resuming operations.
2. **Add *`last_updated_slot`* Field**
   - Introduce a `last_updated_slot` field in the `TreasuryVaultConfig` struct to track configuration update timestamps.
   - Suggested structure modification:

```
pub struct TreasuryVaultConfig {
    version: u8,
    pub(crate) bump: u8,
    index: u8,
```

```
    #[cfg_attr(feature = "debug", debug(skip))]
    padding: [u8; 13],
    pub(crate) config: Pubkey,
    #[cfg_attr(feature = "debug", debug(skip))]
    reserved: [u8; 256],
    tokens: TokenMap,
    pub(crate) last_updated_slot: u64, // Add this field
}
```

3. **Implement Outdated Configuration Check**
   - Create a function to determine whether the configuration is outdated:

```
fn is_config_outdated(vault_config: &TreasuryVaultConfig) → Result<bool>
    {
    let last_restart_slot = LastRestartSlot::get()?;
    Ok(vault_config.last_updated_slot
    ⩽ last_restart_slot.last_restart_slot)
}
```

This ensures the protocol operates with valid configurations, mitigating risks caused by Solana rollbacks.

GMX Solana: Resolved with @5cb3ef004f8...

Zenith: Verified.

[M-12] Funding calculation might not revert when `total_open_interest` is zero

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

Target

- gmsol-store/src/ops/market.rs
- gmsol-model/src/action/update_funding_state.rs

Description:

Note: this will be an issue only when `Wrong funding factor on markets with adaptive funding` is fixed, it's recommended to read it first to have some context.

When the `next_funding_factor_per_second` is computed, the `total_open_interest` should never be zero.

A previous check ensures that this can't happen, and there is a debug assertion with a comment that explains why:

```
// `total_open_interest` must be non-zero here, since if
    `total_open_interest` were zero, the `diff_value` must be zero too.
debug_assert!(
    !total_open_interest.is_zero(),
    "this should not be possible"
);
```

However, on a market with adaptive funding enabled, `diff_value` may be zero, and the function will continue executing. In this case, it's necessary to revert when `total_open_interest` is also zero, but this won't happen on release mode because the code currently uses a `debug_assert`.

Recommendations:

Consider using `assert` instead of `debug_assert` so that the assertion will also work in release mode:

```
debug_assert!(
assert!(
      !total_open_interest.is_zero(),
      "this should not be possible"
  );
```

GMX Solana: Resolved with @f6c5cfb83e...

Zenith: Verified.

[M-13] Switchboard `check_and_get_price()` should use min slot of the min/max prices

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

Target

- /programs/gmsol-store/src/states/oracle/switchboard.rs#L40-L41

Description:

The `check_and_get_price()` will return the `feed.result.slot` value, which is used to determine the slot for the current price. It also provides the minimum and maximum prices. The value of `feed.result.slot` is then used in `validate_min_oracle_slot()` to check that the `min_oracle_slot` of all the price data are not earlier than the action's slot.

However, it is possible that the slot of the minimum and maximum prices could be earlier than the `feed.result.slot`.

That is because the minimum/maximum prices are obtained from the oracle submissions over a span of multiple slots. Whereas `feed.result.slot` refers to the slot for the current price, which is the the median price of the oracle submissions.

This issue could allow a minimum/maximum prices with an earlier slot to be used for the action when the current price's slot is after the action's slot.

```
impl Switchboard {
    #[allow(clippy::manual_inspect)]
    pub(super) fn check_and_get_price<'info>(
        clock: &Clock,
        token_config: &TokenConfig,
        feed: &'info AccountInfo<'info>,
    ) -> Result<(u64, i64, Price)> {
        let feed = AccountLoader::<SbFeed>::try_from(feed)?;
        let feed = feed.load()?;
        let (min_result_ts, _) = feed.current_result_ts_range();
        require_gte!(

    min_result_ts.saturating_add(token_config.heartbeat_duration().into()),
            clock.unix_timestamp,
            CoreError::PriceFeedNotUpdated
```

```
        );
        Ok((
            feed.result.slot,
            feed.result_ts(),
            Self::price_from(&feed, token_config)?,
        ))
    }
```

Recommendations:

Use the minimum slot for the minimum/maximum prices instead of `feed.result.slot`.

GMX Solana: Fixed in @914b63a09e8...

Zenith: Verified. Resolved by using `feed.result_land_slot` instead of `feed.result.slot` based on the design to use price publication time slot and not the price observation time slot.

[M-14] Anyone can prevent market removal from the GLV

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

Target

- programs/gmsol-store/src/instructions/glv/management.rs#L398-L402

Description:

The `remove_glv_market` employs the following check to assure the GLV's market token `vault` is empty before removing the market from the GLV:

```
require_eq!(
    ctx.accounts.vault.amount,
    0,
    CoreError::PreconditionsAreNotMet
);
```

However, anyone in possession of these market tokens can donate a minuscule amount to the `vault` which prevents its removal from the GLV.

This attack vector can become an issue if repeated for multiple markets since there is an upper limit of markets that can be supported by the GLV, see `MAX_ALLOWED_NUMBER_OF_MARKETS`.

Recommendations:

It is recommended to base the above check on internal accounting (from deposits and withdrawals) instead of the actual market token balance of the `vault`. In case there is a mismatch between internal accounting and the actual balance, the market tokens could be sent to the keeper on removal.

GMX Solana: Fixed in @2728a776e3...

Zenith: Verified. Resolved by storing the GLV market token balances in `GlvMarketConfigs`.

[M-15] Malicious TIMELOCK_KEEPER can set `is_signer` for another TIMELOCK_KEEPER's account to steal funds from it

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

Target

- /programs/gmsol-timelock/src/lib.rs#L46

Description:

`create_instruction_buffer` allows a `TIMELOCK_KEEPER` to create an instruction buffer that can be approved and then executed by another `TIMELOCK_KEEPER`.

However, it allows the creating `TIMELOCK_KEEPER` to pass in the `signers`, which are then marked as `is_signer` for the instruction execution.

As the executing `TIMELOCK_KEEPER` will have to sign the transaction during the `execute_instruction()`, it is possible for the creating `TIMELOCK_KEEPER` to set the account of the executing `TIMELOCK_KEEPER` as a signer too.

Example scenario,

1. Malicious Keeper A creates an instruction buffer X that performs a token transfer CPI from Keeper B to Keeper A, and then set Keeper B as the signer.
2. The instruction buffer is approved.
3. Once the instruction is executable, Keeper B simply execute the instruction X.
4. As Keeper B need to sign the transaction to execute instruction X, that means its signature will be passed on to the token transfer CPI within the instruction. This cause the token transfer to be executed, draining Keeper B's token account.

