Grail Token Uniswap Hook Audit

October 2025.

Audited by: Arvolear.

Disclaimer

A smart contract audit does not guarantee absence of bugs, vulnerabilities, and exploits. Subsequent audits and bug bounties are highly recommended.

About Grail Token

The primary intent behind the Grail token Uniswap hook is to collect Grail token trading fees on Uniswap V4. The hook can only be setup with ETH as a primary token (token0), while secondary token (token1) can be arbitrary. The hook takes trading fees in ETH regardless of the direction of the swap, where fees can either be configured static or dynamic by an admin. Dynamic fees calculation takes into account trading volume split into sliding buckets, and current Grail marketcap. The hook explicitely disables the "exactOut" swaps to correctly compute the ETH fees.

The implementation of the hook utilizes three hook handlers: `afterInitialize`, `beforeSwap`, and `afterSwap`.

- `afterInitialize` for calculating the initial Grail marketcap.
- `beforeSwap` for "specified" token fee calculation (in case of ETH -> Grail swap).
- `afterSwap` for "unspecified" token fee calculation (in case of Grail -> ETH swap) and fee distribution.

Minimal and maximal fees the hook can take are hardcoded to 2% and 10% respectively.

Audit Summary

The review indicates that the hook is **safe** to be used in production. No critical or high issues were found.

Severity classification

- Optimization - The issue suggests how to optimize a smart contract.
- Note - The issue requires attention, yet does not pose any risks.
- Low - The issue has minimal impact on the contract's ability to operate.
- Medium - The issue affects the contract's operability and does not lead to a loss of funds.
- High - The issue leads to a loss or incorrect allocation of funds.
- Critical - The issue leads to a straightforward exploit.

Analysis Summary

Review commit hash - c5021c4

Fixes review commit hash - 65ac139

Scope

Smart contracts in scope of the audit:

- `src/hooks/SwapFeeHook.sol`

Number of issues found, categorized by their severity:

- Critical: 0 issues
- High: 0 issues
- Medium: 0 issues
- Low: 1 issues
- Note: 3 issues
- Optimization: 2 issues

Findings

RESOLVED [L-01] "Low" - Misuse of `ONE` leads to wrong EMA and MC calculation if token does not have 18 decimals

Description

In the `_updatePriceEma()` function the `px` variable gets calculated as follows:

```
uint256 px = (ethAbs * ONE) / tokenAbs;
```

Also, the `_afterInitialize` function calculates the initial token price as:

```
uint256 ethPerToken1 = Math.mulDiv(Q192, ONE, sqrt2);
```

If token decimals is not 18, this leads to wrong EMA and MC calculation down the line.

Recommendation

Change `ONE` in both formulas to `_meta[pid].scale`.

Grail comments

> *Fixed in commit `8865791`.*

RESOLVED [N-01] "Note" - `receive` function is not used

Description

The hook declares `receive` function to be able to receive ether. However, this functionality is not used and ether will just get stuck.

Recommendation

Remove `receive` function.

Grail comments

> *Fixed in* `b6ddff5`.

RESOLVED [N-02] "Note" - Inconsistent usage of custom errors

Description

Although the entire hook uses custom errors, `setShortFeeParams` and `setStaticFee` function utilize older require-string style reverts.

Recommendation

Use custom errors throughout the hook.

Grail comments

> *Fixed in* `43f9b2c`.

RESOLVED [N-03] "Note" - Violation of CEI pattern

Description

The `_afterSwap` function violates checks-effects-interactions (CEI) pattern by calling `_distributeEthFees` prior `_addShort` state update. This does not lead any issues or vulnerabilities, yet it is always a good practice to follow this pattern.

Recommendation

Call `_addShort` before `_distributeEthFees`.

Grail comments

> *Fixed in* `b30771f`.

RESOLVED [O-01] "Optimization" - Excessive `onlyPoolManager` usage in internal hooks

Description

There is no need to specify `onlyPoolManager` modified in internal hooks functions since it is already declared in the external ones.

Recommendation

Remove excessive `onlyPoolManager` modifier.

Grail comments

> *Fixed in* `8784ea3`.

RESOLVED [O-02] "Optimization" - Suboptimized storage usage and packing

Description

1. In `ShortWindow` struct, `bucketStart` variable is declared as `uint64` type. This is suboptimal because everytime is it read, Solidity has to clean up and mask higher bits to zeros. Declaring it as `uint256` will optimize gas usage a tiny bit.
2. The variables in `PoolMeta` struct can be reordered to achieve tight storage packing. Alternatively, the `set` variable can be completely removed and `_afterInitialize` logic rewritten to perform pool initialization checks against `decimals` not being zero.

Recommendation

1. Change `bucketStart` variable type to `uint256`.
2. Remove `set` variable and utilize `decimals` for pool initialization checks.

Grail comments

> *Fixed in* `108beb6`.