Mango v4

Security Assessment

[not yet delivered] — Prepared by OtterSec

Nicola Vela nick0ve@osec.io

Harrison Green hgarrereyn@osec.io

Robert Chen r@osec.io

Table of Contents

Executive Summary **3**

Overview 3

Key Findings 3

Scope **4**

Findings **5**

Vulnerabilities **6**

OS-MNG-ADV-00 | Inadequate Access Control Inside HealthRegions 8

OS-MNG-ADV-01 | [0.24.2] Oracle Data Staleness 10

OS-MNG-ADV-02 | [0.22.0] Allocator Can Overflow Into Program Area 11

OS-MNG-ADV-03 | Inadequate Sanitization Of Token Indexes 12

OS-MNG-ADV-04 | [0.17.0] Insufficient Verification Of Pyth Oracle Status 14

OS-MNG-ADV-05 | [0.17.0] Possible Borrow In Token Force Closing 15

OS-MNG-ADV-06 | [0.18.0] Not Reserving TokenPositions 16

OS-MNG-ADV-07 | [0.19.0] Swap Fee Acts As A Rebate 17

OS-MNG-ADV-08 | [0.19.0] Account Resizing May Be Blocked 18

OS-MNG-ADV-09 | [0.20.0] TCS Premium Auction Can Leave Window 20

OS-MNG-ADV-10 | [0.22.0] Missing ATA Ownership Check 21

General Findings **22**

OS-MNG-SUG-00 | Improve Safety Checks On Admin-Only Instructions 24

OS-MNG-SUG-01 | Broken Assumption Of USDC Peg To USD 26

OS-MNG-SUG-02 | Incorrect Check When Copying String Data 27

OS-MNG-SUG-03 | Replace Panics With Descriptive Errors 28

OS-MNG-SUG-04 | Improve Clarity Of Token Index Check 29

OS-MNG-SUG-05 | Net Deposits Calculation May Be Misleading 30

OS-MNG-SUG-06 | [0.17.0] Calculating Buyback Fees Accrued Checks 31

OS-MNG-SUG-07 | [0.17.0] Consider Merging Functions 32

OS-MNG-SUG-08 | [0.17.0] Address Anti-pattern In Edit Instructions 33

OS-MNG-SUG-09 | [0.17.0] Improve Comment Clarity 34

OS-MNG-SUG-10 | [0.17.0] Loss Of Accrued Buyback Fees 35

OS-MNG-SUG-11 | [0.19.0] Swap Fee May Be Bypassed 36

OS-MNG-SUG-12 | [0.20.0] (nit) Simplify TCS Assertion 37

OS-MNG-SUG-13 | [0.20.0] Remove Liqee Token Account Deactivation in TCS Trigger 38

OS-MNG-SUG-14 | [0.20.0] (nit) Align Naming Convention 39

OS-MNG-SUG-15 | [0.20.0] Defensive Assert in TCS Start 40

OS-MNG-SUG-16 | [0.20.0] Inconsistent Maker Fee Computation 41

OS-MNG-SUG-17 | [0.24.0] Incorrect IxGate Variant 43

Appendices

Vulnerability Rating Scale **44**

Procedure **45**

01 — Executive Summary

Overview

Mango engaged OtterSec to perform an assessment of the `mango-v4` program. We performed an initial audit of the core program code and subsequent follow-up audits on specific release candidates. This document contains findings from all reviews. The assessments were conducted between February 22nd, 2023 and April 11th, 2024. For more information on our auditing methodology, see Appendix B.

All the issues were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation.

Key Findings

Over the course of this audit engagement, we produced 29 findings in total.

In particular, we found an issue related to HealthRegion, which allows multiple instructions to execute without performing health checks. This may allow malicious users to create debt and immediately liquidate it in the same transaction, stealing funds from the protocol (OS-MNG-ADV-00). We also noted an issue with token registrations that may lead to a denial of service (OS-MNG-ADV-03).

Additionally, we discovered an issue in how the protocol handles USD to USDC conversions (OS-MNG-SUG-01). We also made recommendations about the general quality of life issues (OS-MNG-SUG-09) and returning unclear errors to the users (OS-MNG-SUG-03).

Overall, we commend the Mango team for being responsive and knowledgeable throughout the audit.

02 — Scope

The source code was delivered to us in a git repository at https://github.com/blockworks-foundation/mango-v4. The initial audit was performed against commit 5c7a2e3. Subsequent audits on specific release candidates were performed as follows:

- **0.17.0** - 9bd3913
- **0.18.0** - d98bf23
- **0.19.0** - e37f1ed
- **0.20.0** - a41a82e
- **0.21.0** - d6f46be
- **0.22.0** - e107b28
- **0.23.0** - 6b13841
- **0.24.0** - 0dc9d9a
- **0.24.1** - 28a6368
- **0.24.2** - 0a55f46

Mango-v4 is a decentralized exchange built on Solana, offering various on-chain cryptocurrency trading options.

Users may build accounts with a variety of positions. However, every position must be over-collateralized so that the protocol may manage the risk of margin trading through permissionless liquidations.

