Openbook

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Thibault Marboud thibault@osec.io

Robert Chen r@osec.io

Contents

01 | Executive Summary

Overview

Openbook engaged OtterSec to perform an assessment of the `openbook-v2` program. This assessment was conducted between September 7th and September 20th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 5 findings in total.

In particular, we identified a critical vulnerability involving the instruction responsible for placing an order, lacking checks to validate if the market vault matches the side of the order (OS-OBK-ADV-00). Additionally, we highlighted the presence of a comparison error concerning the calculated variance and target variance in the market module (OS-OBK-ADV-01).

We also recommended the possibility of an overflow due to utilizing the u64 type field in the market structure (OS-OBK-SUG-00).

02 | Scope

The source code was delivered to us in a git repository at github.com/openbook-dex/openbook-v2/tree/audit/programs/openbook-v2. This audit was performed against commit 840e661.

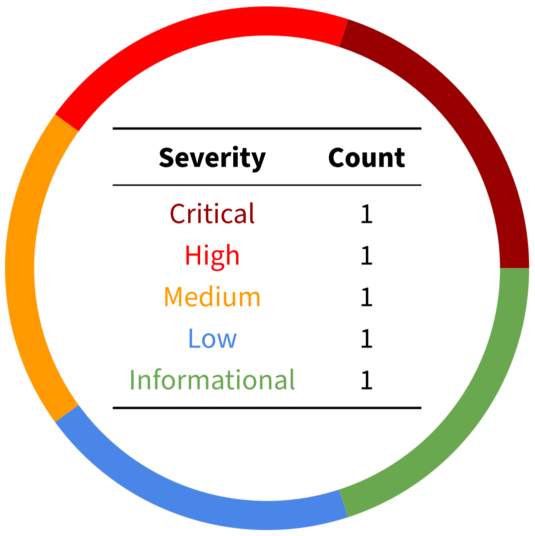
A brief description of the programs is as follows.

Name	Description
openbook-v2	A central limit order book program, built upon the foundation of Mango v4 and the former Openbook program, which had originated as a fork of Serum.

03 | Findings

Overall, we reported 5 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	1
Medium	1
Low	1
Informational	1

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-OBK-ADV-00	Critical	Resolved	`place_order` lacks any checks to validate if `market_vault` matches the side of the order.
OS-OBK-ADV-01	High	Resolved	Comparison error concerning the calculated variance `var` and `target_var` in the `market`.
OS-OBK-ADV-02	Medium	Resolved	Lack of checks to ensure `close_authority` is not set on vault token accounts.
OS-OBK-ADV-03	Low	Resolved	`target_var` in `oracle_price_from_a_and_b` is derived incorrectly.

OS-OBK-ADV-00 [crit] | Missing Side Check On Market Vault Account

Description

The vulnerability is related to a potential mismatch between the side of an order and the market vault utilized for depositing assets in `place_order`. `place_order` is responsible for validating and placing a new order in a trading market, ensuring the calculation of the required deposit amount, and transferring assets between the trader's account and the market's vault.

instructions/place_holder.rs RUST

```rust
pub fn place_order(ctx: Context<PlaceOrder>, order: Order, limit: u8) ->
    ↪  Result<Option<u128>> {
    require_gte!(order.max_base_lots, 0, OpenBookError::InvalidInputLots);
    require_gte!(
        order.max_quote_lots_including_fees,
        0,
        OpenBookError::InvalidInputLots
    );

    [...]

    let mut market = ctx.accounts.market.load_mut()?;
    // abscence of checks regarding side of the order.
    require!(
        !market.is_expired(clock.unix_timestamp),
        OpenBookError::MarketHasExpired
    );
    [...]
}
```

It is crucial to ensure that the deposit and withdrawal of assets into and from a market vault are consistent with the side of the order, i.e., a bid or an ask order. The issue arises since there is no check to ensure that the market vault account utilized for depositing assets (`ctx.accounts.market_vault`) matches the side of the order, as seen in the attached code snippet.

As a result, an attacker may place a bid order but use an ask-side market vault for depositing assets (or vice versa). This mismatch may result in the attacker draining funds from the market vault without providing the expected assets in return. This imbalance in the market vault disrupts the fairness and integrity of the trading system, resulting in financial losses for legitimate traders.

Proof of Concept

1. Trader A places a legitimate bid order to buy one BTC and deposits 10,000 USD into the market vault (as expected for a bid order).
2. Trader B exploits the vulnerability by placing a bid order to buy one BTC but intentionally uses an ask-side market vault (containing BTC) for depositing assets.
3. The attacker's bid order is accepted due to the lack of a check to ensure that the market vault matches the side of the order.
4. The attacker's order executes, and they receive one BTC from Trader A's legitimate order. However, the attacker never deposited the expected USD into the market vault.
5. As a result, Trader A is left with 10,000 USD less in the market vault (as expected), but Trader B never provided the expected USD. This imbalance leaves the market vault with less USD and more BTC.

Remediation

Include checks that ensure the market vault utilized for depositing assets matches the side of the order. If the sides do not match, the order should not be executed, and appropriate error handling should occur.

Patch

Fixed in 1b40b68.