The assumption is that Keepers are bots that will execute instruction once its executable and approved, and they do not have the ability to verify the instruction content since its designed to be arbitrary. Despite that, the approval process will filter out such malicious instruction, making this a low probability.

```
/// Create instruction buffer.
 #[access_control(CpiAuthenticate::only(&ctx, roles::TIMELOCK_KEEPER))]
 pub fn create_instruction_buffer<'info>(
```

```
    ctx: Context<'_, '_, 'info, 'info, CreateInstructionBuffer<'info>>,
    num_accounts: u16,
    data_len: u16,
    data: Vec<u8>,
    signers: Vec<u16>,
) → Result<()> {
    instructions::unchecked_create_instruction_buffer(
        ctx,
        num_accounts,
        data_len,
        &data,
        &signers,
    )
}
```

Recommendations:

Remove the `signers` parameter and simply set the ExecutorWalletSigner PDA as the signer as that is the only signer required for `execute_instruction()`.

GMX Solana: Fixed in @65cc51ac7b..

Zenith: Verified. Resolved by updating `oad_and_init_instruction` to only allow executor wallet as the signer.

[M-16] Imbalanced incentive for ADL creates unfair position closures and increases insolvency risk

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

Target

- /programs/gmsol-store/src/instructions/exchange/position_cut.rs#L258
- /programs/gmsol-store/src/ops/order.rs#L1675-L1677

Description:

When ADL is enabled for a market, order keepers are able to call `auto_deleverage()` to reduce/close positions and bring the PnL factor down and reduces the insolvency risk.

When a ADL result in the closure of the positions, the order keeper will be rewarded with execution fee from `Order::position_cut_rent()` that is given as a refund in close_position(). On the other hand, order keepers are not rewarded for partial close for ADL; in fact they will incur the transaction fees, which are not compensated.

This imbalanced ADL incentive scheme creates three problems,

1. Owners are penalized despite healthy positions - Owners pays the execution fee in `position_cut_rent()` that is given to the keeper for closing their position during ADL, even when their position are healthy, unlike liquidations.
2. Unfair position closures - It drives order keepers to selectively close smaller positions for the reward, and not perform partial close on large whale positions.
3. Increase insolvency risk - When there are no small enough positions to close during ADL, the order keepers are disincentivized by transaction fees to partially close the large positions as they cannot be fully closed (large closure could fail as PnL factor can go below the minimum PnL factor). This will cause the PnL factor to continue to grow and exceed the max level, reaching a dangerous level till one of the large positions is fully closable.

Recommendations:

Consider changing the ADL incentives as such,

For (1), owners should not pay the reward for full closure of their position during ADL. This

ensures owners are not penalized for healthy positions and they will be refunded the `position_cut_rent()` when their positions are fully closed during ADL. This can simply be fixed as follows,

/programs/gmsol-store/src/instructions/exchange/position_cut.rs#L258

```
let refund = Order::position_cut_rent()?;
// Default to 0, then conditionally set if 'kind' is Liquidate
let mut refund = 0;
if let PositionCutKind::Liquidate = kind {
    refund = Order::position_cut_rent()?;
}
```

/programs/gmsol-store/src/ops/order.rs#L1675-L1677

```
    // we have to add a is_liquidation flag here so that executor gets
        back rent for ADL regardless of position closure
    if is_position_removed && is_liquidation {
    if is_position_removed {
        msg!("[Position] the position is removed");
    } else {
        msg!(
            "[Position] the position is not removed, setting the rent
receiver to the executor"
        );
        self.order
            .load_mut()?
            .header
            .set_rent_receiver(self.executor.key());
    }
```

For (2), consider having an ADL incentive based on the reduced PnL factor and to be paid by the LPs or treasury. This ensures that the keeper is rewarded solely based on the improved PnL factor and not based on number of closed positions. Even though this partially mitigates the unfairness, as keepers could focused on large positions, at least the owners are not penalized and pay for the reward.

For (3), keeper should be compensated for the ADL transaction fee regardless of full/partial closure. This ensures there are no disincentives for partial closure.

GMX Solana: Fixed in @1d4e0e158c...

Zenith: Verified. Resolved by not paying execution fees for both ADL full and partial closure, to ensure that ADL is conducted fairly regardless of position size, and users are not

penalized by the ADL execution fees.

[M-17] Malicious order keeper can perform risk-free trade using different price in the same tx

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

Target

- /programs/gmsol-store/src/states/oracle/feed.rs#L60-L87

Description:

`update_price_feed_with_chainlink()` allows the order keeper to update the custom price feed account using a signed price report from Chainlink Data Streams. The `observations_timestamp` is used to indicate the time that the price data was captured.

```
let mut price = Self {
    decimals: Report::DECIMALS - divisor_decimals,
    flags: Default::default(),
    padding: [0; 6],
    ts: i64::from(report.observations_timestamp),
    price: (price / divisor).try_into().unwrap(),
    min_price: (bid / divisor).try_into().unwrap(),
    max_price: (ask / divisor).try_into().unwrap(),
};
```

As Chainlink Data Streams provide subseconds resolution, it allows different prices to be obtained for the same `observations_timestamp`.

This allows a malicious order keeper to pick and choose the price report to pass into `update_price_feed_with_chainlink()` as there is no way to tell which report is newer/older when the reports have the same `observations_timestamp`. That will then allow the order keeper to choose the price for the order execution as the oracle price will be loaded from the custom price feed account.

The issue opens up an exploit where the order keeper can make risk-free trades as follows,

1. Malicious keeper create orders to prepare the attack.

- create "open long position" order with create_order().
- create "close long position" order with create_order().

2. Retrieve prices from Chainlink Datastream DON.

- Malicious keeper retrieve the first price A report.
- The keeper then wait and get next price B report.

3. Now the the keeper can choose which price to use and then execute the orders in the same transaction as follows,

- update_price_feed_with_chainlink() with lower price, i.e. min(price A, price B)
- execute_increase_or_swap_order() to execute "open long position" order with lower price
- update_price_feed_with_chainlink() with higher price, i.e. max(price A, price B)
- execute_decrease_order() to execute "close long position" order with higher price

Note that keeper will cancel both orders and re-try if the expected profit is less than the tx fees.

The attack does need the following pre-requisites, which makes it a low likelihood,

- Requires high volatility condition for the index token such that the price movement within the transaction is large enough to profit, though this is made easier with leverage and when market is volatile.
- Malicious order keeper has to be faster than other keepers, by frontrunning them. Technically possible as there is no incentives for honest keeper to execute orders quickly.

Recommendations:

This issue can be resolved by skipping updates for the custom price feed account if it had already been updated in the same slot.

```
pub(crate) fn update(&mut self, price: &PriceFeedPrice) -> Result<()> {
    let clock = Clock::get()?;
    let slot = clock.slot;
    let current_ts = clock.unix_timestamp;

    // skip if feed has already been updated in current slot
    if Clock::get()?.slot == accounts.price_feed.load()?.last_published_
        at_slot() {
        return Ok(());
    }

    // Validate time.
    require_gte!(
    require_gt!(
```

```
            slot,
            self.last_published_at_slot,
            CoreError::PreconditionsAreNotMet
        );
        require_gte!(
            current_ts,
            self.last_published_at,
            CoreError::PreconditionsAreNotMet
        );

        require_gte!(price.ts, self.price.ts, CoreError::InvalidArgument);
        require_gte!(price.max_price, price.min_price,
    CoreError::InvalidArgument);
        require_gte!(price.max_price, price.price,
    CoreError::InvalidArgument);
        require_gte!(price.price, price.min_price,
    CoreError::InvalidArgument);

        self.last_published_at_slot = slot;
        self.last_published_at = current_ts;
        self.price = *price;

        Ok(())
    }
```

GMX Solana: Fixed in @74b7b84c27...