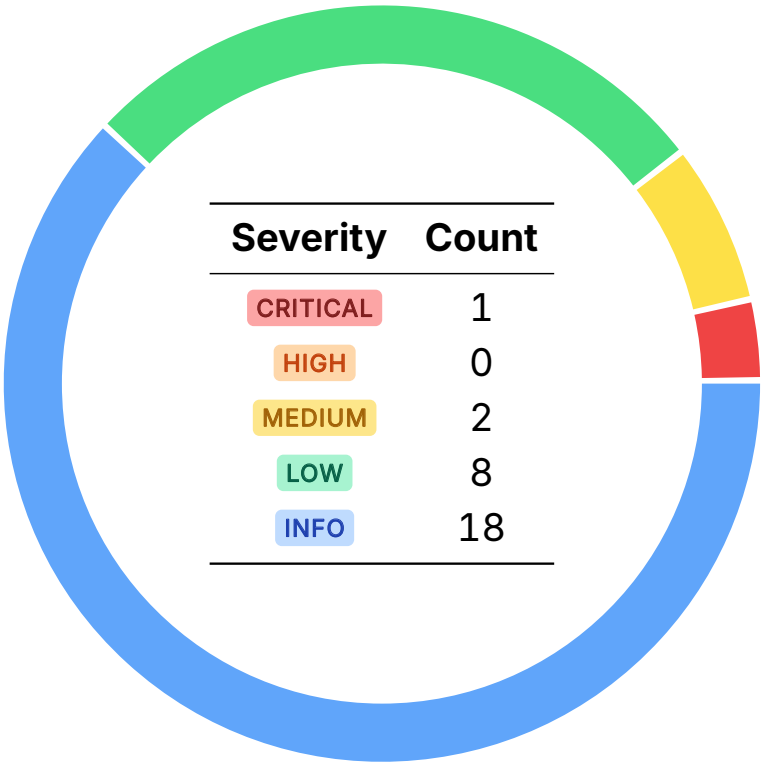
Its key features include:

1. Borrowing and lending capabilities, offering users to lend and earn interest based on how many users borrow.
2. Perpetual futures trading across various assets through its own orderbook.
3. Spot trading through Openbook DEX.

03 — Findings

Overall, we reported 29 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	1
HIGH	0
MEDIUM	2
LOW	8
INFO	18

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-MNG-ADV-00	CRITICAL	RESOLVED	Malicious users may exploit the lack of access controls within `HealthRegions` to generate bad debt and immediately liquidate it within a single transaction.
OS-MNG-ADV-01	MEDIUM	RESOLVED	[0.24.2] `get_pyth_on_demand_state` utilizes `posted_slot` for checking oracle staleness, which may be unreliable since users may post outdated prices.
OS-MNG-ADV-02	MEDIUM	RESOLVED	[0.22.0] The custom allocator is missing an upper bound check and could overwrite the program region.
OS-MNG-ADV-03	LOW	RESOLVED	A token may be registered with a reserved value of , resulting in user token accounts becoming interpreted as deactivated.
OS-MNG-ADV-04	LOW	RESOLVED	[0.17.0] The protocol does not adequately check the `status` on Pyth oracles, potentially resulting in the use of stale data.

OS-MNG-ADV-05	LOW	RESOLVED	[0.17.0] If asset and liability tokens are in `force-close` mode, the current implementation may result in a scenario where the liquidation candidate `liqee` incurs a borrow on the asset token.
OS-MNG-ADV-06	LOW	RESOLVED	[0.18.0] `TokenConditionalSwap`s do not reserve token positions, which may result in triggers failing.
OS-MNG-ADV-07	LOW	RESOLVED	[0.19.0] The flash loan swap fee acts as a rebate in certain cases.
OS-MNG-ADV-08	LOW	RESOLVED	[0.19.0] Rent transfer during account resizing does not consider externally transferred lamports.
OS-MNG-ADV-09	LOW	RESOLVED	[0.20.0] A TCS premium auction can trade at prices outside the start window.
OS-MNG-ADV-10	LOW	RESOLVED	[0.22.0] The `token_withdraw` instruction does not validate the ownership of the user's Associated Token Account.

Inadequate Access Control Inside HealthRegions CRITICAL OS-MNG-ADV-00

Description

`HealthRegion` allows multiple instructions to be executed without performing health checks, thereby reducing gas costs, as long as the account remains in a healthy position at the end. However, this feature also introduces new attack surfaces. For instance, an attacker may bypass the limitations of the `FlashLoan` implementation, which prevents users from calling Mango after receiving a flash loan.

To execute a flash loan without using `FlashLoanBegin/Start`, an attacker may:

1. Borrow any quantity of tokens within a `HealthRegion`, which would render their account in a state that is unchecked and unhealthy.
2. Conduct certain operations using the borrowed tokens.
3. Deposit the tokens back to restore their account to a healthy state.

Furthermore, rather than returning the borrowed tokens to restore the health of the account, an attacker may exploit the `TokenLiqBankruptcy` instruction to liquidate the bad debt. This enables the attacker to force the protocol to cover the liquidation cost, resulting in the unauthorized appropriation of funds.

Proof of Concept

To demonstrate how this attack may be carried out, let us consider a scenario where the attacker has two accounts: A and B. Account A will serve as the liquidatee, while Account B will be the liquidator. The attacker may execute a transaction with the following instructions:

1. `HealthRegionBegin(A)` - `pre_init_health` is zero.
2. `A.TokenWithdraw(amt)` - `A` where `amt` is the amount of tokens to borrow from the `liab_bank`. As A is within the `HealthRegion`, it may proceed into a liquidable stage without restriction.
3. `B.TokenLiqBankruptcy(A)` - B liquidates A through the `token_liq_bankruptcy` instruction. The protocol repays A's liabilities, and `post_init_health` becomes zero.
4. `HealthRegionEnd(A)`.

