Euler Fee Flow

Security Assessment

February 21st, 2024 — Prepared by OtterSec

Robert Chen r@osec.io

Woosun Song procfs@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Euler Finance engaged OtterSec to assess the `euler-fee-flow` program. This assessment was conducted between February 19th and February 20th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 1 findings throughout this audit engagement.

We did not identify any vulnerabilities requiring immediate attention. However, we have provided a minor recommendation around gas optimization (OS-EUL-SUG-00).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/euler-xyz/fee-flow. This audit was performed against commit `0b67c0`.

A brief description of the programs is as follows:

Name	Description
euler-fee-flow	Euler Fee Flow enables the efficient conversion of multiple assets into a single asset via decentralized auctions. This process involves setting an initial price and then linearly decreasing it over time until it matches the market value of the accrued assets. Upon successful conversion, the auction rolls over, and the initial price is reset to a multiple of the last conversion price.

03 — Findings

Overall, we reported 1 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.

04 — 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-EUL-SUG-00	Utilize the prefix increment operator to optimize loop iteration.

Gas Optimization OS-EUL-SUG-00

Description

The `for` loop in `buy` may be optimized by utilizing the prefix increment operator. The loop iterates over the `assets` array by incrementing the integer variable `i` in each step. This loop may be optimized to consume less gas by rewriting `i++` to `++i`. This is due to the prefix and postfix operators requiring different numbers of stack operations, specifically, one for prefix and three for postfix.

```solidity
function buy(/* ... */) external nonReentrant returns(uint256 paymentAmount) {
    /* ... */
    for(uint256 i = 0; i < assets.length; i++) {
        // Transfer full balance to buyer
        uint256 balance = ERC20(assets[i]).balanceOf(address(this));
        ERC20(assets[i]).safeTransfer(assetsReceiver, balance);
    }
    return paymentAmount;
}
```

Interestingly, it is not necessary to use an `unchecked` block for incrementing `i` due to a special optimization in `solc-0.8.22`. `solc-0.8.22` recognizes for-loops with an index variable incremented by a step of one and optimizes the incrementation operator to not check for integer overflows. To benefit from this optimization, upgrade to this specific version.

Remediation

1. Rewrite `i++` to `++i`.
2. Update `solc` from `0.8.20` to `0.8.22`.

Patch

Fixed in 03b4452.

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 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.