Zenith: Verified. Resolved by adding the same validation for the position increase timestamp to the Market Decrease Order.

[M-18] Far future price timestamps can block feed updates

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

Target

- programs/gmsol-store/src/states/oracle/feed.rs#L77

Description:

On invocation of the `update_price_feed_with_chainlink` instruction, the subsequent `update` function of the `PriceFeed` only checks the new price's timestamp against the currently stored one:

```
require_gte!(price.ts, self.price.ts, CoreError::InvalidArgument);
```

Even though price timestamps are subject to further validation when the price data is used, the above check is the sole price timestamp validation at the point of updating the feed.

Consequently, price timestamps during feed updates can be detached from the present UNIX timestamp, i.e. (far) in the past or (far) in the future. A further impact of this check is that a single malicious price timestamp far in the future can block feed updates, therefore no genuine price timestamps will be accepted by the `update` function until that future timestamp is reached.

It is worth mentioning that there is a call to verify_report preceding the call to `update`. However, this verification call performs CPI and the destination code is foreign to this audit's scope. Therefore, it cannot be assumed to sufficiently mitigate the present timestamp issue.

Recommendations:

It is recommended to perform a plausibility check of price timestamps already at the feed update stage, similar to the `check_and_get_price` price functions whenever price data is used. To mitigate the mentioned DoS scenario it is crucial to reject timestamps that are too far in the future from the current UNIX timestamp.

GMX Solana: Fixed in @24340e071d...

Zenith: Verified. Resolved by adding plausibility checks regarding future timestamps.

[M-19] GLV shifts could be blocked by a high spread

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

Target

- gmsol-store/src/ops/glv.rs

Description:

Before a GLV shift can be executed, the price impact should be assessed to ensure that the destination market has a higher value than the current market or that the maximum price impact is not exceeded.

The issue is that the destination market uses the minimum price instead of the maximum price.

With a sufficiently high spread, this can lead to an unnecessary revert: the condition for shifting should depend solely on the price impact, independent of how high the spread is for a given market.

By using the maximized price for both the source and destination markets, we can ensure that the price impact check is strictly based on the same price, effectively removing the spread from the equation.

This approach allows the decision to be determined exclusively by the positive or negative price impact, aligning with the original GMX implementation.

Recommendations:

Consider maximizing the price when calculating the destination value:

```
    let (to_market_token_value, _, _) = get_glv_value_for_market(
        self.oracle,
        &mut prices,
        to_market.market(),
        received.into(),
        false,
        true
    )?;
```

GMX Solana: Verified. Resolved with @634499325a5...

Zenith: The issue was fixed as recommended.

[M-20] GLV max market value can be exceeded

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

Target

- gmsol-store/src/ops/glv.rs

Description:

When a GLV deposit is executed, the `market_pool_value` is checked to ensure that the new balance won't surpass the max limit, if this value is set.

The`pool_value` is calculated with maximize set to `false`. This is the expected behavior when calculating the total minted amount.

However, this value is used later to validate the token balance; but this check should use a `pool_value` calculated with maximize set to `true`, which uses the max price instead of the min price.

Due to this, it's possible to exceed the maximum limit if the spread is sufficiently high.

Recommendations:

Consider recalculating `market_pool_value` by calling `pool_value` with `maximize = true` instead of using the result of `get_glv_value_for_market`.

GMX Solana: Resolved with @ce04e7863e...

Zenith: Verified

[M-21] ADL can be DoS by spamming `update_adl_state()`

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

Target

- /programs/gmsol-store/src/states/market/utils.rs#L203-L226

Description:

Order keepers can call `update_adl_state()` for any market to enable/disable ADL based on whether the PnL factor is exceeded. This will also set the `latest_adl_time()` to the current time to indicate the ADL's last updated time.

```
fn update_adl_state(&mut self, oracle: &Oracle, is_long: bool) →
    Result<()> {
    ...
    self.set_adl_enabled(is_long, is_exceeded);

    let kind = if is_long {
        ClockKind::AdlForLong
    } else {
        ClockKind::AdlForShort
    };
    let clock = self
        .state
        .clocks
        .get_mut(kind)
        .ok_or_else(|| error!(CoreError::NotFound))?;
    *clock = Clock::get()?.unix_timestamp;
    Ok(())
}
```

When ADL is enabled and the order keeper calls `auto_leverage()`, it will perform `validate_adl()` to check that the provided oracle's timestamp is not earlier than the ADL's last updated time.

As the oracle timestamps are subtracted by a non-zero timestamp adjustment (which makes the timestamp older), it means that ADL can only be executed after a delay equivalent to the timestamp adjustment. For example, if timestamp adjustment is 1,

`auto_leverage()` can only be executed at least 1 sec after `update_adl_state()`.

```
fn validate_adl(&self, oracle: &Oracle, is_long: bool) → CoreResult<()> {
    if !self.is_adl_enabled(is_long) {
        return Err(CoreError::AdlNotEnabled);
    }
    if oracle.max_oracle_ts() < self.latest_adl_time(is_long)? {
        return Err(CoreError::OracleTimestampsAreSmallerThanRequired);
    }
    Ok(())
}
```

Due to how `latest_adl_time()` is set, it can be abused by a malicious order keeper to DoS `auto_leverage()` by simply spamming `update_adl_state()`. The DoS could also occur unintentionally due to race conditions when two keepers are trying to call `update_adl_state()` and `auto_leverage()` during the same period.

Attack Scenario

Suppose ADL is already enabled at `t = 0` with the following state,

- `latest_adl_time() == 0`
- `timestamp_adjustment == 2`
- `is_adl_enabled() == true`

1. At `t = 1`, Malicious Keeper A spam the first `update_adl_state()`, which sets `latest_adl_time() = 1`.
2. At `t = 2`, ADL is executable after the delay from `timestamp_adjustment`.
3. Now honest keeper B calls `auto_deleverage()` to peform an ADL, which will trigger `validate_adl()`.
   - Within it, the check `(max_oracle_ts < latest_adl_time())` will throw an error.
   - That is because `max_oracle_ts = t - 2 = 0` (due to `timestamp_adjustment`), which is less than `latest_adl_time() == 1`.
4. Malicious Keeper A can constantly spam `update_adl_state()` to DoS every subsequent ADL.

Note: The attack will be effective as long as the attack cost is much smaller than the potential loss from denying ADL.

Recommendations:

Only update the `latest_adl_time()` when the ADL enable state is changed.

GMX Solana: Fixed in @6e6ee94750a37...

Zenith: Verified. Resolved by adding a configurable `max_staleness` that can be used to offset `timestamp_adjustment` for the `latest_adl_time()` check in `validate_adl()`.

