Wagmi Security Review

Pashov Audit Group

Conducted by: windhustler, pontifex, SpicyMeatball, carrotsmuggler

April 1st 2024 - April 18th 2024

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 **wagmi-leverage** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Wagmi

Wagmi Leverage is a leverage product, built on concentrated liquidity without a price-based liquidation or price oracles. This system caters to liquidity providers and traders (borrowers). The trader pays for the time to hold the position as long as interest is paid.

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 - 15ef9740b196b146dae0a48d75811512788040a1

fixes review commit hash - 48cbe0b73ef952ed7c6a12817cffc61c0535f3b5

Scope

The following smart contracts were in scope of the audit:

- `LightQuoterV3`
- `LiquidityBorrowingManager`
- `Vault`
- `FlashLoanAggregator`
- `TransferHelper`
- `Constants`
- `ErrLib`
- `ExternalCall`
- `AmountsLiquidity`
- `Keys`
- `ApproveSwapAndPay`
- `LiquidityManager`
- `OwnerSettings`
- `DailyRateAndCollateral`

7. Executive Summary

Over the course of the security review, windhustler, pontifex, SpicyMeatball, carrotsmuggler engaged with Wagmi to review Wagmi. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Wagmi
Repository	https://github.com/RealWagmi/wagmi-leverage
Date	April 1st 2024 - April 18th 2024
Protocol Type	Leverage protocol

Findings Count

Severity	**Amount**
Medium	1
Low	2
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Protocol charges platform fees twice	Medium	Resolved
[L-01]	Users might overpay while borrowing	Low	Acknowledged
[L-02]	Missing check in the _excuteCallback function	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Protocol charges platform fees twice

Severity

Impact: High

Likelihood: Low

Description

LP can call the repay on an underwater loan to retrieve their tokens back without their position being restored. If there are no other positions associated with this loan, the caller will receive the collateral minus platform fees and a liquidation bonus. However, there is an issue where platform fees are being charged twice.

```
if (params.isEmergency) {
            (!underLiquidation).revertError(ErrLib.ErrorCode.FORBIDDEN);
            (
                uint256 removedAmt,
                uint256 feesAmt,
                bool completeRepayment
            ) = _calculateEmergencyLoanClosure(
                    zeroForSaleToken,
                    params.borrowingKey,
                    currentFees,
                    borrowing.borrowedAmount
                );
            (removedAmt == 0).revertError(ErrLib.ErrorCode.LIQUIDITY_IS_ZERO);
            // Subtract the removed amount and fees from borrowedAmount and
            // feesOwed
            borrowing.borrowedAmount -= removedAmt;
            borrowing.dailyRateCollateralBalance -= feesAmt;
>>          feesAmt =
                _pickUpPlatformFees(borrowing.holdToken, feesAmt) /
                Constants.COLLATERAL_BALANCE_PRECISION;
            // Deduct the removed amount from totalBorrowed
            unchecked {
                holdTokenRateInfo.totalBorrowed -= removedAmt;
            }
            // If loansInfoLength is 0, remove the borrowing key from storage
            // and get the liquidation bonus
            if (completeRepayment) {
                LoanInfo[] memory empty;
                _removeKeysAndClearStorage
                  (borrowing.borrower, params.borrowingKey, empty);
>>              feesAmt =
                  _pickUpPlatformFees(borrowing.holdToken, currentFees) /
                    Constants.COLLATERAL_BALANCE_PRECISION +
                    liquidationBonus;
            } else {
```

This will break protocol accounting since the recorded sum of tokens will be greater than the actual amount.

Here is the coded POC in `LiquidityBorrowingManager.t.sol`:

```
function testDoublePlatformFee() public {
        uint128 minLiqAmt = _minimumLiquidityAmt(253_320, 264_600);
        address[] memory tokens = new address[](1);
        tokens[0] = address(WETH);
        address vault = borrowingManager.VAULT_ADDRESS();

        vm.startPrank(bob);
        borrowingManager.borrow(createBorrowParams
          (tokenId, minLiqAmt), block.timestamp + 1);
        bytes32[] memory key = borrowingManager.getBorrowingKeysForTokenId
          (tokenId);
        vm.stopPrank();

        ILiquidityBorrowingManager.FlashLoanRoutes memory routes;
        ILiquidityBorrowingManager.SwapParams[] memory swapParams;


            isEmergency: true,
            routes: routes,
            externalSwap: swapParams,
            borrowingKey: key[0],
            minHoldTokenOut: 0,
            minSaleTokenOut: 0
        });

        // time to repay underwater loan
        vm.warp(block.timestamp + 86401);
        vm.prank(alice);
        (uint saleOut, uint holdToken) = borrowingManager.repay
          (repay, block.timestamp + 1);

        borrowingManager.collectProtocol(address(this), tokens);

        vm.expectRevert(bytes("W-ST"));
        vm.prank(alice);
        borrowingManager.collectLoansFees(tokens);
    }
```

In this scenario LP is unable to collect the rewards after platform fees were collected.

Recommendations

```
if (completeRepayment) {
                LoanInfo[] memory empty;
                _removeKeysAndClearStorage
                  (borrowing.borrower, params.borrowingKey, empty);
+               feesAmt +=
-                   _pickUpPlatformFees(borrowing.holdToken, currentFees) /
-                   Constants.COLLATERAL_BALANCE_PRECISION +
                    liquidationBonus;
            } else {
```

8.2. Low Findings

[L-01] Users might overpay while borrowing

The final cost of borrowing is determined by taking a sum of `marginDeposit`, `liquidationBonus`, `dailyRateCollateral` and `holdTokenEntranceFee`.

```
## LiquidityBorrowingManager.sol

uint256 amountToPay;
unchecked {
    // Updating borrowing details
    borrowing.borrowedAmount += cache.borrowedAmount;
    borrowing.liquidationBonus += liquidationBonus;
    // Transfer the required tokens to the VAULT_ADDRESS for collateral and
    // holdTokenBalance
    borrowing.dailyRateCollateralBalance +=
        cache.dailyRateCollateral *
        Constants.COLLATERAL_BALANCE_PRECISION;
    amountToPay =
        marginDeposit +
        liquidationBonus +
        cache.dailyRateCollateral +
        cache.holdTokenEntranceFee;
}
```

Users can pass on `maxDailyRate` to check if the `dailyRateCollateral` has changed in between them submitting the transaction and the transaction being executed.

There is no option to check if `liquidationBonus` and `entranceFeeBps` have changed. The maximum value for `entranceFee` and `liquidationBonus` is 10%.

Given the unfortunate circumstances of a borrower taking a big loan and governance changing these parameters, the borrower might pay much more than intended.

Consider allowing the caller of the borrow function to revert in case these two parameters surpass a certain value.

[L-02] Missing check in the `_excuteCallback` function

The `FlashLoanAggregator.flashLoan` function contains the `address(POOL) != address(0)` check. But in case the `flashLoanParams[0].protocol` is not `Protocol.AAVE` the check will be skipped. As a result, the `FlashLoanAggregator._excuteCallback` function can throw an incorrect error. This can cause difficulties with the solving of the error reason.