This results in the attacker stealing `amt` tokens from the bank.

Remediation

Update the current `HealthRegion` implementation to prevent liquidations and withdrawals from being executed inside it by leveraging the existing instruction introspection logic.

Patch

Resolved in b22a1e7 by explicitly allowing only the instructions necessary to interact with the perpetual and spot markets inside `HealthRegions`.

diff

```
+   let allowed_inner_ix = [
+       crate::instruction::PerpCancelAllOrders::discriminator(),
+       crate::instruction::PerpCancelAllOrdersBySide::discriminator(),
+       crate::instruction::PerpCancelOrder::discriminator(),
+       crate::instruction::PerpCancelOrderByClientOrderId::discriminator(),
+       crate::instruction::PerpPlaceOrder::discriminator(),
+       crate::instruction::PerpPlaceOrderPegged::discriminator(),
+       crate::instruction::Serum3CancelAllOrders::discriminator(),
+       crate::instruction::Serum3CancelOrder::discriminator(),
+       crate::instruction::Serum3PlaceOrder::discriminator(),
+       crate::instruction::Serum3SettleFunds::discriminator(),
+       crate::instruction::Serum3SettleFundsV2::discriminator(),
+   ];
```

[0.24.2] Oracle Data Staleness MEDIUM OS-MNG-ADV-01

Description

In `oracle::get_pyth_on_demand_state`, the `posted_slot` from the `price_account` is utilized to determine the last update slot of the oracle. However, utilizing `posted_slot` as an indicator for the freshness of the data may be problematic as users may post outdated price updates. There is no inherent check in the current setup to ensure that the `posted_slot` reflects a recent or valid price update, which allows for old or stale data to be posted as if it were recent.

RUST

```rust
pub fn get_pyth_on_demand_state(
    acc_info: &(impl KeyedAccountReader + ?Sized),
    base_decimals: u8,
) -> Result<OracleState> {
    [...]
    let last_update_slot = price_account.posted_slot;
    require_gte!(price, 0);
    Ok(OracleState {
        price,
        last_update_slot,
        deviation,
        oracle_type: OracleType::PythV2,
    })
}
```

Remediation

Compare the last price publication time (`price_account.price_message.publish_time`) with the current program clock time.

Patch

Fixed in 0a55f46.

[0.22.0] Allocator Can Overflow Into Program Area MEDIUM OS-MNG-ADV-02

Description

In commit v0.22.0, a custom allocator was introduced to address limitations in the default allocator. The default allocator lacks the ability to leverage additional heap space requested for a transaction.

However the custom allocator could technically overflow into the program region if an allocation of size bigger than `0x100000000` is requested.

Remediation

Implement a check in the allocator to ensure that the requested allocation size does not exceed a safe upper-bound limit.

Patch

Fixed in afc2ff9.

Inadequate Sanitization Of Token Indexes LOW OS-MNG-ADV-03

Description

The protocol does not adequately check the provided `token_index` when registering a token. While the value `TokenIndex::MAX` is reserved to represent a disabled `TokenPosition`, it is not explicitly disallowed from being registered. As a result, if token registration occurs with `TokenIndex::MAX`, all user token accounts for that token become interpreted as deactivated, and utilizing them will result in an error.

RUST

```
impl Default for TokenPosition {
    ...
        TokenPosition {
            ...
            token_index: TokenIndex::MAX,
            ...
        }
    ...
}
```

This may result in all user token accounts for this token being interpreted as deactivated, and attempting to trade them will result in an error.

Moreover, there is inconsistent behavior between `is_active_for_token` and `is_active`. Specifically, when `token_index == TokenIndex::MAX`, then `is_active_for_token` returns true, while `is_active` returns false.

RUST

```
impl TokenPosition {
    ...
    pub fn is_active(&self) -> bool {
        self.token_index != TokenIndex::MAX
    }

    pub fn is_active_for_token(&self, token_index: TokenIndex) -> bool {
        self.token_index == token_index
    }
    ...
}
```

Similar behavior occurs for the `Serum3MarketIndex` and `PerpMarketIndex` indices, which have a reserved `MAX` value reserved for deactivated entries.

Remediation

Disallow the reserved value `TokenIndex::MAX` for token registration. Additionally, modify the inconsistent behavior between `is_active_for_token` and `is_active`.

diff

```
    pub fn token_register(
        ...
        token_index: TokenIndex,
        ...
    ) -> Result<()> {
        ...
+       require_neq!(token_index, TokenIndex::MAX);
        ...
    }

    pub fn token_register_trustless(
        ...
        token_index: TokenIndex,
        ...
    ) -> Result<()> {
        ...
+       require_neq!(token_index, TokenIndex::MAX);
        ...
    }

    pub fn is_active_for_token(&self, token_index: TokenIndex) -> bool {
-       self.token_index == token_index
+       self.token_index == token_index && self.token_index != TokenIndex::MAX
    }
```

Patch

Resolved in 99360e6.

[0.17.0] Insufficient Verification Of Pyth Oracle Status LOW OS-MNG-ADV-04

Description

The Pyth oracle suggests validating the `status` on the `PriceAccount` and utilizing the returned price only when the status is `Trading`. This practice is particularly crucial for markets not operating round-the-clock, like US equities.