[M-22] `initialize_referral_code()` can be griefed using `transfer_referral_code()`

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

Target

- /programs/gmsol-store/src/states/user.rs#L151-L157

Description:

`initialize_referral_code()` allows an user to register a referral code of their choice and set it as their referral code. This allows the user to share the referral code with other users to sign up an account as referrees. `initialize_referral_code()` only allows referral code to be set for the user, if the user does not have any existing referral code (enforced in `set_code()` below).

However, the check in `set_code()` will allow attacker to prevent a victim from initializing their referral code by transferring a referral to the victim.

Example Scenario:

1. Victim initializes a new user account.
2. Attacker backruns the victim's tx in step 1 with `initialize_referral_code()` and `transfer_referral_code()` to set the referral code for victim.
3. Victim tries to call `initialize_referral_code()` to register and set a referral code.
4. As victim's referral code is set to the transferred code by attacker, the intended referral code initialization fails.

```
pub(crate) fn set_code(&mut self, code: &Pubkey) -> Result<()> {
    require_eq!(self.code, DEFAULT_PUBKEY,
    CoreError::ReferralCodeHasBeenSet);

    self.code = *code;

    Ok(())
}
```

Recommendations:

Make `transfer_referral_code()` opt-in by requiring the new owner to sign the transaction as well.

Alternatively, allow the user to override any existing referral code when calling `initialize_referral_code()`.

GMX Solana: Fixed in PR-81

Zenith: Verified. Resolved with a 2-step referral code transfer.

[M-23] forPositiveImpact Neglects To Incentivize Markets Without Price Impact Configuration

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

Description:

Throughout the codebase actions which move the market to a more balanced state are rewarded with lower fees. This is measured via checking if the price impact value is positive, however for markets where the price impact is configured to be zero this will fail to incentivize actions which balance the market.

Recommendations:

The GMX Solidity implementation has implemented balanceWasImproved logic to replace the price impact check to address this: https://github.com/gmx-io/gmx-synthetics/pull/93

GMX Solana: Acknowledged. When the price impact mechanism is disabled, the fairness of the parameters must be ensured.

4.3 Low Risk

A total of 24 low risk findings were identified.

[L-1] `final_short_token` doesn't have any constraints on a close withdrawal request

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

Target

- gmsol-store/src/instructions/exchange/withdrawal.rs

Description:

`CloseWithdrawal` uses a wrong constraint for the `final_short_token`, as it checks the long token instead of the short one. Effectively, the `final_short_token` has not been checked to be valid.

The impact is limited, as the `final_short_token_escrow` is also checked (which points to the correct `short_token` when the withdrawal request is created), so spoofing the address while canceling a withdrawal shouldn't be currently possible.

Recommendations:

Consider the following fix:

```
#[account(constraint = withdrawal.load()?.tokens.final_long_token() ==
    final_long_token.key() @ CoreError::TokenMintMismatched)]
#[account(constraint = withdrawal.load()?.tokens.final_short_token() ==
    final_short_token.key() @ CoreError::TokenMintMismatched)]
pub final_short_token: Box<Account<'info, Mint>>,
```

GMX Solana: @ea50664122...

Zenith: Verified.

[L-2] TIMELOCK_KEEPER can steal rent from others by calling `execute_instruction`

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

Target

- instruction_buffer.rs#L369

Description:

Upon executing an instruction, the `instruction` account is closed and the lamports (rent) from it are transferred to the `TIMELOCK_KEEPER` that calls the `execute_instruction`.

However, there are no mechanism that keep tracks of who paid the rent for that instruction.

If there are multiple `TIMELOCK_KEEPERs`, one could steal from another `TIMELOCK_KEEPER` just by executing the instruction created by others.

```
/// The acccounts definition for
    [`execute_instruction`](crate::gmsol_timelock::execute_instruction).
#[derive(Accounts)]
pub struct ExecuteInstruction<'info> {
    /// Authority.
    pub authority: Signer<'info>,
    /// Store.
    /// CHECK: check by CPI.
    pub store: UncheckedAccount<'info>,
    /// Timelock config.
    #[account(has_one = store)]
    pub timelock_config: AccountLoader<'info, TimelockConfig>,
    /// Executor.
    #[account(has_one = store)]
    pub executor: AccountLoader<'info, Executor>,
    /// Executor Wallet.
    #[account(
        mut,
        seeds = [Executor::WALLET_SEED, executor.key().as_ref()],
        bump,
    )]
    pub wallet: SystemAccount<'info>,
```

```
    /// Instruction to execute.
    #[account(mut, has_one = executor, close = authority)]
    pub instruction: AccountLoader<'info, InstructionHeader>,
    /// Store program.
    pub store_program: Program<'info, GmsolStore>,
}
```

Recommendations:

Consider keeping track of the `TIMELOCK_KEEPER` that paid the rent for `instruction` account and refund it to that `TIMELOCK_KEEPER` when `execute_instruction` is called.

GMX Solana: Fixed in @ea50664122...

Zenith: Verified. Resolved by tracking and refunding to `rent_receiver`.

[L-3] Wrong `INIT_SPACE` of `SwapExecuted`

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

Target

- programs/gmsol-store/src/events/swap.rs#L11-L23

Description:

The `INIT_SPACE` of `SwapExecuted` only accounts `1` instead of `32` bytes for the `market_token: Pubkey`.

Recommendations:

We recommend adjusting the `INIT_SPACE` accordingly, e.g. using `std::mem::size_of::<Self>()`.

GMX Solana: Fixed in @deaf1e6078ee...

Zenith: Verified. Resolved by revising manual init space computation.

[L-4] Unexpected transfer of treasury fees

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

Target

- programs/gmsol-store/src/lib.rs#L1542-L1544l
- programs/gmsol-store/src/instructions/market.rs#L734-L747

Description:

The documentation of the `claim_fees_from_market` instruction claims that:

> This instruction allows the fee receiver to claim accumulated fees from a market. The claimed amount is not immediately transferred - it remains in the market's balance and must be transferred in a separate instruction.

However, the actual implementation transfers the claimed fees, see `MarketTransferOutOperation`.

Recommendations:

It is recommended to adapt either the documentation or the implementation and add separate instruction for the fee transfer.

GMX Solana: Fixed in @63731cf0fc9...

Zenith: Verified. Resolved by adapting the documentation.

[L-5] Insufficient authority segregation on `Store` initialization

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

Target

- programs/gmsol-store/src/states/store.rs#L93-L103

Description:

When initializing the `Store`, the same `authority` account is utilized as authority for three distinct purposes:

- Store authority
- Treasury authority (fee receiver)
- Address authority (holding key); cannot be changed after initialization

```
pub fn init(&mut self, authority: Pubkey, key: &str, bump: u8) →
    Result<()> {
    self.key = crate::utils::fixed_str::fixed_str_to_bytes(key)?;
    self.key_seed = to_seed(key);
    self.bump = [bump];
    self.authority = authority;     // store
    self.treasury.init(authority);  // treasury
    self.amount.init();
    self.factor.init();
    self.address.init(authority);   // address
    Ok(())
}
```

Recommendations:

We recommend utilizing three distinct accounts/addresses for the above purposes.

GMX Solana: Fixed in @e639cc1bd9...

