AstraSec

Wagmi Leverage

Security Audit Report

March 26, 2024

Contents

1 Introduction

1.1 About Wagmi Leverage

`Wagmi Leverage` is a leverage product which is built on concentrated liquidity without a price-based liquidation or price oracles. The system caters to liquidity providers and traders (borrowers). The trader pays for the time to hold the position as long as he wants and as long as interest is paid. `Wagmi` enhances yields for `Uniswap V3` liquidity providers by offsetting impermanent loss. LPs can earn yield even when their liquidity position is out of range. When not utilized for trading, their liquidity position is lent to traders, earning them higher yields through premiums and trading fees. Traders on `Wagmi` can margin long or short any pair without the risk of forced price-based liquidations. Even if their position is underwater, they are only required to pay premiums to LPs to maintain their position. This model gives traders access to high leverage on every asset and eliminates the concern of forced liquidations.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/RealWagmi/wagmi-leverage
- CommitID: b707bd1 (Wagmi Leverage V1)
- CommitID: a7ef3f2 (Wagmi Leverage V2)

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/RealWagmi/wagmi-leverage
- CommitID: 7575ab6

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Wagmi Leverage` protocol. Throughout this audit, we identified a total of 7 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	1	-	-	1
Medium	1	1	-	-
Low	3	-	-	3
Informational	2	1	-	1
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~H-1~~ Improper Borrowing Fee Distribution in harvest()

M-1 Potential Risks Associated with Centralization

~~L-1~~ Revisited Liquidation Bonus Calculation in borrow()

~~L-2~~ Improved Entrance Fee Calculation Logic in _precalculateBorrowing()

~~L-3~~ Timely accLoanRatePerSeconds Update during Emergency Repayment

~~I-1~~ Improved Token Swap Logic in _restoreLiquidity()

I-2 Meaningful Events for Key Operations

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Improper Borrowing Fee Distribution in harvest()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityBorrowingManager.sol	Business Logic	High	High	Addressed

In the `LiquidityBorrowingManager` contract, the `harvest()` function allows lenders to harvest the fees accumulated from their loans. While examining its logic,we notice the current fee distribution logic is not correct.

To elaborate, we show below this `harvest()` routine. This routine implements a rather straightforward logic in computing the accumulated fees for each lender: it is calculated based on the proportion of the `holdTokenDebt` to the `borrowedAmount`. However, we note that the current implementation does not take the loan duration into consideration. This means that under the condition of holding the same `holdTokenDebt` , both those who lend for a longer period and those who lend for a shorter period will benefit equally.

LiquidityBorrowingManager::harvest()

```
function harvest(bytes32 borrowingKey) external nonReentrant returns (uint256
    harvestedAmt) {
    ...
    LoanInfo[] memory loans = loansInfo[borrowingKey];
    // Iterate through each loan in the loans array.
    for (uint256 i; i < loans.length; ) {
        LoanInfo memory loan = loans[i];
        // Get the owner address of the loan's token ID using the
            underlyingPositionManager contract.
        address creditor = _getOwnerOf(loan.tokenId);
        // Check if the owner of the loan's token ID is equal to the `msg.sender
            `.
        if (creditor != address(0)) {
            // Update the liquidity cache based on the loan information.
            _upNftPositionCache(zeroForSaleToken, loan, cache);
            uint256 feesAmt = FullMath.mulDiv(feesOwed, cache.holdTokenDebt,
                borrowedAmount);
            // Calculate the fees amount based on the total fees owed and
                holdTokenDebt.
            loansFeesInfo[creditor][cache.holdToken] += feesAmt;
            harvestedAmt += feesAmt;
        }
        unchecked {
            ++i;
        }
    }
```

```
    borrowing.feesOwed -= harvestedAmt;

    emit Harvest(borrowingKey, harvestedAmt);
}
```

Note a number of functions share the similar issue, including `LiquidityBorrowingManager::repay()` and `LiquidityManager::_restoreLiquidity()`.