Remediation

Ensure that the `status` of the Pyth oracle is `Trading` to guard against utilizing outdated data under unusual market circumstances.

Patch

Fixed in baaec4e.

diff

```
+            if price_account.agg.status != pyth_sdk_solana::PriceStatus::Trading {
+                msg!(
+                    "Pyth price status isn't 'Trading': status: {}",
+                    price_data.status as u64
+                );

+                return Err(MangoError::OracleStale.into());
+            }
```

[0.17.0] Possible Borrow In Token Force Closing LOW OS-MNG-ADV-05

Description

When both the asset and liability tokens are in `force-close` mode, chain-liquidation may occur, providing additional fees for liquidators. Although check three in `TokenForceCloseBorrowsWithToken` ensures that the `health` of the `liqee` does not decrease, preventing flip-flopping between two tokens, it may be more beneficial to verify that the asset bank is not also in `force-close` mode.

Remediation

Modify the implementation of `TokenForceCloseBorrowsWithToken` to ensure the asset bank is not in the `force-close` mode. This may prevent the occurrence of chain-liquidation and any undue advantage given to liquidators at the expense of the `liqee`.

Patch

Fixed in fe84c6a.

diff

```
+        require!(
+            !asset_bank.are_borrows_reduce_only(),
+            MangoError::TokenInReduceOnlyMode
+        );
```

[0.18.0] Not Reserving TokenPositions LOW OS-MNG-ADV-06

Description

TokenConditionalSwaps (TCS) were introduced in 0.18.0 as a new type of order that occurs when the price falls within a certain user-specified window. These orders allow users to create stop-losses and other similar types of conditional orders.

Order fulfillment occurs asynchronously through a liquidator-initiated trigger instruction. Upon meeting the price criterion, a TCS is eligible to be triggered via `TokenConditionalSwapTrigger`, which performs the swap and updates the state of the TCS order, removing it if it has been fully completed.

Completing a TCS requires the `liqee`, a user who placed the TCS, to have a token position for both sides of the swap. If these token positions do not yet exist, they will be created during `TokenConditional-SwapTrigger`. However, since the construction of a TCS does not reserve token positions, it is possible that at the time of a trigger, the `liquee` may have no available spots for new token positions, resulting in the transaction failing.

Since this error occurs in a liquidator-initiated instruction rather than a user-initiated one, from the user's perspective, it may appear as though the TCS is simply failing to be filled. The specific issue may be difficult to pinpoint without looking through liquidator logs. Certain types of TCS orders, such as stop-losses, are somewhat time-sensitive, and failing to fill these timely would be detrimental to system users.

Remediation

Reserve a token position for both sides of a TCS during creation so that it will not fail to trigger.

Patch

Fixed in 348fef8.

[0.19.0] Swap Fee Acts As A Rebate LOW OS-MNG-ADV-07

Description

A flash loan swap fee was recently introduced, intended to be applied to deposits that occur during the flash loan. However, the fee is incorrectly applied both to positive change amounts (deposits) and negative change amounts (withdraws):

```rust
let swap_fee = if flash_loan_type == FlashLoanType::Swap {
    change.amount * I80F48::from_num(max_swap_fee_rate)
} else {
    I80F48::ZERO
};
```

The result of this logic is that a fee is properly collected on deposits, but it actually acts as a rebate on withdraws.

Remediation

Ensure the fee is never negative (indicating a rebate).

Patch

Fixed in #693 along with a restructuring of the fee type.

[0.19.0] Account Resizing May Be Blocked LOW OS-MNG-ADV-08

Description

When increasing the size of Mango accounts, it may be necessary to transfer more lamports into the account to meet the new rent exemption requirements. `AccountExpand` attempts to perform this transfer but fails to consider the situation where there are more lamports than expected in the user account.

For example, consider the following code in `AccountExpand`:

```rust
let new_space = MangoAccount::space(
    token_count,
    serum3_count,
    perp_count,
    perp_oo_count,
    token_conditional_swap_count,
);
let new_rent_minimum = Rent::get()?.minimum_balance(new_space);

let realloc_account = ctx.accounts.account.as_ref();
let old_space = realloc_account.data_len();
let old_lamports = realloc_account.lamports();

require_gt!(new_space, old_space);

// transfer required additional rent
anchor_lang::system_program::transfer(
    anchor_lang::context::CpiContext::new(
        ctx.accounts.system_program.to_account_info(),
        anchor_lang::system_program::Transfer {
            from: ctx.accounts.payer.to_account_info(),
            to: realloc_account.clone(),
        },
    ),
    new_rent_minimum - old_lamports,
)?;
```

If a user directly transfers lamports into the Mango account (outside of Mango instructions), it is possible that `old_lamports` is actually larger than `new_rent_minimum`. In this case, the subtraction will overflow, and the instruction will fail.

Remediation

Do not attempt to transfer more lamports if the target account already meets the new rent exemption requirements.

Patch

Fixed in #694.

[0.20.0] TCS Premium Auction Can Leave Window LOW OS-MNG-ADV-09

In v0.20.0, two new TCS types were introduced: “linear auction” and “premium auction”. The original TCS type was renamed “fixed premium”. In short, the purpose of a TCS is to allow users to place orders that will be executed at some point in the future, when the oracle price is above or below a threshold for example. To incentive liquidators to trigger the TCS (perform the trade), creators of a TCS offer a premium on top of the oracle price.

In the original implementation of TCS (fixed premium), this premium was specified as a fixed percentage of the oracle price. However, this may be inefficient. Users risk overpaying on premium when a smaller one would have sufficed, and risk underpaying resulting in a TCS that doesn’t trigger. The two new TCS types address this inefficiency.

The linear auction allows users to specify a start and end time for the auction and a start and end price. The oracle price is ignored and the price offered by the TCS is simply the linear interpolation of the start and end price while the auction is ongoing. This format is stateless as the offered price is simply a function of time, and the user’s have full control over the price window in which the trade can execute.

The premium auction instead operates like a fixed premium where the premium rate linearly ramps up over time. Users specify a price window in which the auction can start (like fixed premium) and specify a minimum and maximum premium rate and duration to ramp up. In order to start a premium auction, the oracle price must first fall into the specified window at which point a user can invoke `TokenConditionalSwapStart` to mark the TCS as “active.” Once active, the TCS can be triggered with a premium price computed based on the current oracle price and how long the auction has been running.

Importantly, while the TCS can only start when the oracle price falls in the specified window, there is no similar mechanism to prevent trading when the oracle price leaves the window. Since the premium price is computed as a factor of the current oracle price, the TCS may end up trading at prices that fall considerably outside the specified window.

Out of window execution does not pose a threat to the protocol as triggering a TCS is gated on a health check, however it may cause users of the protocol to perform a bad trade unexpectedly if they are not aware of the behavior.

Patch

Implementing stop/restart functionality presents a challenge as those instructions need to be incentivized (similar to start). Authors acknowledge that out-of-window execution can happen and will provide info/warning for users.

[0.22.0] Missing ATA Ownership Check LOW OS-MNG-ADV-10

In the `token_withdraw` instruction, the program expects to send funds to the Mango account owner. However, it does not validate whether the owner field of the Associated Token Account is indeed the genuine owner.

```rust
let owner_ata = associated_token::get_associated_token_address(
    &account.fixed.owner,
    &ctx.accounts.vault.mint,
);
require_keys_eq!(
    ctx.accounts.token_account.key(),
    owner_ata,
    MangoError::DelegateWithdrawOnlyToOwnerAta
);
```

Remediation

Perform a check to ensure that owner field of `ctx.accounts.token_account` matches the owner.

Patch

Fixed in 719aee3.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-MNG-SUG-00	Admin-only instructions with incorrect parameters may result in unexpected behavior.
OS-MNG-SUG-01	The protocol assumes that one USDC always pegs to one USD, which may not always be true.
OS-MNG-SUG-02	`fill_from_str` utilizes 31 bytes out of 32 bytes.
OS-MNG-SUG-03	Multiple instances of unwrapping may result in panic for users.
OS-MNG-SUG-04	Utilize named constants instead of magic numbers to improve code clarity.
OS-MNG-SUG-05	Utilizing different oracle prices at different time periods may result in misleading net deposit calculations.
OS-MNG-SUG-06	[0.17.0] `reduce_buyback_fees_accrued` may be called with an amount larger than the sum of `buyback_fees_accrued_current` and `*_previous`.
OS-MNG-SUG-07	[0.17.0] `expire_buyback_fees` and `accrue_buyback_fees` may be more effective if merged.
OS-MNG-SUG-08	[0.17.0] The current implementation of `token_edit` may result in dangerous outcomes due to its order of operations.
OS-MNG-SUG-09	[0.17.0] Comments in `perp_liq_force_cancel_orders` may result in confusion due to their ambiguity.

OS-MNG-SUG-10	[0.17.0] Users may lose accrued buyback fees when the group interval changes from zero to a positive value.
OS-MNG-SUG-11	[0.19.0] The flash loan swap fee may be bypassed.
OS-MNG-SUG-12	[0.20.0] (nit) Simplify an assertion in TCS trigger
OS-MNG-SUG-13	[0.20.0] Remove the attempt to deactivate liqee token accounts during TCS trigger
OS-MNG-SUG-14	[0.20.0] (nit) Use liqor/liqee notation for TCS start
OS-MNG-SUG-15	[0.20.0] Add a defensive assert in TokenConditionalSwapStart
OS-MNG-SUG-16	[0.20.0] Inconsistent maker fee computation in TCS trigger
OS-MNG-SUG-17	[0.24.0] Incorrect IxGate variant for HealthCheck instruction

Improve Safety Checks On Admin-Only Instructions OS-MNG-SUG-00

Description

Various admin-gated instructions to configure the protocol do not perform adequate validation on the parameters supplied, potentially resulting in future vulnerabilities. While this may not pose an immediate security risk, it leaves the system open to exploitation in the future.

One example is the `init_base_asset_weight` parameter for editing a perpetual market utilizing the `perp_edit_market` instruction. This parameter must be positive; however, creating the perpetual market through the `perp_create_market` instruction does not enforce this requirement.

