Mango

Mango v4

Audit

Presented by:

OtterSec contact@osec.io

Nicola Vela nick0ve@osec.io

Harrison Green hgarrereyn@osec.io

Contents

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 and July 21st, 2023. 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 16 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-01).

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-05) 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 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

Mango-v4 is a decentralized exchange built on Solana, offering a variety of on-chain trading options for cryptocurrencies.

Users may build accounts with a variety of positions. However, every position has to be over-collateralized so that the risk associated with margin trading can be managed by the protocol through permissionless liquidations.

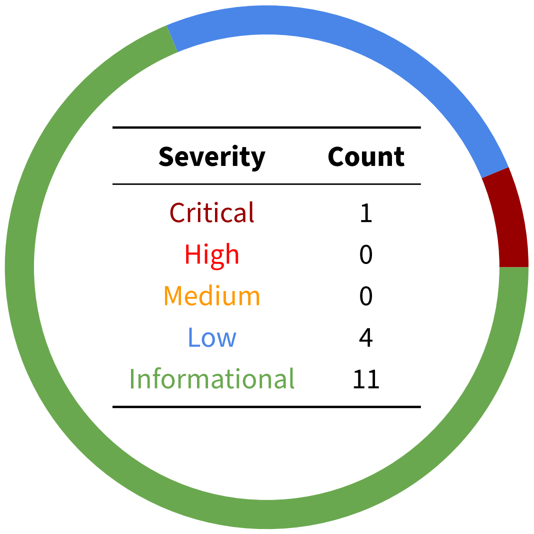
Its key features include:

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

03 | Findings

Overall, we reported 16 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 help mitigate future vulnerabilities.



Severity	Count
Critical	1
High	0
Medium	0
Low	4
Informational	11

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	Low	Resolved	A token may be registered with a reserved value of `token_index`, leading to user token accounts becoming interpreted as deactivated.
OS-MNG-ADV-02	Low	Resolved	[0.17.0] The protocol does not adequately check the `status` on Pyth oracles, potentially leading to the use of stale data.
OS-MNG-ADV-03	Low	Resolved	[0.17.0] `TokenForceCloseBorrowsWithToken`'s current implementation may lead to a scenario where the liquidation candidate `liqee` incurs a borrow on the asset token.
OS-MNG-ADV-04	Low	Resolved	[0.18.0] `TokenConditionalSwaps` do not reserve token positions which can cause triggers to fail.

OS-MNG-ADV-00 [crit] | Inadequate Access Control Inside HealthRegions

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 can 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 perp 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(),
+   ];
```

OS-MNG-ADV-01 [low] | Inadequate Sanitization Of Token Indexes

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 using them will result in an error.

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

This may lead to all user token accounts for this token being interpreted as deactivated, and the attempt 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.

OS-MNG-ADV-02 [low] | [0.17.0] Insufficient Verification Of Pyth Oracle Status

Description

The Pyth oracle suggests validating the `status` on the `PriceAccount` and using 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

To guard against the use of outdated data under unusual market circumstances, ensure that the `status` of the Pyth oracle is `Trading`.

Patch

Fixed in baaec4e.

```diff
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());
+            }
```

OS-MNG-ADV-03 [low] | [0.17.0] Possible Borrow In Token Force Closing

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
+        );
```

OS-MNG-ADV-04 [low] | [0.18.0] TokenConditionalSwaps Do Not Reserve TokenPositions

Description

TokenConditionalSwaps (TCS) were introduced in 0.18.0 as a new type of order which take place 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 `TokenConditionalSwapTrigger`. 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, and therefore the transaction will fail.

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 hard 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 users of the system.

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.

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 lead to security issues in the future.

ID	Description
OS-MNG-SUG-00	Admin-only instructions with incorrect parameters may lead to 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` uses 31 bytes out of 32 bytes.
OS-MNG-SUG-03	Multiple instances of unwrapping may cause panic for users.
OS-MNG-SUG-04	Using named constants instead of magic numbers would improve code clarity.
OS-MNG-SUG-05	Using different oracle prices at different time periods may lead to 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 lead to dangerous outcomes due to its order of operations.
OS-MNG-SUG-09	[0.17.0] Comments in `perp_liq_force_cancel_orders` may lead to 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-00 | Improve Safety Checks On Admin-Only Instructions

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 perp market using the `perp_edit_market` instruction. This parameter must be positive; however, creating the perp market through the `perp_create_market` instruction does not enforce this requirement.

RUST

```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 the `perp_create_market` instruction, ensure that the `init_base_asset_weight` parameter is positive.

Patch

Improved checks will be implemented gradually.

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

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

The issue is acknowledged, and a better solution for removing the peg is under development.

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

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 uses 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_)
  }
```

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

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 cause panic.

Remediation

Use descriptive errors that provide clear information about what caused the error.

Patch

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

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

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.

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

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 used to calculate the value of tokens deposited or withdrawn may not be the same as the one used in the previous operation, which may lead to 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.

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

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

Use 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` and vulnerabilities in the future.

Patch

Fixed in 4dc0e71.

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

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 perp 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.

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

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 lead to incorrect results. For instance, if we refactor the code to update `force_close` before `reduce_only`, the check would break.

Remediation

To mitigate the potential risks, it would be more reliable to 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 applies to `PerpEditMarket` and `Serum3EditMarket` as well.

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

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 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.

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

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, leading to 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.

A | Vulnerability Rating Scale

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

Critical	Vulnerabilities that immediately lead to 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 could lead to 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 could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	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 implementation of the program 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 sum, 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 get a comprehensive understanding of 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 other 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.