SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: RealWagmi
Prepared by: Sherlock
Lead Security Expert: bughuntoor
Dates Audited: March 18 - March 23, 2024
Prepared on: April 2, 2024

Introduction

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 he wants as long as interest is paid.

Scope

Repository: RealWagmi/wagmi-leverage

Branch: main

Commit: 09a2afefc33a08452287a08bc1ebf34665c55165

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
2	0

Issues not fixed or acknowledged

Medium	High
0	0

Issue M-1: Liquidation bonus scales exponentially instead of linearly.

Source: https://github.com/sherlock-audit/2024-03-wagmileverage-v2-judging/issues/7

Found by

bughuntoor

Summary

Liquidation bonus scales exponentially instead of linearly.

Vulnerability Detail

Let's look at the code of `getLiquidationBonus`

```
function getLiquidationBonus(
    address token,
    uint256 borrowedAmount,
    uint256 times
) public view returns (uint256 liquidationBonus) {
    // Retrieve liquidation bonus for the given token
    Liquidation memory liq = liquidationBonusForToken[token];
    unchecked {
        if (liq.bonusBP == 0) {
            // If there is no specific bonus for the token
            // Use default bonus
            liq.minBonusAmount = Constants.MINIMUM_AMOUNT;
            liq.bonusBP = dafaultLiquidationBonusBP;
        }
        liquidationBonus = (borrowedAmount * liq.bonusBP) / Constants.BP;

        if (liquidationBonus < liq.minBonusAmount) {
            liquidationBonus = liq.minBonusAmount;
        }
        liquidationBonus *= (times > 0 ? times : 1);
    }
}
```

As we can see, the liquidation bonus is based on the entire `borrowAmount` and multiplied by the number of new loans added. The problem is that it is unfair when the user makes a borrow against multiple lenders.

If a user takes a borrow for X against 1 lender, they'll have to pay a liquidation bonus of Y. However, if they take a borrow for 3X against 3 lenders, they'll have to pay 9Y, meaning that taking a borrow against N lenders leads to overpaying liquidation bonus by N times.

Furthermore, if the user simply does it in multiple transactions, they can avoid these extra fees (as they can simply call `borrow` for X 3 times and pay 3Y in Liquidation bonuses)

Impact

Loss of funds

Code Snippet

https://github.com/sherlock-audit/2024-03-wagmileverage-v2/blob/main/wagmi-leverage/contracts/LiquidityBorrowingManager.sol#L280

Tool used

Manual Review

Recommendation

make liquidation bonus simply a % of totalBorrowed

Discussion

fann95

We discussed as a team multiplying the bonus depending on the method of taking out a loan and ultimately decided to abandon it completely. a few days ago I made the corresponding commit https://github.com/RealWagmi/wagmi-leverage/commit/7575ab6659e99e59f5b7b7d1454649091c0295c6

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/RealWagmi/wagmi-leverage/commit/7575ab6659e99e59f5b7b7d1454649091c0295c6.

sherlock-admin3

2 comment(s) were left on this issue during the judging contest.

WangAudit commented:

> decided that it's a H not an M cause even if it's intended behaviour; user can easily bypass it; or if it's not intended' then it will be applied every time when times > 2

takarez commented:

> i think that is a design choice to charge whenever there's a borrow.

fann95

> 2 comment(s) were left on this issue during the judging contest.
>
> **WangAudit** commented:
>
> > decided that it's a H not an M cause even if it's intended behaviour; user can easily bypass it; or if it's not intended' then it will be applied every time when times > 2
>
> **takarez** commented:
>
> > i think that is a design choice to charge whenever there's a borrow.

The liquidation bonus was charged more than initially expected, so the user's cunning could only lead to a fair payment. But ultimately the bonus would still be returned to the trader. The trader paid a more liquidation bonus if he extracted more than one NFT position, which in itself is a rare case... anyway, we removed it from the code..

spacegliderrrr

Fix looks good, liquidation bonus is now correctly proportional to borrow amount, no matter the number of lenders.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-2: When the amout of token acquired by a flash loan exceeds the expected value, the callback function will fail.

Source: https://github.com/sherlock-audit/2024-03-wagmileverage-v2-judging/issues/9

Found by

zraxx

Summary

When the amout of token acquired by a flash loan exceeds the expected value, the callback function will fail.

Vulnerability Detail

The function `wagmiLeverageFlashCallback` is used to handle the repayment operation after flash loan. After obtaining enough saleToken, it uses `_v3SwapExact` to convert the saleToken into holdToken. We know that the amount of holdTokens (`holdTokenAmtIn`) is proportional to the amount of saleTokens (`amountToPay`) obtained from flash loans. Later, the function will check the holdTokenAmtIn is no large than decodedData.holdTokenDebt.

```
// Swap tokens to repay the flash loan
uint256 holdTokenAmtIn = _v3SwapExact(
    v3SwapExactParams({
        isExactInput: false,
        fee: decodedData.fee,
        tokenIn: decodedData.holdToken,
        tokenOut: decodedData.saleToken,
        amount: amountToPay
    })
);
decodedData.holdTokenDebt -= decodedData.zeroForSaleToken
    ? decodedData.amounts.amount1
    : decodedData.amounts.amount0;

// Check for strict route adherence, revert the transaction if conditions are
↪  not met
(decodedData.routes.strict && holdTokenAmtIn >
↪  decodedData.holdTokenDebt).revertError(
```

```
    ErrLib.ErrorCode.SWAP_AFTER_FLASH_LOAN_FAILED
);
```

In the function `_excuteCallback`, the amount of token finally obtained by the user through flash loan is `flashBalance`, which is the balance of the contract.

```
// Transfer the flashBalance to the recipient
decodedData.saleToken.safeTransfer(decodedDataExt.recipient, flashBalance);
// Invoke the WagmiLeverage callback function with updated parameters
IWagmiLeverageFlashCallback(decodedDataExt.recipient).wagmiLeverageFlashCallback(
    flashBalance,
    interest,
    decodedDataExt.originData
);
```

Now let me describe how the attacker compromises the flash loans.

First, the attacker makes a donation to the `FlashLoanAggregator` contract before the victim performs a flash loan (using front-run). Then victim performs a flash loan, and he/she will get much more flashBalance than expected. Finally, in the function `wagmiLeverageFlashCallback`, the holdTokenAmtIn is larger than experted, which leads to fail.

Impact

DOS

Code Snippet

https://github.com/sherlock-audit/2024-03-wagmileverage-v2/blob/main/wagmi-leverage/contracts/FlashLoanAggregator.sol#L553-L560

https://github.com/sherlock-audit/2024-03-wagmileverage-v2/blob/main/wagmi-leverage/contracts/abstract/LiquidityManager.sol#L460-L478

Tool used

Manual Review

Recommendation

In the function `_excuteCallback`, the amount of token finally obtained by the user through flash loan should be the the balance difference during the flash loan period.

Discussion

WangSecurity

seems to be working as expected since all these parameters are user supplied (I mean decodedData.routes.strict)

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/RealWagmi/wagmi-leverage/commit/37b0268ec936d63beee873dd61b096b47cd673c1.

WangSecurity

After additional discussions, we decided to make this a valid medium, cause it's a valid attack vector, but takes a lot of funding and additional external factors

spacegliderrrr

Fix looks good, in case aggregator has extra funds, callback will not have to repay for them.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.