Plasma

Security Assessment

September 9th, 2024 — Prepared by OtterSec

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Ellipsis Labs engaged OtterSec to assess the `plasma` program. This assessment was conducted between August 26th and September 3rd, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 3 findings throughout this audit engagement.

In particular, we identified a vulnerability where the limit order size calculation in the AMM module incorrectly accounts for fees, potentially stopping early instead of excluding fees (OS-EPL-ADV-00).

We also made recommendations to ensure adherence to coding best practices (OS-EPL-SUG-00) and recommended modifying the codebase to mitigate potential overflow issues (OS-EPL-SUG-01).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/Ellipsis-Labs/plasma. This audit was performed against commit 889d61a.

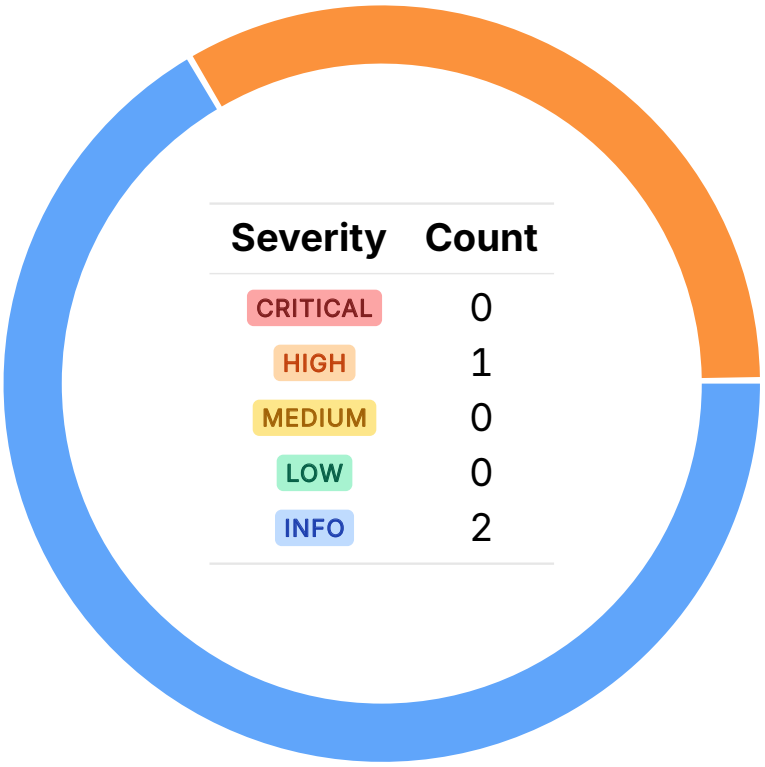
A brief description of the program is as follows:

Name	Description
plasma	The Plasma program provides functionality for creating and managing liquidity pools, allowing users to swap tokens, add or remove liquidity from pools, and manage LP positions.

03 — Findings

Overall, we reported 3 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	0
HIGH	1
MEDIUM	0
LOW	0
INFO	2

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-EPL-ADV-00	HIGH	RESOLVED	The limit order size calculation in the AMM module incorrectly accounts for fees, potentially stopping early instead of excluding fees.

Inconsistencies in the AMM Fee Logic HIGH OS-EPL-ADV-00

Description

`amm::get_limit_order_size_in_base_and_quote` incorrectly applies the fee to the base asset amount rather than the quote asset. The fee is applied after converting the quote asset to the base asset. This results in an incorrect post-fee calculation. The function first converts the quote amount to base via the exchange rate (`base_snapshot / quote_snapshot`), and then applies the fee on the converted base amount. However, fees should only be applied to the quote amount before it is converted to the base asset. This is because the fee should be calculated on the traded amount (in this case, the quote asset) rather than the converted result (base asset).

>_ *crates/plasma_state/src/amm.rs* RUST

```rust
pub fn get_limit_order_size_in_base_and_quote(&self, side: Side) -> LimitOrderConfiguration {
   [...]
    match side {]
        Side::Buy => {
            let ask = if quote_snapshot * base_reserves > base_snapshot * quote_reserves {
                [...]
                let size_in_base = self.post_fee_adjust_rounded_down(
                    size_in_quote * base_snapshot / quote_snapshot,
                );
                let fee_in_quote = self.fee_rounded_down(size_in_quote);
                LimitOrderConfiguration {
                    size_in_base,
                    size_in_quote: size_in_quote - fee_in_quote,
                    fee_in_quote,
                }
            }[..]
        }
        Side::Sell => {
            let bid = if base_snapshot * quote_reserves > quote_snapshot * base_reserves {
                [...]
                let fee_in_quote = self.fee_rounded_down(size_in_quote);
                LimitOrderConfiguration {
                    size_in_base,
                    size_in_quote: size_in_quote - fee_in_quote,
                    fee_in_quote,
                }
            }[...]
        }
    }
}
```

Furthermore, there are inconsistencies in the way the limit order mechanism within the AMM accounts for fees and handles the price snapshot. Currently, the limit order is stopped early to account for fees.

This implies that when a user places a trade, the system prematurely adjusts the amount based on fees before fully consuming the limit order. By prematurely adjusting for fees, the order is not fully executed at the actual limit price. The issue arises because, after the limit order is consumed, the price in the pool may improve due to the natural mechanics of the AMM.

The user will not benefit from this potential price improvement because the limit order stops early due to fee deductions. Additionally, after a swap that fully consumes the limit order, if a new swap is executed in the same direction, the limit order may be hit again due to the pricing mechanism not resetting properly.

Remediation

Apply the fee to the quoted amount first, and then convert the post-fee quote into the base currency. This ensures that the fee is deducted from the traded amount (quote) rather than the converted result (base). Additionally, the fee should only be deducted after the limit order is fully executed. This ensures the order is executed at the correct price and allows traders to benefit from any potential price improvements that may arise.

Patch

Resolved in 46b9aa9.

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-EPL-SUG-00	Suggestions regarding inconsistencies in the codebase and ensuring adherence to coding best practices.
OS-EPL-SUG-01	Recommendations for modifying the codebase to mitigate potential overflow issues.

Code Maturity OS-EPL-SUG-00

1. `process_renounce_liquidity` allows overriding `RenouncedWithBurnedFees`. Once a liquidity position is set to `RenouncedWithBurnedFees`, it is generally expected that the fees will not be recovered. Allowing an override to `RenouncedWithFeeWithdrawal` may be problematic as if the status was initially set to `RenouncedWithBurnedFees`, changing it to `RenouncedWithFeeWithdawal` could potentially allow a provider to recover fees that should have been forfeited.
2. Assert minimum liquidity provider shares while adding liquidity to prevent scenarios where users may manipulate the liquidity pool by reordering swap transactions and adding liquidity to benefit themselves.

Remediation

1. Restrict `process_renounce_liqidity` from overriding once a position is set to `RenouncedWithBurnedFees`.
2. Implement the above-mentioned suggestion.

Code Refactoring OS-EPL-SUG-01

Description

Within the codebase, it would be appropriate to avoid unchecked casts (such as `u64`) and unchecked math and to possibly enable overflow checks. Specifically, the liquidity addition process may be susceptible to overflow issues. Furthermore, switching from `u64` to `u128` for counters, like collected fees, will help mitigate the risk of overflow, as calculations involving large sums of fees may quickly exceed the maximum value that a `u64` type can hold.

Remediation

Implement the above-mentioned recommendations to prevent overflow issues.

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 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.