RUST

```
pub fn perp_create_market(
        ...
        init_base_asset_weight: f32,
        ...
) -> Result<()> {
        ...
        let mut perp_market = ctx.accounts.perp_market.load_init()?;
        *perp_market = PerpMarket {
        ...
        init_base_asset_weight: I80F48::from_num(init_base_asset_weight),
        ...
        };
}

pub fn perp_edit_market(
        ...
        init_base_asset_weight_opt: Option<f32>,
        ...
) -> Result<()> {
        ...
        require_gte!(
                init_base_asset_weight,
                0.0,
                MangoError::InitAssetWeightCantBeNegative
        );
        ...
}
```

Remediation

Implement additional rigorous validation checks on the parameters supplied to admin-only instructions to prevent future vulnerabilities. Specifically, for `perp_create_market`, ensure that the `init_base_asset_weight` parameter is positive.

Patch

Improved checks will be implemented gradually.

Broken Assumption Of USDC Peg To USD OS-MNG-SUG-01

Description

The protocol assumes that one USDC always pegs to one USD. However, in exceptional events such as market instability or other macroeconomic factors, the peg may break down, and the value of USDC may deviate from one USD.

The protocol employs a `StubOracle` that sets the price at one to determine the value of USDC/USD. When calculating Token/USDC pricing, the protocol utilizes Token/USD oracles and derives the USDC pricing directly from the USD pricing without any intermediate conversion.

This approach may result in loss of funds for users if the USDC peg is unstable and arbitrage bots are taking advantage of the price discrepancy between USDC and USD.

Remediation

Implement a more robust and adaptable pricing mechanism for USDC that accounts for the possibility of a broken USDC peg.

Patch

USDC is no longer assumed to peg to one USD; rather, it is treated like any other token. As such, a de-peg event would not have knock-on effects on other tokens on the platform.

Incorrect Check When Copying String Data OS-MNG-SUG-02

Description

`fill_from_str` is frequently used throughout the codebase to copy string data into 32-byte-sized buffers. However, the function requires input data to be < 32 bytes instead of <= 32 bytes. As a result, it never utilizes the byte at index 31.

Remediation

Require input data to be less than or equal to 32 bytes.

Patch

Resolved in 658a220.

diff

```
  pub fn fill_from_str<const N: usize>(name: &str) -> Result<[u8; N]> {
    let name_bytes = name.as_bytes();
-   require!(name_bytes.len() < N, MangoError::SomeError);
+   require!(name_bytes.len() <= N, MangoError::SomeError);
    let mut name_ = [0u8; N];
    name_[..name_bytes.len()].copy_from_slice(name_bytes);
    Ok(name_)
  }
```

Replace Panics With Descriptive Errors OS-MNG-SUG-03

Description

The codebase contains multiple instances of unwrapping, which may cause panics that are difficult for users to understand. For example, invoking `token_add_bank` while the maximum number of banks has already been added may result in panic.

Remediation

Utilize descriptive errors that provide clear information about what produced the error.

Patch

The issue is acknowledged; more descriptive errors will gradually be integrated into the codebase.

Improve Clarity Of Token Index Check OS-MNG-SUG-04

Description

In `token_register_trustless`, there is a check to ensure that the token index is not equal to zero using the `require_neq!` macro. However, this check may be unclear to developers and difficult to understand without additional context.

Remediation

Use the `QUOTE_TOKEN_INDEX` constant, which is more descriptive of the intended behavior.

diff

```
pub fn token_register_trustless(
    ctx: Context<TokenRegisterTrustless>,
    token_index: TokenIndex,
    name: String,
) -> Result<()> {
-   require_neq!(token_index, 0);
+   require_neq!(token_index, QUOTE_TOKEN_INDEX);
```

Patch

Resolved in 99360e6.

Net Deposits Calculation May Be Misleading OS-MNG-SUG-05

Description

When depositing and withdrawing tokens into a bank, the `net_deposits` field of the user's `MangoAccount` is updated by adding or subtracting the deposited or withdrawn token values in USD.

Multiplying the amount of tokens by the oracle price at the time of the operation calculates this value. However, the oracle price utilized to calculate the value of tokens deposited or withdrawn may not be the same as the one used in the previous operation, which may result in misleading `net_deposits` calculations.

Consider the following scenario:
A user deposits one SOL when its value is $100 and later withdraws one SOL when its value is $10. In this case, the `net_deposits` field would display a result of $90, despite the user not having made any actual deposits into the bank.

Remediation

Ensure the `net_deposits` field is zero to avoid such discrepancies.

Patch

This feature is working as intended, and improvement to the documentation is planned to clarify the intent.

[0.17.0] Calculating Buyback Fees Accrued Checks OS-MNG-SUG-06

Description

Currently, it is possible to call `reduce_buyback_fees_accrued` with an amount that exceeds the total of `buyback_fees_accrued_current` and `*_previous`. Even though `AccountBuybackFeesWithMango` usually prevents this logic, it would be safer to add more checks to disallow such scenarios from occurring.

Remediation

Utilize checked math for subtraction in `reduce_buyback_fees_accrued`. This will prevent the function from being called with an amount larger than the sum of `buyback_fees_accrued_current` and `*_previous`.

Patch

Fixed in 4dc0e71.

[0.17.0] Consider Merging Functions OS-MNG-SUG-07

Description

Currently, `expire_buyback_fees` and `accrue_buyback_fees` are implemented as separate functions. However, these functions are often called in succession to ensure that the timestamp is updated correctly.

For instance, in the perpetual flow, `expire_buyback_fees` is invoked when an order is *placed*, but `accrue_buyback_fees` is called when an order is *filled*.

If the duration between the order placement and filling exceeds the group interval, accrued fees for a user may be inadvertently stored "in the past" (when the user's `buyback_fees_expiry_timestamp` is outdated). Consequently, when the user attempts to buy back the fees, the `expire_buyback_fees` call in `AccountBuybackFeesWithMango` may reset accrued fees to zero.

Remediation

Merge `expire_buyback_fees` and `accrue_buyback_fees` into a single function. This prevents potential loss of accrued fees and ensures the timestamp is updated correctly.

Patch

Fixed in 12c9c35.

[0.17.0] Address Anti-pattern In Edit Instructions OS-MNG-SUG-08

Description

There is a potential anti-pattern in the `token_edit` implementation.

>_ *token_edit.rs* RUST

```rust
            if let Some(force_close) = force_close_opt {
            if force_close {
                require!(bank.reduce_only > 0, MangoError::SomeError);
            }
            msg!(
                "Force close: old - {:?}, new - {:?}",
                bank.force_close,
                u8::from(force_close)
            );
            bank.force_close = u8::from(force_close);
            require_group_admin = true;
        };
    }
```

It may be dangerous to check stateful requirements while editing the state of an object. In this specific scenario, it works, but any changes in the order of updates may result in incorrect results. For instance, the check would break if the code updated `force_close` before `reduce_only`.

Remediation

Group necessary invariants and check them together in the end after performing all the possible updates. For instance, another invariant to assert could be that the asset weight is always less than the liability weight, possibly by a certain factor. This recommendation also applies to `PerpEditMarket` and `Serum3EditMarket`.

[0.17.0] Improve Comment Clarity OS-MNG-SUG-09

Description

The comments in the `perp_liq_force_cancel_orders` and `serum3_liq_force_cancel_orders` may be interpreted differently and not accurately reflect the conditions for the cancellation of orders that may occur.

>_ *perp_liq_force_cancel_orders.rs* RUST

```rust
    // Early return if if liquidation is not allowed or if market is not in force close
    let liquidatable = account.check_liquidatable(&health_cache)?;
    let can_force_cancel = !account.fixed.is_operational()
        }} liquidatable == CheckLiquidatable::Liquidatable
        }} perp_market.is_force_close();
    if !can_force_cancel {
        return Ok(());
    }
```

>_ *serum3_liq_force_cancel_orders.rs* RUST

```rust
    // Early return if if liquidation is not allowed or if market is not in force close
    ...
        let liquidatable = account.check_liquidatable(&health_cache)?;
        let can_force_cancel = !account.fixed.is_operational()
            }} liquidatable == CheckLiquidatable::Liquidatable
            }} serum_market.is_force_close();
        if !can_force_cancel {
            return Ok(());
        }
```

Remediation

Improve the clarity of the comments to correctly represent the conditions for canceling orders. The comments should indicate similar to "Orders may be canceled if: 1. the account is frozen, 2. the account is liquidatable, or 3. the market is in force-close mode."

Patch

Fixed in af0bb41.

[0.17.0] Loss Of Accrued Buyback Fees

OS-MNG-SUG-10

Description

When `expire_buyback_fees` is first invoked after modifying `buyback_fees_expiry_interval`, the user's `buyback_fees_expiry_timestamp` is likely to be a past timestamp. As a result, both the `_current` and `_previous` buyback fees accrued will be reset, resulting in a loss for the user.

Remediation

Implement a `buyback_fees_expiry_interval_start` storage on the group that records the timestamp of the last group interval update. Then, verify this value within `expire_buyback_fees` to prevent premature expiration of buyback fees.

[0.19.0] Swap Fee May Be Bypassed

OS-MNG-SUG-11

Description

A fee was introduced to flash loans involving exactly two banks (i.e. “swaps”). The fee is intended to be collected on deposits during the flash loan.

Flash loans in Mango have an associated `FlashLoanType`, which may be `UNKNOWN` or `SWAP`. The only requirement for this parameter is that if a flash loan is marked as a `SWAP`, there must be exactly two banks. Apart from this requirement, users may select either type when issuing the instruction. This parameter is intended to be a UI parameter, and it is set correctly when users interact with the Mango program through the client.

The recently introduced swap fee uses this parameter to check whether or not to apply the fee. However, since the parameter is freely adjustable by the user, an observant user may simply set the flash loan type to `UNKNOWN` to avoid paying the fee. While this is not normally possible through the Mango client, users can construct flash loan instructions outside of the client to adjust the `FlashLoanType` parameter.

Remediation

Ensuring that all swap-type flash loans pay the fee is not entirely straightforward. Even by properly classifying all two-bank flash loans as a `SWAP`, users may avoid the fee by including a third “dummy” bank. Therefore, we recommend considering a more general-purpose fee for all flash loans or an alternative economic constraint to incentivize users to properly classify flash loans as a `SWAP`.

Patch

Fixed in #693 by introducing a general-purpose deposit fee on flash loans.

Reverted in #754.

[0.20.0] (nit) Simplify TCS Assertion

OS-MNG-SUG-12

Consider replacing the condition:

```rust
if tcs_is_expired {
    if min_buy_token > 0 {
        require!(!tcs_is_expired, MangoError::TokenConditionalSwapExpired);
    }
    ...
}
```

with the simplified:

```rust
if tcs_is_expired {
    require!(min_buy_token == 0, MangoError::TokenConditionalSwapExpired);
    ...
}
```

This pattern occurs in the `token_conditional_swap_trigger` implementation.