Remediation When distributing the accumulated fees, the lending duration of the lenders should be considered.

[M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Medium	Acknowledged

In the `Wagmi Leverage` protocol, the existence of a privileged `owner` account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative functions potentially affected by the privileges associated with the privileged account.

Example Privileged Operations in `Wagmi Leverage`

```
function updateSettings(ITEM _item, uint256[] calldata values) external
    onlyOwner {
    if (_item == ITEM.LIQUIDATION_BONUS_FOR_TOKEN) {
        require(values.length == 3);
        if (values[1] > Constants.MAX_LIQUIDATION_BONUS) {
            revert InvalidSettingsValue(values[1]);
        }
        if (values[2] == 0) {
            revert InvalidSettingsValue(0);
        }
        liquidationBonusForToken[address(uint160(values[0]))] = Liquidation(
            values[1],
            values[2]
        );
    } else {
        require(values.length == 1);
        if (_item == ITEM.PLATFORM_FEES_BP) {
            if (values[0] > Constants.MAX_PLATFORM_FEE) {
                revert InvalidSettingsValue(values[0]);
            }
            platformFeesBP = values[0];
```

```
        } else if (_item == ITEM.DEFAULT_LIQUIDATION_BONUS) {
            if (values[0] > Constants.MAX_LIQUIDATION_BONUS) {
                revert InvalidSettingsValue(values[0]);
            }
            dafaultLiquidationBonusBP = values[0];
        } else if (_item == ITEM.OPERATOR) {
            operator = address(uint160(values[0]));
        }
    }
    emit UpdateSettingsByOwner(_item, values);
}

function updateHoldTokenDailyRate(
    address saleToken,
    address holdToken,
    uint256 value
) external onlyOperator {
    ...
    holdTokenRateInfo.currentDailyRate = value;
    emit UpdateHoldTokenDailyRate(saleToken, holdToken, value);
}

function updateHoldTokenEntranceFee(
    address saleToken,
    address holdToken,
    uint256 value
) external onlyOperator {
    ...
    holdTokenEntranceFeeInfo.entranceFeeBP = value;
    emit UpdateHoldTokeEntranceFee(saleToken, holdToken, value);
}
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

[L-1] Revisited Liquidation Bonus Calculation in borrow()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityBorrowingManager.sol	Business Logic	Low	Low	Addressed

When a trader opens a long position by borrowing the liquidity of `Uniswap V3`, the trader needs to provide additional assets to ensure that the removed liquidity can be restored again. These additional paid assets including margin deposit, liquidation bonus, daily collateral and entrance fee. Among them, the liquidation bonus is an additional amount added to the debt as a bonus in case of liquidation. While examining the current borrow logic, we notice the liquidation bonus calculation needs to be revisited. Specifically, the calculation for liquidation bonus is based on hold token, borrowed amount, and the number of used loans. It means that traders will need to pay less liquidation bonus if they only borrow one position each time.

LiquidityBorrowingManager::borrow()

```
  function borrow(
    BorrowParams calldata params,
    uint256 deadline
)
    external
    nonReentrant
    checkDeadline(deadline)
    returns (uint256, uint256, uint256, uint256, uint256)
{
    ...
    uint256 liquidationBonus;
    {
        // Adding borrowing key and loans information to storage
        uint256 pushCounter = _addKeysAndLoansInfo(borrowingKey, params.loans);
        // Calculating liquidation bonus based on hold token, borrowed amount,
            and number of used loans
        liquidationBonus = getLiquidationBonus(
            params.holdToken,
            cache.borrowedAmount,
            pushCounter
        );
    }
    if (cache.holdTokenBalance > cache.borrowedAmount) {
        cache.borrowedAmount = cache.holdTokenBalance;
    }