OS-OBK-ADV-01 [high] | Incorrect Check On Variance Value

Description

In `market::oracle_price_from_a_and_b`, the `target_var` value indicates the target variance, which is the upper bound for the permissible range of values for the variance. An issue arises when comparing `target_var` and `var` in `oracle_price_from_a_and_b`.

market.rs RUST

```rust
fn oracle_price_from_a_and_b(
    &self,
    oracle_a_acc: &impl KeyedAccountReader,
    oracle_b_acc: &impl KeyedAccountReader,
    now_slot: u64,
) -> Result<Option<I80F48>> {
    [...]

    let (price, var) = oracle_a.combine_div_with_var(&oracle_b);
    let target_var = self.oracle_config.conf_filter.powi(2);

    if target_var > var {
        msg!(
            "Combined variance too high; value {}, target {}",
            var,
            target_var
        );
        Ok(None)
    } else {
        [...]
    }
}
```

In the existing code, the condition `target_var > var` implies that if the calculated variance (`var`) is less than the target variance (`target_var`), the function returns None. This means that if the actual variance is less than the target variance, the function will not return an oracle price, even though it should. Thus, the only scenario in which it will function without issues is when the variance value is incorrect. In such a situation, users will receive an inaccurate price with a significant error, causing the resulting price to deviate either above or below the actual asset price.

Remediation

Ensure that the calculated variance does not exceed the desired variance by modifying the condition to: `target_var <= var`. This change will allow the function to return an oracle price when the actual variance is less than or equal to the target variance, which is the correct behavior.

Patch

Fixed in dad37aa.

OS-OBK-ADV-02 [med] | Proper Access Control Implementation

Description

The issue relates to the security and access control of market-related accounts, specifically `market_base_vault` and `market_quote_vault`. These vault accounts store and manage tokens associated with the market and are expected to be controlled by certain authorities.

Currently, `create_market` does not explicitly check if the `close_market_admin` is set as an authority on the `market_base_vault` and `market_quote_vault` accounts. If a malicious actor sets the `close_market_admin` as an authority on these vaults, they may manipulate or drain the funds held in those vaults, even if the market should be closed.

Remediation

Implement proper checks and make use of program-derived addresses as detailed below for better security:

- Utilize Program Derived Addresses(PDAs) for vaults instead of creating vault accounts directly within `create_market`, ensuring that the vault accounts are controlled solely by the program and cannot have external authorities.
- When creating the market, the program may initialize the vaults with the appropriate data and permissions. This includes setting the program itself as the authority and specifying other necessary parameters.
- After creating the market and initializing the vaults, the program should explicitly check and ensure that only authorized entities have the necessary authority over these vaults.

Patch

Fixed in 851eca8.

OS-OBK-ADV-03 [low] | Incorrect Variance Calculation

Description

In `market`, `oracle_price_from_a_and_b` combines prices from two oracle accounts (`oracle_a` and `oracle_b`), checks for staleness and variance, and returns the adjusted and converted price if it meets the criteria.

market.rs RUST

```rust
fn oracle_price_from_a_and_b(
    &self,
    oracle_a_acc: &impl KeyedAccountReader,
    oracle_b_acc: &impl KeyedAccountReader,
    now_slot: u64,
) -> Result<Option<I80F48>> {
    [...]

    let (price, var) = oracle_a.combine_div_with_var(&oracle_b);
    let target_var = self.oracle_config.conf_filter.powi(2);

    [...]
}
```

However, the problem lies in how the `target_var` is calculated in a composite oracle utilizing two sources, `oracle_a` and `oracle_b`. `target_var` is calculated as `self.oracle_config.conf_filter.powi(2)`, and is intended to measure the maximum allowable variance.

Thus, while calculating `target_var`, only `config_filter` is considered the uncertainty; it does not consider the `(priceA / priceB)^2` factor, inaccurately calculating `target_var`, which may result in unreasonable asset prices being passed to the user causing financial instability as the assets are traded at incorrect prices due to a faulty oracle. This factor should be included to make the target variance consistent with the single oracle case. Hence, it should be `conf_filter * price` for consistency with the single oracle case.

Remediation

Omit `(priceA / priceB)^2` on both sides of the equation to ensure proper calculation of `target_var`.

Patch

Fixed in 44d7608.

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-OBK-SUG-00	Modification of `Market` structure fields to mitigate any possibility of overflow.

OS-OBK-SUG-00 | Potential Overflow Prevention

Description

The `Market` structure has several fields related to accounting for trading volumes and fees in a market. These fields include:

- `taker_volume_wo_oo`.
- `maker_volume`.
- `fees_accrued`.
- `fees_to_referrers`.

The above fields are intended to keep track of various numeric values related to trading activities. However, they are currently defined as `u64`. While this type is compatible with storing large numbers, in certain situations, especially in highly active markets or over long periods of time, trading volumes and fees may accumulate to a point where `u64` is insufficient to represent these values accurately, resulting in a potential integer overflow and the values becoming incorrect.

market.rs RUST

```rust
#[account(zero_copy)]
#[derive(Debug)]
pub struct Market {
    [...]
    /// Total fees accrued in native quote
    pub fees_accrued: u64,
    // Total fees settled in native quote
    pub fees_to_referrers: u64,
    // Total referrer rebates
    pub referrer_rebates_accrued: u64,

    // Fees generated and available to withdraw via sweep_fees
    pub fees_available: u64,

    /// Cumulative maker volume (same as taker volume) in quote native units
    pub maker_volume: u64,

    /// Cumulative taker volume in quote native units due to place take orders
    pub taker_volume_wo_oo: u64,
    [...]
}
```

Remediation

Utilize the `u128` type, which provides a significantly larger range for these fields, greatly reducing the risk of overflow.

Patch

Fixed in 6808285.

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.