Patch

Resolved in bb6b4b9.

[0.20.0] Remove Liqee Token Account Deactivation in TCS Trigger OS-MNG-SUG-13

After transferring funds during a TCS trigger, the instruction checks if it is possible to deactivate the liqee or liqor token positions for either the buy or sell tokens. This may be possible for example when transferring funds would bring the account position close to zero and no other account orders have a lock on the token position.

RUST

```rust
if !liqee_buy_active {
    liqee.deactivate_token_position_and_log(liqee_buy_raw_index, liqee_key);
}
if !liqee_sell_withdraw.position_is_active {
    liqee.deactivate_token_position_and_log(liqee_sell_raw_index, liqee_key);
}
```

However, in this case the currently executing TCS instruction will always have a lock on both the liqee token positions. The token positions will be unlocked when the TCS is destroyed (completed or cancelled). Therefore, this optimistic deactivation is currently dead code.

Patch

Resolved in 15537ea.

[0.20.0] (nit) Align Naming Convention OS-MNG-SUG-14

Most instructions that operate on two accounts simultaneously involving the transfer of funds, including TCS trigger, use the convention of naming the signing account the `liqor` and the target account the `liqee`.

The newly introduced `TokenConditionalSwapStart` instruction is used to initiate the start of a TCS premium auction after the oracle price falls in a certain window. Accounts that call this instruction to start the premier auction are incentivized to do so by a small fee that is paid by the user who created the TCS auction.

Currently, the instruction uses the naming convention of `caller` for the signing account and `account` for the target account. We recommend adopting the same `liqor` / `liqee` notation as other related instructions.

Patch

Resolved in d482650.

[0.20.0] Defensive Assert in TCS Start

OS-MNG-SUG-15

During the execution of `TokenConditionalSwapStart`, the user who created the TCS pays a small incentive fee to the user who invokes the instruction. This fee is tracked as part of the total paid amount in the TCS struct and it should not overflow the designated `max_sell` parameter as configured by the TCS creator.

While it does not currently appear possible that the fee can overflow this parameter, it would be safe to add a defensive assertion such as:

```rust
assert!(tcs.sold <= tcs.max_sell);
```

to future-proof this code against potential changes to the incentive computation.

Patch

Resolved in e531b4f.

[0.20.0] Inconsistent Maker Fee Computation OS-MNG-SUG-16

Currently in TCS trigger, there is a discrepancy with the way the maker fee is computed for the `liqee` and `liqor`.

Specifically, for a given *premium price* the instruction computes the adjusted *maker price* whereby the price is adjusted slightly higher. This represents the price that the `liqee` will see. Given this price and the current TCS limits, the instruction computes the maximum amount to trade via a call to `trade_amount` resulting in two values:

1. `buy_token_amount`: amount of BUY tokens to transfer
2. `sell_token_amount_with_maker_fee`: amount of SELL tokens to withdraw from the liqee

In this case, the `liqee` transfers necessary SELL tokens to match the premium price and additionally pays a maker fee on the top. Computing the TCS bounds with the adjusted maker price rather than the premium price allows the fee to be properly accounted for without exceeding limits (i.e. transferring too much).

From the `liqor`'s perspective, they should receive necessary SELL tokens to match the premium price *minus* a taker fee. This is currently computed by *recalculating* the number of sell tokens to transfer assuming the normal premium price.

```rust
buy_token_amount, sell_token_amount_with_maker_fee = trade_amount(...);

sell_token_amount = buy_token_amount * premium_price
maker_fee = sell_token_amount * maker_rate
taker_fee = sell_token_amount * taker_rate
```

Finally, the amount of SELL tokens to transfer to the `liqor` can be computed as:

```rust
to_liqor = sell_token_amount_with_maker_fee - maker_fee - taker_fee;
```

The intuition is that subtracting the maker fee should cancel out the added maker fee as a result of the `trade_amount()` call. However, these two maker fees are computed in slightly different ways and subject to different rounding behavior. These differences result in the `liqor` and `liqee` observing slightly different fees during the transaction.

This discrepancy does not result in mishandling of funds as the only decision happening is how much of the transaction to give to the protocol (as fees) rather than transfer between `liqee` and `liqor`. The instruction performs correct bookkeeping on the realized fees and it is not possible for the `liqor` to receive more than the `liqee` pays.

However, it would be advisable to change this computation logic such that the `maker_fee` is computed consistently from both the `liqee` and `liqor`'s perspectives.

Patch

Resolved in 33add65.

[0.24.0] Incorrect IxGate Variant

OS-MNG-SUG-17

The 0.24.0 update introduced a new instruction called `HealthCheck`. Its purpose is to fail if the health isn't greater than or equal to the specified parameter. However, the instruction uses the wrong IxGate variant in the `constraint` macro.

diff

```
#[derive(Accounts)]
pub struct HealthCheck<'info> {
    #[account(
-   constraint = group.load()?.is_ix_enabled(IxGate::SequenceCheck) @ MangoError::IxIsDisabled,
+    constraint = group.load()?.is_ix_enabled(IxGate::HealthCheck) @ MangoError::IxIsDisabled,
    )]
    pub group: AccountLoader<'info, Group>,
    ...
}
```

Patch

Resolved in 0dc9d9a.

A — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.