    ...
}
```

Remediation The calculation of liquidity bonus should not be related to the number of positions used in a single borrow transaction.

[L-2] Improved Entrance Fee Calculation Logic in _precalculateBorrowing()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityBorrowingManager.sol	Business Logic	Low	Low	Addressed

As mentioned previously, when a trader opens a long position by borrowing the liquidity of `Uniswap V3`, this trader needs to pay entrance fee. The hold token entrance fee is calculated based on the hold token balance and entrance fee basis points (lines 925 – 929). However, the hold token balance can be manipulated by transferring the swapped sale token to the trader instead of address(this) (line 904). Therefore, the entrance fee that needs to be paid by the trader can also be manipulated.

LiquidityBorrowingManager::_precalculateBorrowing()

```
896 function _precalculateBorrowing(
897     BorrowParams calldata params
898 ) private returns (BorrowCache memory cache) {
899     {
900     ...

902     if (params.externalSwap.length != 0) {
903         // Call the external swap function
904         _callExternalSwap(params.saleToken, params.externalSwap);
905     }
906     uint256 saleTokenBalance;
907     // Get the balance of the sale token and hold token in the pair
908     (saleTokenBalance, cache.holdTokenBalance) = _getPairBalance(
909         params.saleToken,
910         params.holdToken
911     );
912     // Check if the sale token balance is greater than 0
913     if (saleTokenBalance > 0) {
914         // Call the internal v3SwapExactInput function and update the hold token
                balance in the cache
915         cache.holdTokenBalance += _v3SwapExactInput(
916             v3SwapExactInputParams({
917                 fee: params.internalSwapPoolfee,
918                 tokenIn: params.saleToken,
919                 tokenOut: params.holdToken,
920                 amountIn: saleTokenBalance
921             })
922         );
923     }
924     // Calculate the hold token entrance fee based on the hold token balance and
            entrance fee basis points
925     cache.holdTokenEntraceFee =
926         (cache.holdTokenBalance *
927             cache.holdTokenEntraceFee *
928             Constants.COLLATERAL_BALANCE_PRECISION) /
```

```
        Constants.BP;
    ...
}
```

Remediation Use `holdTokenDebt` to calculate the entrance fee instead of hold token balance.

[L-3] Timely accLoanRatePerSeconds Update during Emergency Repayment

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV3SwapExactAmountOut.sol	Business Logic	Low	Low	Addressed

In `Wagmi Leverage` protocol, the liquidity provider whose liquidity is present in the trader's position can use the emergency mode to withdraw their liquidity. In this case, he/she will receive hold tokens and the liquidity will not be restored in the `Uniswap V3` pool. While examining the emergency repayment logic, we notice the state variable `borrowingStorage.accLoanRatePerSeconds` is not timely updated if all loans have not been removed. This may cause losses to the traders if they want to add more daily collateral later.

UniswapV3SwapExactAmountOut::swapExactAmountOutOnUniswapV3()

```
function repay(
    RepayParams calldata params,
    uint256 deadline
)
    external
    nonReentrant
    checkDeadline(deadline)
    returns (uint256 saleTokenOut, uint256 holdTokenOut)
{
    ...
    // Check if it's an emergency repayment
    if (params.isEmergency) {
        ...
        if (completeRepayment) {
            LoanInfo[] memory empty;
            _removeKeysAndClearStorage(borrowing.borrower, params.borrowingKey,
                empty);
            feesAmt += liquidationBonus;
        } else {
            // make changes to the storage
            BorrowingInfo storage borrowingStorage = borrowingsInfo[params.
                borrowingKey];
            borrowingStorage.dailyRateCollateralBalance = 0;
            borrowingStorage.feesOwed = borrowing.feesOwed;
```