Zenith: Verified.

[L-6] Order keeper can DoS GLV Shift by executing dust amount shift

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

Target

- /programs/gmsol-store/src/instructions/glv/shift.rs#L112-L121

Description:

There is a GLV shift interval to prevent order keepers from performing GLV shifts at a high frequency.

However, this also makes it easier for a malicious order keeper to DoS legitimate shifts by creating and executing GLV shifts with dust amount to keep extending the shift interval.

Though likelihood is low as the malicious order keeper still needs to pay the execution fee to sustain the attack.

```
fn validate(&self, _params: &Self::CreateParams) → Result<()> {
    let glv = self.glv.load()?;
    let market_token = self.to_market_token.key();
    let is_deposit_allowed = glv
        .market_config(&market_token)
        .ok_or_else(|| error!(CoreError::Internal))?
        .get_flag(GlvMarketFlag::IsDepositAllowed);
    require!(is_deposit_allowed, CoreError::GlvDepositIsNotAllowed);
    glv.validate_shift_interval()
}
```

Recommendations:

Consider imposing a minimum shift value (in USD) for GLV shift to prevent spamming of dust shifts.

GMX Solana: Fixed in @ea50664122...

Zenith: Verified. Resolved by adding `shift_min_value` and validating it to prevent dust shifts.

[L-7] `CreateGlvShift` can be DoS by another order keeper

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

Target

- /programs/gmsol-store/src/instructions/glv/shift.rs#L57-L64

Description:

`glv_shift` PDA in `CreateGlvShift` is created based on the seeds `[GlvShift::SEED, store.key().as_ref(), glv.key().as_ref(), &nonce]`. The `nonce` ensures that unique PDA are created for each new shift in that particular GLV.

However, as `nonce` is a parameter for `create_glv_shift`, this makes it possible for order keepers to use the same nonce, and cause other GLV shifts to abort as `glv_shift` will fail to init.

That means a malicious order keeper could intentionally frontrun another GLV shift using the same nonce but with a small shift amount. That will prevent legitimate order keepers from rebalancing the GLV, though the malicious order keeper has to pay execution fee constantly to sustain the DoS.

```
/// The accounts definition for
    [`create_glv_shift`](crate::create_glv_shift) instruction.
#[derive(Accounts)]
#[instruction(nonce: [u8; 32])]
pub struct CreateGlvShift<'info> {
...
    /// GLV shift.
    #[account(
        init,
        payer = authority,
        space = 8 + GlvShift::INIT_SPACE,
        seeds = [GlvShift::SEED, store.key().as_ref(), glv.key().as_ref(),
    &nonce],
        bump,
    )]
    pub glv_shift: AccountLoader<'info, GlvShift>,
```

Recommendations:

Consider adding the order keeper account as part of the seed to ensure that `create_glv_shift` can be completed regardless of the nonce.

GMX Solana: Fixed in @ea50664122a...

Zenith: Verified.

[L-8] Missing constraining of `final_short_token` mint in `ExecuteWithdrawal` context

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

Target

- programs/gmsol-store/src/instructions/exchange/execute_withdrawal.rs#L66-L67

Description:

The constraint of the `final_short_token` mint in the `ExecuteWithdrawal` context accidentally constrains the `final_long_token` instead of the `final_short_token`, therefore leaving it effectively unconstrained.

```
#[account(constraint = withdrawal.load()?.tokens.final_long_token()
    == final_long_token.key() @ CoreError::TokenMintMismatched)]
pub final_short_token: Box<Account<'info, Mint>>,
```

Recommendations:

We recommend changing the constraint as follows:

```
#[account(constraint = withdrawal.load()?.tokens.final_long_token() ==
    final_long_token.key() @ CoreError::TokenMintMismatched)]
#[account(constraint = withdrawal.load()?.tokens.final_short_token() ==
    final_short_token.key() @ CoreError::TokenMintMismatched)]
pub final_short_token: Box<Account<'info, Mint>>,
```

GMX Solana: Fixed in @7f779c9fe8...

Zenith: Verified.

[L-9] Incorrect use of the `require_eq` macro for `Pubkey` values

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

Target

- programs/gmsol-store/src/events/trade.rs#L415
- programs/gmsol-store/src/instructions/gt.rs#L187-L191
- programs/gmsol-store/src/instructions/gt.rs#L293-L297
- programs/gmsol-store/src/instructions/gt.rs#L298-L302
- programs/gmsol-store/src/instructions/user.rs#L54
- programs/gmsol-store/src/instructions/user.rs#L55
- programs/gmsol-store/src/instructions/exchange/execute_order.rs#L58
- programs/gmsol-store/src/instructions/exchange/execute_order.rs#L72-L76
- programs/gmsol-store/src/instructions/exchange/execute_order.rs#L77-L81
- programs/gmsol-store/src/instructions/exchange/order.rs#L154
- programs/gmsol-store/src/instructions/exchange/order.rs#L155-L159
- programs/gmsol-store/src/instructions/exchange/order.rs#L160-L164
- programs/gmsol-store/src/instructions/exchange/order.rs#L165
- programs/gmsol-store/src/instructions/exchange/order.rs#L805-L809
- programs/gmsol-store/src/ops/order.rs#L320-L324
- programs/gmsol-store/src/ops/order.rs#L325-L329
- programs/gmsol-store/src/ops/order.rs#L399-L402
- programs/gmsol-store/src/ops/order.rs#L404-L408
- programs/gmsol-store/src/ops/shift.rs#L119-L123
- programs/gmsol-store/src/ops/shift.rs#L125-L129
- programs/gmsol-store/src/states/deposit.rs#L93-L96
- programs/gmsol-store/src/states/glv.rs#L133
- programs/gmsol-store/src/states/glv.rs#L173-L177
- programs/gmsol-store/src/states/glv.rs#L178-L182
- programs/gmsol-store/src/states/glv.rs#L266-L270
- programs/gmsol-store/src/states/glv.rs#L272-L276
- programs/gmsol-store/src/states/glv.rs#L406-L410
- programs/gmsol-store/src/states/glv.rs#L424-L428
- programs/gmsol-store/src/states/glv.rs#L731-L735
- programs/gmsol-store/src/states/glv.rs#L769-L773

- programs/gmsol-store/src/states/user.rs#L107-L111
- programs/gmsol-store/src/states/user.rs#L152
- programs/gmsol-store/src/states/user.rs#L160
- programs/gmsol-store/src/states/common/swap.rs#L255
- programs/gmsol-store/src/states/common/swap.rs#L296
- programs/gmsol-store/src/states/common/swap.rs#L358
- programs/gmsol-store/src/states/market/mod.rs#L296
- programs/gmsol-store/src/states/market/mod.rs#L399-L404
- programs/gmsol-store/src/states/market/mod.rs#L406-L410
- programs/gmsol-store/src/states/market/revertible/revertible_position.rs#L32-L37
- programs/gmsol-store/src/states/market/revertible/swap_market.rs#L367
- programs/gmsol-store/src/states/oracle/feed.rs#L119
- programs/gmsol-store/src/states/oracle/mod.rs#L373
- programs/gmsol-store/src/utils/token.rs#L58-L61
- programs/gmsol-store/src/utils/token.rs#L83
- programs/gmsol-store/src/utils/token.rs#L86
- programs/gmsol-store/src/utils/token.rs#L251
- programs/gmsol-store/src/utils/token.rs#L252-L255
- programs/gmsol-timelock/src/instructions/instruction_buffer.rs#L214-L218
- programs/gmsol-timelock/src/instructions/instruction_buffer.rs#L316-L320
- programs/gmsol-treasury/src/instructions/gt_bank.rs#L97-L101
- programs/gmsol-treasury/src/instructions/gt_bank.rs#L102-L106
- programs/gmsol-treasury/src/instructions/gt_bank.rs#L360
- programs/gmsol-treasury/src/instructions/gt_bank.rs#L378-L381
- programs/gmsol-treasury/src/instructions/treasury.rs#L673
- programs/gmsol-treasury/src/instructions/treasury.rs#L675
- programs/gmsol-treasury/src/instructions/treasury.rs#L682
- programs/gmsol-treasury/src/instructions/treasury.rs#L683-L687

Description:

In the above instances, the `require_eq` macro is used to check the equality of two `Pubkey` values. However, this macro is explicitly not designed for this purpose according to the documentation:

> Ensures two NON-PUBKEY values are equal.

Recommendations:

It is recommended to use the `require_keys_eq` macro instead in the above instances.

GMX Solana: Fixed in @8f1061796d...

Zenith: Verified.

[L-10] Unsorted Referral Reward Factors Validation

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

Target

- gt.rs#L151

Description:

The `set_referral_reward_factors` function currently does not validate whether the input slice of referral reward factors is sorted. This could lead to inconsistencies in the ranking scheme, as the factors are expected to reflect a hierarchical structure (e.g., higher ranks should have equal or greater rewards than lower ranks). Without this validation, unsorted factors could disrupt the intended logic of the ranking system.

Recommendations:

To address the issue, the following modifications can be made to the `set_referral_reward_factors` function:

1. **Add a Sorting Check:**
   Ensure the input slice is sorted in ascending order before proceeding. This will maintain the integrity of the ranking scheme.

```
// Check if the input slice is sorted in ascending order
if !factors.windows(2).all(|window| window[0] ≤ window[1]) {
    return Err(CoreError::UnsortedFactors.into());
}
```

GMX Solana: Resolved with @bcba3a680e...

Zenith: Verified.

[L-11] Limit orders can't specify an execution time

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

Target

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

Description:

In TradFi, limit orders typically include the option to specify a time frame for execution. While the original GMX implementation supported this feature, it is currently impossible in GMX-sol with the current implementation.

This limitation significantly restricts the flexibility and functionality available to traders when using limit orders.

Recommendations:

Consider the following:

1. Add a `min_target_time` to `Order`
2. Return an error when the oracle time is less than `min_target_time` when executing a limit order

GMX Solana: Resolved with @828c0ac48f...

Zenith: Verified.

[L-12] ADL cannot be triggered when `min_pnl_factor` is reached

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

Target

- gmsol-store/src/ops/order.rs

Description:

ADL execution should be considered valid until the pnl factor is under the min pnl factor (which should be inclusive). However, this doesn't happen as the operation will fail when `new_pnl_factor == min_pnl_factor`.

Recommendations:

Consider the following change:

```
    require_gt!(
    require_gte!(
        pnl_factor_after_execution,
        min_pnl_factor,
        CoreError::InvalidAdl
    );
```

GMX Solana: Resolved with @4920e4350e1...

Zenith: Verified.

[L-13] Limit orders don't have a deadline

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

Target

- gmsol-store/src/ops/order.rs

Description:

The current workflow for any type of order (deposits, withdrawals, swaps, etc.) is as follows:

1) A user creates an order. 2) An order keeper must execute the order for it to take effect.

Solana had multiple outages, often lasting 4+ hours, occurring at least once a year. If an outage happens between these steps, it could impact users. When the cluster comes back online, orders may be executed at unexpected prices.

Recommendations:

Consider adding an optional deadline to the order, so that it can't be executed once it's due.

GMX Solana: For market actions, there is a global timeout mechanism.

For limit orders, having a deadline option would indeed be better, but we do not plan to provide it in the current version. Even without a deadline, since the acceptable price will be validated, users will at least not execute their orders at an unexpected price.

Zenith: The issue was acknowledged for limit orders.

[L-14] Missed validation for Account Length in `load_instruction`

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

Target

- programs/gmsol-timelock/src/states/instruction.rs

Description:

In the `load_instruction` function, the **account length stored in the instruction header is not validated against the actual length of the accounts section**. This can lead to **inconsistent instructions**, potentially causing unexpected behavior.

Code Snippet:

```
fn load_instruction(&self) → Result<InstructionRef> {
    // Check the instruction header account
    self.load()?;

    let data = self.as_ref().try_borrow_data()?;

    let (_disc, remaining_data) = Ref::map_split(data, |d| d.split_at(8));
    let (header, remaining_data) = Ref::map_split(remaining_data, |d| {
        d.split_at(std::mem::size_of::<InstructionHeader>())
    });
    let header = Ref::map(header,
    bytemuck::from_bytes::<InstructionHeader>);
    let data_len = usize::from(header.data_len);


    let (data, accounts) = Ref::map_split(remaining_data,
    |d| d.split_at(data_len));

    Ok(InstructionRef {
        header,
        data,
```

```
            accounts,
        })
    }
```

Recommendations:

Validate that the account length stored in the instruction header matches the actual length of the accounts section. Specifically:

1. **Add a validation check**:

“‘rust let expected_accounts_len = header.num_accounts ; require!(expected_accounts_len == accounts.len(), CustomError::InvalidAccountsLength); “‘

GMX Solana: Resolved with @7fcfbabcc3...

Zenith: Verified.

[L-15] Misleading core error `UnknownOrDisabledToken`

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

Target

- programs/gmsol-store/src/states/token_config.rs#L568
- programs/gmsol-store/src/states/common/swap.rs#L108
- programs/gmsol-store/src/states/common/token.rs#L100
- programs/gmsol-treasury/src/states/gt_bank.rs#L125

Description:

In the above instances, the `TokenConfig` of a given token is obtained using the getter function of the `TokenMap`.

In case the specified token is not found in the `TokenMap`, the `UnknownOrDisabledToken` error is returned. This is misleading since the `TokenConfig` of a disabled token can be obtained regardless, therefore the error can only occur in the case of an unknown token.

Recommendations:

It is recommended to rename this core error to `UnknownToken`.

GMX Solana: Fixed in @6f7e83a7bc...

Zenith: Verified.

[L-16] Invalid future oracle price timestamps pass verification

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

Target

- programs/gmsol-store/src/states/oracle/chainlink.rs#L43-L45
- programs/gmsol-store/src/states/oracle/feed.rs#L124-L126
- programs/gmsol-store/src/states/oracle/pyth.rs#L30-L41
- programs/gmsol-store/src/states/oracle/switchboard.rs#L34-L38

Description:

In the above instances of the `check_and_get_price` function (Chainlink, internal feed, Pyth and Switchboard), a maximum age of the price's timestamp is checked to avoid stale oracle data. However, any future price timestamp greater than the current UNIX timestamp will pass these checks.

Recommendations:

It is recommended to perform plausibility checks of price timestamps rejecting price data with invalid future timestamps.

GMX Solana: Fixed in @24340e071...

Zenith: Verified. Resolved by adding plausibility checks regarding future timestamps.

[L-17] Missing implementation of GT reserve

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

Target

- programs/gmsol-store/src/states/gt.rs#L62
- programs/gmsol-store/src/states/gt.rs#L122
- programs/gmsol-store/src/constants/mod.rs#L68-L69

Description:

The `GtState` struct has a `reserve_factor` field which is also initialized with a default value. However, this `reserve_factor` field has no further usage within the codebase.

Recommendations:

It is recommended to either implement a GT reserve as seemingly intended or remove the `reserve_factor` field.

GMX Solana: Fixed in @28dfd62efc...

Zenith: Verified.Resolved by removing the `reserve_factor` field.

[L-18] Overrestrictive `treasury_vault_config` check on treasury withdrawals can leave old vaults inaccessible

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

Target

- programs/gmsol-treasury/src/instructions/treasury.rs#L480-L500

Description:

The currently authorized `treasury_vault_config` can be replaced by a new one via the `set_treasury_vault_config` instruction. Therefore, many protocol operations such as treasury deposits, swaps and GT exchanges/buybacks are rightfully restricted to the authorized `treasury_vault_config`.

However, also treasury withdrawals are affected by this restriction allowing only withdrawal from vaults which are owned the by the current `treasury_vault_config`. Since there is no migration of treasury vaults when setting a new config, there are potentially still funds held in the vaults associated with the previous config. These funds will be inaccessible due to the config restriction, i.e. withdrawal will fail.

Recommendations:

It is recommended to remove the restriction to the currently authorized `treasury_vault_config` from the `WithdrawFromTreasuryVault` context and therefore allow withdrawal using previous configs.

GMX Solana: Fixed in @f64fc114226...

Zenith: Verified.

[L-19] Missing validation for `min_output` in `Order::update()`

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

Target

- order.rs#L642-L644

Description:

`Order::update()` allows the user to update the order parameters for limit orders/swaps before it is executed.

However, it allows the user to set the `min_output = 0` for limit swap, which is inconsistent with `init_swap()` that requires `min_output ≠ 0`.

In the unlikely case that `min_output` was updated to zero, it allows order keeper to execute the swap that could gives zero output.

```
pub(crate) fn update(&mut self, id: u64, params: &UpdateOrderParams) →
    Result<()> {
    let current = &mut self.params;
    require!(current.is_updatable()?, CoreError::InvalidArgument);
    require!(!params.is_empty(), CoreError::InvalidArgument);

    self.header.id = id;

    if let Some(size_delta_value) = params.size_delta_value {
        current.size_delta_value = size_delta_value;
    }

    if let Some(acceptable_price) = params.acceptable_price {
        current.acceptable_price = acceptable_price;
    }

    if let Some(trigger_price) = params.trigger_price {
        current.trigger_price = trigger_price;
    }

    if let Some(min_output) = params.min_output {
```

```
        current.min_output = min_output;
    }
```

Recommendations:

Suggest to add a check to ensure `min_output ≠ 0` when updating orders.

GMX Solana: Fixed in @1fecba2a48b...

Zenith: Verified.

[L-20] Account seeds based on `index` of type `u8` might limit long-term protocol operation

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

Target

- programs/gmsol-store/src/instructions/oracle/custom.rs#L29
- programs/gmsol-store/src/instructions/exchange/execute_order.rs#L44
- programs/gmsol-store/src/instructions/glv/management.rs#L47
- programs/gmsol-treasury/src/instructions/treasury.rs#L48

Description:

In the above instances (`InitializePriceFeed`, `PrepareTradeEventBuffer`, `InitializeGlv`, `InitializeTreasuryVaultConfig`) the accounts' seeds contain an `index` parameter of type `u8`. Assuming other seed parameters (accounts) like `store` and `config` are constant during the lifetime of the protocol, the above instances only allow 256 distinct instances of GLV tokens, vault configs, etc. to be initialized.

This upper limit on account instances might turn out to be insufficient during the long-term operation of the protocol.

Recommendations:

It is recommended to adjust the type of `index` from `u8` to `u16` or greater in the above instances.

GMX Solana: Fixed in @001b308785b...

Zenith: Verified. Resolved by adjusting the type from `u8` to `u16` in all above instances.

[L-21] There is no feature flag to disable deposits and withdrawals

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

Target

- gmsol-store/src/utils/internal/action.rs

Description:

Unlike orders, deposits and withdrawals cannot be paused. While this feature is present in the original GMX implementation, it is missing in GMX-Solana, which could potentially lead to issues if an attack could be mitigated by pausing either of these actions.

Recommendations:

Consider implementing a feature flag to enable/disable deposits and withdrawals.

GMX Solana: Resolved with @4763b473e5b...

Zenith: Verified.

[L-22] Missing rent exempt check on withdrawal creation

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

Target

- gmsol-store/src/ops/withdrawal.rs

Description:

When a withdrawal action is created, there aren't any checks to ensure that the amount sent by the user is rent-exempt after deducting the execution fees, unlike other actions.

Recommendations:

Consider implementing a rent-exempt check, e.g. by calling `validate_balance`.

GMX Solana: Withdrawal implements the Action trait, so the relevant check has already been automatically implemented through the ActionExt trait. Perhaps we should remove the code you pointed out, as they should already be inconsistent and unnecessary.

Zenith: Verified. Resolved by removing the code.

[L-23] Order keepers are wrongfully charged for GLV shifts

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

Target

- gmsol-store/src/instructions/glv/shift.rs

Description:

GLV shifts can only be created by order keepers; nevertheless, they are required to pay execution fees for this action with the following code path: `create_impl` > `execute` > `validate_params`, as the minimum execution lamports are non-zero.

This enables a scenario where other order keepers can 'snipe' these fees for an action that is normally free of charge in the original GMX implementation.

Recommendations:

Consider readjusting the logic so that order keepers aren't forced to pay to create a GLV shift. The easiest solution is to set the minimum execution lamports for GLV shifts to zero:

```
impl Action for GlvShift {
    const MIN_EXECUTION_LAMPORTS: u64 = 200_000;
    const MIN_EXECUTION_LAMPORTS: u64 = 0;

    fn header(&self) -> &ActionHeader {
        &self.shift.header
    }
}
```

GMX Solana: Resolved with @fded1be1145...

Zenith: Verified.

[L-24] Users overpay a position's rent on pure markets

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

Target

- gmsol-store/src/states/order.rs

Description:

When a new position is prepared, users are charged rent, which is calculated based on the token account size:

```
    /// Get rent for position cut.
    pub fn position_cut_rent() → Result<u64> {
        use anchor_spl::token::TokenAccount;

        let rent = Rent::get()?;
        let amount = rent.minimum_balance(Self::INIT_SPACE + 8)
→           + rent.minimum_balance(TokenAccount::LEN) * 2
            + Self::MIN_EXECUTION_LAMPORTS;

        Ok(amount)
    }
```

The issue is that this implies all positions are non-pure since the token accounts are always considered as two. However, users can open positions on pure markets that share the same token account, effectively making them pay twice as much in this case.

Recommendations:

Consider charging users the correct rent amount, i.e. half of the current value, if there is a pure market.

GMX Solana: Fixed in @b52cd6523e0...

Zenith: Verified.

4.4 Informational

A total of 8 informational findings were identified.

[I-1] The project relies on vulnerable crate dependencies

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

Description:

During a scan of this project's `Cargo.lock` file, the following vulnerable dependencies could be identified:

Timing variability in *curve25519-dalek*'s *Scalar29::sub*/*Scalar52::sub*

```
Crate:     curve25519-dalek
Version:   3.2.1
Title:     Timing variability in `curve25519-dalek`'s
    `Scalar29::sub`/`Scalar52::sub`
Date:      2024-06-18
ID:        RUSTSEC-2024-0344
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to ⩾4.1.3
```

Double Public Key Signing Function Oracle Attack on *ed25519-dalek*

```
Crate:     ed25519-dalek
Version:   1.0.1
Title:     Double Public Key Signing Function Oracle Attack on
    `ed25519-dalek`
Date:      2022-06-11
ID:        RUSTSEC-2022-0093
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0093
Solution:  Upgrade to ⩾2
```

Borsh serialization of HashMap is non-canonical

```
Crate:     hashbrown
Version:   0.15.0
Title:     Borsh serialization of HashMap is non-canonical
Date:      2024-10-11
ID:        RUSTSEC-2024-0402
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0402
Solution:  Upgrade to ⩾0.15.1
```

Recommendations:

If possible regarding compatibility, it is suggested to update the affected dependencies to a non-vulnerable version.

GMX Solana: Fixed in @d882bbff1...

Zenith: `hashbrown` could be upgraded without introducing conflicts.

[I-2] Inconsistency of market vault token account seeds

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

Target

- programs/gmsol-store/src/instructions/token.rs#L33-L37

Description:

When a market vault's token account is initialized, its seed is given a follows:

```
seeds = [
    constants::MARKET_VAULT_SEED,
    store.key().as_ref(),
    mint.key().as_ref(),
],
```

However in 25 different instances in the codebase, the market vault's token account seeds are given with a trailing `&[]`, for example:

```
seeds = [
    constants::MARKET_VAULT_SEED,
    store.key().as_ref(),
    vault.mint.as_ref(),
    &[],
],
```

Recommendations:

It is recommended to remove the trailing `&[]` from the seeds in the 25 instances where `constants::MARKET_VAULT_SEED` is used.

GMX Solana: Fixed in @ca8425f8e6e...

Zenith: Verified.

[I-3] Unused discount when minting GT

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

Target

- programs/gmsol-store/src/states/order.rs#L577

Description:

When minting GT, the `get_mint_amount` function has a `discount` parameter.

However, the `discount` parameter is hardcoded to zero.

Recommendations:

It is recommended to utilize or remove the discount when minting GT.

GMX Solana: Fixed in @7a36b45b115...

Zenith: Verified. Resolved by removing discount functionality.

[I-4] Unused accounts in `swap.rs`

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

Target

- /programs/gmsol-treasury/src/instructions/swap.rs#L58-L64
- /programs/gmsol-treasury/src/instructions/swap.rs#L265
- /programs/gmsol-treasury/src/instructions/swap.rs#L269

Description:

`gmsol-treasury/src/instructions/swap.rs` has the following unused accounts for the `CreateSwap` and `CancelSwap` structs.

```
pub struct CreateSwap<'info> {
    ...
    /// Swap out token receiver vault.
    #[account(
        mut,
        associated_token::authority = receiver,
        associated_token::mint = swap_out_token,
    )]
    pub swap_out_token_receiver_vault: Account<'info, TokenAccount>,
    ...
}
```

```
pub struct CancelSwap<'info> {
    ...
    /// The vault for swap in token.
    /// CHECK: check by CPI.
    #[account(mut)]
    pub swap_in_token_vault: UncheckedAccount<'info>,

    /// The vault for swap out token.
    /// CHECK: check by CPI.
    #[account(mut)]
    pub swap_out_token_vault: UncheckedAccount<'info>,
```

```
    ...
}
```

Recommendations:

Remove these unused accounts.

GMX Solana: Fixed in @447c5b257e...

Zenith: Verified.

[I-5] Use of outdated Pyth Solana Receiver Rust SDK

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

Target

- Cargo.toml#L44

Description:

The protocol depends on the outdated v0.3.2 of the Pyth Solana Receiver Rust SDK. However, there is already v0.5.0 which contains substantial changes such as audit fixes, see changelog.

Recommendations:

It is recommended to upgrade to a newer version of the Pyth Solana Receiver Rust SDK, if possible concerning compatibility.

GMX Solana: Fixed in @956caa67dc...

Zenith: Verified.

[I-6] Incorrect error used for `event_loader` failure

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

Target

- /programs/gmsol-store/src/instructions/exchange/execute_order.rs#L334-L337
- /programs/gmsol-store/src/ops/order.rs#L941-L944

Description:

The error for `event_loader.as_ref()` uses the wrong `CoreError::PositionIsRequired` message, which is meant for position loader.

```
if should_send_trade_event {
    let event_loader = accounts.event.clone();
    let event = event_loader
        .as_ref()
        .ok_or_else(|| error!(CoreError::PositionIsRequired))?
        .load()?;
    let event = TradeEventRef::from(&*event);
    event_emitter.emit_cpi(&event)?;
}
```

Recommendations:

Provide a relevant error message for `event_loader`.

GMX Solana: Fixed in @ea58eac4294...

Zenith: Verified.

[I-7] `unchecked_insert_factor` is callable by both `CONFIG_KEEPER` and `MARKET_KEEPER`

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

Target

- gmsol-store/src/instructions/config.rs

Description:

The documentation wrongly states that `unchecked_insert_factor` is only callable by `CONFIG_KEEPER`, but it can be also called by `MARKET_KEEPER` through `insert_order_fee_discount_for_referred_user`.

Recommendations:

Consider updating the documentation.

GMX Solana: Fixed in @a94a8bc84f...

Zenith: Verified.

[I-8] Use of outdated Switchboard On-Demand libraries

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

Target

- Cargo.toml#L81
- Cargo.toml#L82

Description:

The protocol depends on `v0.2.2` of the `switchboard-on-demand` library as well as on `v0.2.9` of the `switchboard-on-demand-client` library. However, there are already substantially newer versions available, i.e. `v0.3.4` of the `switchboard-on-demand` library and `v0.2.12` of the `switchboard-on-demand-client` library.

Recommendations:

Is recommended to upgrade to a newer version of the Switchboard On-Demand libraries, if possible concerning compatibility.

GMX Solana: Fixed in @c11ddf20934...

Zenith: Verified.