Hyperbloom Security Review

Pashov Audit Group

Conducted by: Ch_301, Said, Klaus

June 24th 2025 - June 25th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **de-swarm/contracts.hyperbloom.xyz** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Hyperbloom

Hyperbloom is a Yield Optimizer that helps users earn compound interest on their tokens by automatically maximizing rewards from liquidity pools, AMMs, and yield farming opportunities. Through its Vaults, Hyperbloom compounds farm rewards back into your deposited assets, allowing you to withdraw your funds at any time. Hyperbloom is a fork of Beefy Finance.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 7dba168e001e46188b0d83c2783daf741c7e1d2a

fixes review commit hash - ab2c86305e6baecb6c9d6f4d0883b773fd91d759

Scope

The following smart contracts were in scope of the audit:

- `contracts/BIFI/infra/`
- `contracts/BIFI/interfaces/`
- `contracts/BIFI/strategies/`
- `contracts/BIFI/utils/`
- `contracts/BIFI/vaults/`
- `contracts/BIFI/zaps/`

7. Executive Summary

Over the course of the security review, Ch_301, Said, Klaus engaged with Hyperbloom to review Hyperbloom. In this period of time a total of **4** issues were uncovered.

Protocol Summary

Protocol Name	Hyperbloom
Repository	https://github.com/de-swarm/contracts.hyperbloom.xyz
Date	June 24th 2025 - June 25th 2025
Protocol Type	Yield Optimizer

Findings Count

Severity	**Amount**
Medium	1
Low	3
Total Findings	**4**

Summary of Findings

ID	Title	Severity	Status
[M-01]	withdraw operation is prone to sandwich attacks	Medium	Resolved
[L-01]	Sending without checking if the amount is zero	Low	Resolved
[L-02]	Lack of configurable time interval for shortTwap	Low	Resolved
[L-03]	Division by zero in TWAP calculation	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] `withdraw` operation is prone to sandwich attacks

Severity

Impact: Medium

Likelihood: Medium

Description

Inside `withdraw` operation of `StrategyPassiveManagerHyperswap` and `StrategyPassiveManagerKittenswap`, `_onlyCalmPeriods` will be checked only when `calmAction` is false and `lastDeposit` is equal to `block.timestamp`.

```
function withdraw(uint256 _amount0, uint256 _amount1) external {
        _onlyVault();

>>>     if (block.timestamp == lastDeposit && !calmAction) _onlyCalmPeriods();

        if (_amount0 > 0) IERC20Metadata(lpToken0).safeTransfer
          (vault, _amount0);
        if (_amount1 > 0) IERC20Metadata(lpToken1).safeTransfer
          (vault, _amount1);

        if (!_isPaused()) _addLiquidity();

        (uint256 bal0, uint256 bal1) = balances();
        calmAction = false;

        emit TVL(bal0, bal1);
    }
```

An attacker can exploit this by depositing dust amount, waiting for the next block, then performing a swap to manipulate the price. After that, they can call `withdraw` to trigger `_removeLiquidity` and `_addLiquidity`, causing the strategy to add liquidity using the manipulated price.

```
function _addLiquidity() private {
        _whenStrategyNotPaused();

        (uint256 bal0, uint256 bal1) = balancesOfThis();

        uint160 sqrtprice = sqrtPrice();
        uint128 liquidity = LiquidityAmounts.getLiquidityForAmounts(
            sqrtprice,
            TickMath.getSqrtRatioAtTick(positionMain.tickLower),
            TickMath.getSqrtRatioAtTick(positionMain.tickUpper),
            bal0,
            bal1
        );

        bool amountsOk = _checkAmounts
          (liquidity, positionMain.tickLower, positionMain.tickUpper);

>>>     if (liquidity > 0 && amountsOk) {
            minting = true;
            IKittenswapV3Pool(pool).mint(address(
              pool
            ).mint(address
              (this
        } else {
            if(!calmAction) _onlyCalmPeriods();
        }

        (bal0, bal1) = balancesOfThis();

        liquidity = LiquidityAmounts.getLiquidityForAmounts(
            sqrtprice,
            TickMath.getSqrtRatioAtTick(positionAlt.tickLower),
            TickMath.getSqrtRatioAtTick(positionAlt.tickUpper),
            bal0,
            bal1
        );

        // Flip minting to true and call the pool to mint the liquidity.
        if (liquidity > 0) {
            minting = true;
            IKittenswapV3Pool(pool).mint(address(
              pool
            ).mint(address
              (this
        }
    }
```

This can happen because, inside `_addLiquidity`, the `_onlyCalmPeriods` check is skipped as long as the price remains within the `positionMain` ticks.

Recommendations

Add `onlyCalmPeriods` modifier to `_addLiquidity`.

8.2. Low Findings

[L-01] Sending without checking if the amount is zero

In `StrategyPassiveManagerKittenswap._chargeFees`, we call `safeTransfer` without checking that the values of `callFeeAmount`, `strategistFeeAmount`, and `beefyFeeAmount` are zero. Since we are sending a native token(WHYPE), sending zero will not cause the transaction to fail, but it will waste gas unnecessarily.

[L-02] Lack of configurable time interval for `shortTwap`

Currently, the time interval for `shortTwap` is hardcoded to 3 seconds.

```
function shortTwap() public view returns (int56 twapTick) {
        uint32[] memory secondsAgo = new uint32[](2);
        secondsAgo[0] = uint32(3);
        secondsAgo[1] = 0;

        (int56[] memory tickCuml,) = IUniswapV3Pool(pool).observe(secondsAgo);
        twapTick = (tickCuml[1] - tickCuml[0]) / int32(3);
    }
```

If `twapInterval` for `twap` is set to less than 3, `shortTwap` will not function as intended. Consider allowing the twap interval for `shortTwap` to be configurable.

[L-03] Division by zero in TWAP calculation

The `StrategyPassiveManagerHyperswap` contract contains a potential division by zero vulnerability in the `twap()` function. The owner can set `twapInterval` to zero using `setTwapInterval()`, which would cause the TWAP calculation to revert.

```
function setTwapInterval(uint32 _interval) external onlyOwner {
    emit SetTwapInterval(twapInterval, _interval);
    twapInterval = _interval; // No validation - can be set to 0
}

function twap() public view returns (int56 twapTick) {
    uint32[] memory secondsAgo = new uint32[](2);
    secondsAgo[0] = uint32(twapInterval);
    secondsAgo[1] = 0;
    (int56[] memory tickCuml,) = IUniswapV3Pool(pool).observe(secondsAgo);
    twapTick = (tickCuml[1] - tickCuml[0]) / int32
    //(twapInterval); // Division by zero if twapInterval = 0
}
```

If `twapInterval` is set to zero, the following critical functions will revert due to the division by zero in `twap()`

deposit(), withdraw(),harvest(),moveTicks(),setPositionWidth(),unpause() This would effectively break the core functionality of the strategy until `twapInterval` is set to a non-zero value.

This can be resolved by adding validation in `setTwapInterval()` to prevent setting the interval to zero.