```
            borrowingStorage.borrowedAmount = borrowing.borrowedAmount;
        }
        holdTokenOut = removedAmt + feesAmt;
        // Transfer removedAmt + feesAmt to msg.sender and emit
            EmergencyLoanClosure event
        Vault(VAULT_ADDRESS).transferToken(borrowing.holdToken, msg.sender,
            holdTokenOut);
        emit EmergencyLoanClosure(borrowing.borrower, msg.sender, params.
            borrowingKey);
    } else {
        ...
    }
}
```

Remediation Timely update the `borrowingStorage.accLoanRatePerSeconds` during the emergency repayment.

[I-1] Improved Token Swap Logic in _restoreLiquidity()

Target	Category	IMPACT	LIKELIHOOD	STATUS
LiquidityManager.sol	Business Logic	N/A	N/A	Addressed

When a position is closed either by the trader or by the liquidator if the trader has not paid for holding the position on time, the liquidity borrowed from `Uniswap V3` must be restored from the hold token. To restore the borrowed liquidity, the required sales token is obtained by exchanging the hold token through `Uniswap V3`. When examining the related logic, we notice the token swap logic can be improved.

To elaborate, we show below the related code snippet. Specifically, we should perform exact output `V3` swap instead of exact input `V3` swap if `params.swapPoolfeeTier != cache.fee`. Otherwise, we may need to perform another swap and exchange the excess sale tokens for hold tokens if the function caller only want to get hold tokens, which is a waste of gas.

LiquidityManager::_restoreLiquidity()

```
function _restoreLiquidity(
    RestoreLiquidityParams memory params,
    LoanInfo[] memory loans
) internal {
    NftPositionCache memory cache;

    for (uint256 i; i < loans.length; ) {
        ...
        if (creditor != address(0)) {
```

```
            ...
            if (holdTokenAmountIn > 0) {
                //  The internal swap in the same pool in which liquidity is
                    restored.
                if (params.swapPoolfeeTier == cache.fee) {
                    (sqrtPriceX96, holdTokenAmountIn, amounts) =
                        _calculateAmountsToSwap(
                        !params.zeroForSaleToken,
                        sqrtPriceX96,
                        loan.liquidity,
                        cache,
                        saleTokenBalance
                    );
                }

                // Perform v3 swap exact input and update sqrtPriceX96
                _v3SwapExactInput(
                    v3SwapExactInputParams({
                        fee: params.swapPoolfeeTier,
                        tokenIn: cache.holdToken,
                        tokenOut: cache.saleToken,
                        amountIn: holdTokenAmountIn
                    })
                );
            }

            // Increase liquidity and transfer liquidity owner reward
            _increaseLiquidity(
                cache.saleToken,
                cache.holdToken,
                loan,
                amounts.amount0,
                amounts.amount1
            );
            ...
        }

        unchecked {
            ++i;
        }
    }
}
```

Remediation Perform exact output `V3` swap instead of exact input `V3` swap if `params.swapPoolfeeTier != cache.fee`.

[I-2] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	N/A	N/A	Acknowledged

The `event` feature is vital for capturing runtime dynamics in a contract. Upon emission, `events` store transaction arguments in logs, supplying external analytics and reporting tools with crucial information. They play a pivotal role in scenarios like modifying system-wide parameters or handling token operations.

However, in our examination of protocol dynamics, we observed that certain privileged routines lack meaningful events to document their changes. We highlight the representative routines below.

LiquidityBorrowingManager.sol

```
function setSwapCallToWhitelist(
    address swapTarget,
    bytes4 funcSelector,
    bool isAllowed
) external onlyOwner {
    (swapTarget == VAULT_ADDRESS
        swapTarget == address(this)
        swapTarget == address(underlyingPositionManager)
        funcSelector == IERC20.transferFrom.selector).revertError(ErrLib.
            ErrorCode.FORBIDDEN);
    whitelistedCall[swapTarget][funcSelector] = isAllowed;
}
```

FlashLoanAggregator.sol

```
function setWagmiLeverageAddress(address _wagmiLeverageAddress) external
    onlyOwner {
    wagmiLeverageContracts[_wagmiLeverageAddress] = true;
}
```

Remediation Ensure the proper emission of meaningful events containing accurate information to promptly reflect state changes.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Name	AstraSec Team
Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI