SHERLOCK

Security Review For Lode

Collaborative Audit Prepared For:	**Lode**
Lead Security Expert(s):	**eeyore** **xiaoming90**
Date Audited:	**February 3 - February 9, 2025**
Final Commit:	**8af694b**

Introduction

LODE is an intents-based derivatives exchange deploying on Berachain. This audit will focus on their first structured yield farming funding rate product.

Scope

Repository: Intent-X/sf-core-contracts

Audited Commit: 9b8e7574f42db41d5c759ca119f1d49ec18c0e0c

Final Commit: 8af694bfbd4857b76b60352ae6e8213de399cfe3

Files:

- contracts/funding/Account.sol
- contracts/funding/AccountsCenter.sol
- contracts/funding/BaseAccount.sol
- contracts/funding/interfaces/IAccount.sol
- contracts/funding/interfaces/IAccountsCenter.sol
- contracts/funding/interfaces/IBaseAccount.sol

Final Commit Hash

8af694bfbd4857b76b60352ae6e8213de399cfe3

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.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
5	4	7

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Undervalued fee accounting

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/7

Summary

The fee is not scaled to the appropriate SYMMIO's precision before performing an internal transfer, leading to the protocol receiving fewer fees than expected.

Vulnerability Detail

When calling the Symmio.internalTransfer() function, the amount must be in the original SYMMIO's 18 decimals precision.

However, the `fee` variable in Line 783 below is in the collateral token's native precision (e.g., USDC = 6 decimals). Thus, the fee received by the protocol's treasury will be much fewer than expected.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L775

```
File: Account.sol
775:             if (fee > 0) {
776:                 finalBalance -= fee;
777:                 positionsInfo[id_].fee += fee;
778:                 _simpleMultiAccountCall(
779:                     position.subAccount,
780:                     abi.encodeWithSelector(
781:                         ISymmio.internalTransfer.selector,
782:                         IAccountsCenter(center).tresuary(),
783:                         fee
784:                     )
785:                 );
786:                 emit Fee(id_, fee);
787:             }
```

Impact

Fees received by the protocol's treasury will be much fewer than expected.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol

#L775

Tool Used

Manual Review

Recommendation

Scale the `fee` to SYMMIO's 18 decimals precision before calling the `Symmio.internalTransfer` function.

Discussion

aegas-io

Will fix

xiaoming9090

Fix Confirmed. Fixed here as per recommendation.

Issue H-2: `totalSpotLongAssetInvolved` will be inflated when position is closed if it is partially filled

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/9

Summary

`totalSpotLongAssetInvolved` will be inflated as all reserved LONG assets cannot be sold off during a partial fill.

Vulnerability Detail

Assume that Alice opens a new position via the `_tryOpenPosition` function and sets the `nativeSpotLongAssetInvolved_` to 100 WETH. In this case, the `totalSpotLongAssetInvolved[WETH]` will be set to 100 WETH in Line 1436 below. 100 WETH is reserved.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1436

```
File: Account.sol
1273:     function _tryOpenPosition(
..SNIP..
1433:         if (nativeSpotLongAssetInvolved_ > 0) {
1434:             totalSpotLongAssetInvolved[
1435:                 spotLongAsset_
1436:             ] += nativeSpotLongAssetInvolved_;
..SNIP..
1442:         } else {
1443:             totalSpotLongAssetCollateralReserv += toLongSpotPosition;
1444:         }
```

However, only 50% of the short positions end up being filled. It might not always be possible to fill the entire amount due to the following reasons:

- Order size is too big, and there are insufficient interested PartyB/Hedger to fulfill the orders; OR
- After the first SYMMIO's position (e.g., a position size of 50 WETH) is created, the position incurs a loss due to a sudden change in price. The account goes underwater and is subjected to liquidation. No new position can be created from this point.

Thus, when the `confirmOpenPosition` function is executed, the `positionsInfo[id_].longAssetBalance` will be set to 50 WETH.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L992

```
File: Account.sol
917:     function confirmOpenPosition(
..SNIP..
985:         } else { // @audit-info if position.nativeSpotLongAssetInvolved > 0
986:             spotLongOutAmount =
987:                 (spotLongOutAmount * filledPercentage) /
988:                 _PERCENTAGE_PRECISION;
989:         }
990:         positionsInfo[id_]
991:             .filledShortPositionAmount += filledShortPositionAmount_;
992:         positionsInfo[id_].longAssetBalance += spotLongOutAmount;
```

Scenario 1 - Position remains healthy

After some time, Alice decided to close the position. Thus, she calls `sellSpotLongAsset` function to swap all the existing LONG assets to the collateral. The `positionsInfo[id_].longAssetBalance` will reduce from 50 WETH to 0 AND the `totalSpotLongAssetInvolved[WETH]` will reduce from 100 WETH to 50 WETH.

Note that the `sellSpotLongAsset` function cannot be executed again to sell off the remaining `totalSpotLongAssetInvolved[WETH] = 50 WETH` because the `positionsInfo[id_].longAssetBalance` is already zero at this point and will revert due to underflow.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1067

```
File: Account.sol
1020:     function sellSpotLongAsset(
..SNIP..
1066:         positionsInfo[id_].exitLongAssetAmount += collateralOutAmount;
1067:         positionsInfo[id_].longAssetBalance -= longAssetToSell_;
1068:         totalSpotLongAssetInvolved[position.spotLongAsset] -=
↪  longAssetToSell_;
```

Since the `positionsInfo[id_].longAssetBalance` is now equal to zero, Alice can close the position and wind up the current subaccount (deallocate & withdraw)

However, the issue is that after winding up the subaccount, the `totalSpotLongAssetInvolved[WETH]` will still remain at 50 WETH.

The current design assumes that all the 100 WETH will be entirely filled before the position is closed. However, the position may be closed before all the 100 WETH is filled, leading to the above issue. As a result, the `totalSpotLongAssetInvolved[WETH]` will be inflated, leading to the LONG asset (e.g., WETH here) being stuck and cannot be utilized.

Scenario 2 - Position becomes unhealthy and account is subjected to liquidation

Assume that the account is being liquidated. The position will be closed. Similarly, users will call the `sellSpotLongAsset` function, but only up to 50 WETH can be sold off. The `totalSpotLongAssetInvolved[WETH]` remains at 50 WETH, similar to the problem in the first scenario.

Impact

`totalSpotLongAssetInvolved` will be inflated, leading to LONG assets being stuck.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1436

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L992

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1067

Tool Used

Manual Review

Recommendation

The `sellSpotLongAsset` function should be redesigned to handle the edge case when there is a partial fill. In the above scenario, the `sellSpotLongAsset` function should allow all the reserved 100 WETH (LONG asset) to be sold off.

Discussion

aegas-io

Will fix

xiaoming9090

Fix Confirmed.

The updated implementation will track the actual amount of `nativeSpotLongAssetInvolved` utilised via the `nativeSpotLongAssetUsed` variable in here.

Using the same scenario one from the report, when the position is closed via the confirmClosePosition function, the _tryUnreserveRest function will be executed, and the unused long assets of 50 WETH will be ”unreserved”/released in Line 1581 of the `_tryUnreserveRest` function. Users can then withdraw the unused long assets from their accounts.

Issue H-3: `totalSpotLongAssetCollateralReserv` will be inflated when position is closed if it is partially filled

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/10

Summary

`totalSpotLongAssetCollateralReserv` will be inflated as unused reserved collateral in `totalSpotLongAssetCollateralReserv` were not released.

Vulnerability Detail

Note

Note: This issue is quite similar to "`totalSpotLongAssetInvolved` will be inflated when position is closed if it is partially filled" issue

This issue happens when `nativeSpotLongAssetInvolved_ == 0`, while the latter occurs when `nativeSpotLongAssetInvolved_ > 0`. However, since different state variables are affected, it is easier to keep track of the problems with two separate issues.

Assume that Alice opens a new position via the `_tryOpenPosition` function. She sets the `amount_` to 400 USDC and `nativeSpotLongAssetInvolved_` to zero.

In this case, `toLongSpotPosition` will be 300 USDC while `toShortSymmioPosition` will be 100 USDC.

The current state at this point is as follows:

- `totalSpotLongAssetInvolved[ETH]` = 0 ETH
- `positionsInfo[id_].longAssetBalance` = 0 ETH
- `totalSpotLongAssetCollateralReserv` = 300 USDC

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1443

```
File: Account.sol
1273:     function _tryOpenPosition(
..SNIP..
1433:         if (nativeSpotLongAssetInvolved_ > 0) {
..SNIP..
1442:         } else { // @audit-info nativeSpotLongAssetInvolved_ == 0 => True
```

```
1443:             totalSpotLongAssetCollateralReserv += toLongSpotPosition;
1444:         }
```

After the `_tryOpenPosition` function is executed, the `totalSpotLongAssetCollateralReserv` will be 300 USDC.

Someone perform a 50% partial fill of the SYMM's short position. Thus, `totalSpotLongAssetCollateralReserv` will be reduced by 150 USDC (300 USDC * 50%) from 300 USDC to 150 USDC. Assume the price of USDC and ETH is 1:1 for simplicity's sake. In this case, 150 USDC will be swapped for 150 ETH.

The current state at this point is as follows:

- `totalSpotLongAssetInvolved[ETH]` = 150 ETH
- `positionsInfo[id_].longAssetBalance` = 150 ETH
- `totalSpotLongAssetCollateralReserv` = 150 USDC

Scenario 1 - Position remains healthy

Alice decided to close her position. Thus, she must sell off all her LONG asset (ETH). After selling off all her ETH, she will receive 150 USDC in return. The state will be as follows:

- `positionsInfo[id_].longAssetBalance` = 150 ETH - 150 ETH = 0
- `totalSpotLongAssetInvolved[ETH]` = 150 ETH - 150 ETH = 0
- `totalSpotLongAssetCollateralReserv` = 150 USDC

She then proceeds to call `confirmClosePosition() => deallocateFromSubAccount() => withdrawFromSubAccount()` to close the existing position completely.

At the end, after the subaccount and its position are closed and cleared completely, we can still see that the `totalSpotLongAssetCollateralReserv` remains at 150 USDC.

As a result, `totalSpotLongAssetCollateralReserv` will always be inflated, resulting in collateral tokens (USDC) in the contract being underrepresented or stuck when `getAvailableCollateralBalance()` is called.

Scenario 2 - Position becomes unhealthy and account is subjected to liquidation

Assume that the account is being liquidated. The position will be closed. Similarly, users will call the `sellSpotLongAsset` function to sell off all existing LONG assets (ETH). The same issue described in the first scenario also occurs here.

Impact

`totalSpotLongAssetCollateralReserv` will be inflated, leading to collateral assets being stuck.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1443

Tool Used

Manual Review

Recommendation

Any unused reserved collateral in `totalSpotLongAssetCollateralReserv` should be released when the position is closed.

Discussion

aegas-io

Will fix

xiaoming9090

Observed that the solution to address the scenarios highlighted in the report is to execute the `_tryUnreserveRest()` function when closing the position to release the unused reserved collateral in `totalSpotLongAssetCollateralReserv`. Using back the same example in the report, when the `_tryUnreserveRest`, the unused long reserved assets of 150 USDC will be released from `totalSpotLongAssetCollateralReserv` at Line 1593 below.

```
File: Account.sol
1573:     function _tryUnreserveRest(uint256 id_) internal {
1574:         DeltaNeutralPosition memory position = positionsInfo[id_];
1575:         if (!position.alreadyUnreserved) {
1576:             if (
1577:                 _symmio().getPartyAPendingQuotes(position.subAccount).length
 ↪  ==
1578:                 0
1579:             ) {
1580:                 if (position.nativeSpotLongAssetInvolved > 0) {
1581:                     totalSpotLongAssetInvolved[position.spotLongAsset] -=
1582:                         position.nativeSpotLongAssetInvolved -
1583:                         position.nativeSpotLongAssetUsed;
1584:                 } else {
1585:                     uint256 filledPercentage = (position
1586:                         .filledShortPositionAmount * _PERCENTAGE_PRECISION) /
1587:                         position.initToShortAssetAmount;
1588:
```

```
1589:                     uint256 unusedSpotLongAssetCollateralReserv =
↪  (filledPercentage *
1590:                             position.initToLongAssetAmount) /
1591:                             _PERCENTAGE_PRECISION;
1592:
1593:                     totalSpotLongAssetCollateralReserv -=
1594:                         position.initToLongAssetAmount -
1595:                         unusedSpotLongAssetCollateralReserv;
1596:                 }
1597:
1598:                 positionsInfo[position.id].alreadyUnreserved = true;
1599:             }
1600:         }
1601:     }
```

However, it was observed that there are two math division operations in Lines 1585 and 1589 above, which might result in rounding error. The outcome is that the `unusedSpotLongAssetCollateralReserv` might be lower than expected (e.g., 1 wei or few wei).

Assume that `unusedSpotLongAssetCollateralReserv` is lower than expected, then (`position.initToLongAssetAmount - unusedSpotLongAssetCollateralReserv`) will end up larger than expected.

Assume that there is only one position left in the system. If (`position.initToLongAssetAmount - unusedSpotLongAssetCollateralReserv`) is larger than expected, in a rare edge case, it might underflow when attempting to subtract the value from `totalSpotLongAssetCollateralReserv` in Line 1593, likely due to 1 or few wei difference.

To eliminate any potential rounding error, it is recommended to use a state variable to keep track of the reserved long collateral used instead of computing it on the fly with multiple division operations that might be susceptible to rounding errors.

```
function _tryUnreserveRest(uint256 id_) internal {
..SNIP..
            } else {
-                uint256 filledPercentage = (position
-                    .filledShortPositionAmount * _PERCENTAGE_PRECISION) /
-                    position.initToShortAssetAmount;
-
-                uint256 unusedSpotLongAssetCollateralReserv = (filledPercentage *
-                        position.initToLongAssetAmount) /
-                        _PERCENTAGE_PRECISION;

                totalSpotLongAssetCollateralReserv -=
                    position.initToLongAssetAmount -
-                   unusedSpotLongAssetCollateralReserv;
+           position.spotLongAssetCollateralReservUsed
            }
```

```
            positionsInfo[position.id].alreadyUnreserved = true;
        }
    }
}
```

```
function confirmOpenPosition(
..SNIP..
        spotLongOutAmount =
            IERC20Upgradeable(position.spotLongAsset).balanceOf(
                address(this)
            ) -
            balanceBefore;
        if (spotLongOutAmount < swapRouterLongAssetMinAmountOut_) {
            revert InsufficientAmountOut();
        }
        totalSpotLongAssetCollateralReserv -= collateralToSwap;
+   positionsInfo[id_].spotLongAssetCollateralReservUsed += collateralToSwap;
```

xiaoming9090

On a side note, if the above recommendation is adopted, the following refactor can be made to simplify the code:

```
function deallocateFromSubAccount(
..SNIP..
-    uint256 filledPercentage = (position.filledShortPositionAmount *
-        _PERCENTAGE_PRECISION) / position.initToShortAssetAmount;
-
-    uint256 usedSpotLongAssetCollateralReserv = (filledPercentage *
-        position.initToLongAssetAmount) / _PERCENTAGE_PRECISION;
-
    uint256 usedCollateralBalance = position
        .additionalShortPositionBalance +
        position.initToShortAssetAmount +
-       usedSpotLongAssetCollateralReserv;
+       positionsInfo[id_].spotLongAssetCollateralReservUsed
```

xiaoming9090

Fixed Confirmed. Change made in https://github.com/Intent-X/sf-core-contracts/commit/76a757daaf0f433835850ef924f0364c446a7575

Issue H-4: The `confirmOpenPosition()` function should support the `QUOTE_CANCEL_PENDING` position status

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/14

Summary

In certain situations, SYMMIO can ignore a user's request to cancel their quote, leading to the quote being filled and the user's funds becoming locked in the `Account` contract.

Vulnerability Detail

This issue arises when a user calls the `requestToCancelQuote()` function on a quote that is already `LOCKED` for filling. In such cases, the quote is assigned a `CANCELED_PENDING` status within the SYMMIO system.

The possible next `quoteStatus` values after it being set to `CANCELED_PENDING` are:

- `CANCELLED`
- `OPENED`
- `LIQUIDATED_PENDING`
- `EXPIRED`

Unfortunately, within the `Account` contract, when a quote receives the `CANCELED_PENDING` status, the internal position status is updated to `QUOTE_CANCEL_PENDING`. When this position status is set, the only accepted next quote status is `CANCELLED`.

```
File: Account.sol
674:     function forceCancelQuote(
675:         bool shouldRefund_,
676:         uint256 id_
677:     ) external gasRefund(shouldRefund_) onlyOwnerOrKeeper {
678:         DeltaNeutralPosition memory position = positionsInfo[id_];
679:         _checkStatus(
680:             position.status,
681:             DeltaNeutralPositionStatus.QUOTE_CANCEL_PENDING
682:         );
..SNIP..
692:
693:         _checkQuoteStatus(quoteStatus, QuoteStatus.CANCELED);
```

However, within SYMMIO, a `CANCELED_PENDING` quote can still be filled by PartyB and receive an `OPENED` status. In such a situation, user funds become locked, and the position may be liquidated due to market conditions or accrue funding fees.

Even if closed with the `closeSymmioPosition` function, the position cannot be finalized internally.

Impact

User funds may become locked in the subAccount.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L654-L657

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L693

Tool Used

Manual Review

Recommendation

Keepers should support this scenario and correctly report an opened position in the `confirmOpenPosition()` function:

```
    function confirmOpenPosition(
..SNIP..
        if (
            position.status !=
            DeltaNeutralPositionStatus.WAITING_CONFIRMATION &&
+           position.status != DeltaNeutralPositionStatus.QUOTE_CANCEL_PENDING &&
            position.status != DeltaNeutralPositionStatus.POSITION_OPENED
        ) {
            revert InvalidDeltaNeutralPositionStatus();
        }
```

Discussion

aegas-io

Will fix

0xklapouchy

Fix Confirmed. Fixed here as per recommendation.

Issue H-5: Incorrect PnL calculation during partial fill

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/18

Summary

When closing a LODE's position, the system only checks the first opened position's status but does not verify that the rest of the open positions and pending quotes are closed. As a result, the subaccount's allocated balance might not reflect the entire sum of profit/loss, as some positions/quotes remain open, leading to incorrect calculation.

Vulnerability Detail

The PartyB/Hedger at SYMMIO can perform a partial fill (Reference). Assume that during the `Account.tryOpenPosition()` transaction, a 100 ETH short quote is created at SYMMIO.

It is possible that:

- PartyB_1 fills the first 50 ETH of the short quote (quoteID = 900). Created short position (PosID=1234) and spawned a new short quote (quoteID = 901) with the remaining size of 50 ETH.
- PartyB_2 fills the 30 ETH of the short quote (quoteID = 901). Created short position (PosID=1235) and spawned a new short quote (quoteID=902) with the remaining size of 20 ETH.
- Pending short quote (quoteID=902) of 20 ETH remains unfilled

In this case, two (2) short positions with different position IDs or quote IDs will be opened for a single LODE's position.

Note

One crucial point is that when the original quote is partially filled, a new quote with a **different quote ID** will be created/"spawned" (Reference).

Within the `confirmClosePosition()` function, it only checks whether the first short position (Position ID = 1234) is closed. If it's closed, the LODE's position will be considered closed (aka status = `POSITION_WAITING_DEALLOCATE`).

However, for a LODE's position to be considered as fully closed, all short positions and pending quotes of the subaccount must be closed.

In this case, we can see a scenario where the first short position is closed, but the second position (PosID=1235) and a pending quote (quoteID=902) remain open. As a result, a 30 ETH short position + 20 ETH pending short quote will remain open at SYMMIO's end after the LODE's position is closed.

Impact

The profit & loss (PnL) calculation of the LODE's position will be incorrect. In this scenario, the system will assume that the current LODE's position to be closed to incur a huge loss because the assets in the second short position and pending quote are ignored. On the other hand, the subsequent LODE's position opens with this specific subaccount will have its profit inflated.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1173

Tool Used

Manual Review

Recommendation

Consider only closing a LODE position if all the following requirements are met:

- All open positions of the subaccount are closed
- All pending quotes of the subaccount are closed
- Subaccount is not in liquidation status (`liquidationStatus == false`). Refer to "Position is closed even if liquidation is still on-going" issue for more details

Discussion

aegas-io

Will fix

The idea was to optimize the process, gas consumption and put the need to process extreme cases on Keeper (partialy filled), which has this capability, but since there are simple ways to check for exist open/pending positions, this will be improved and a strong check will be added.

This is the reason why there are no hard validations of quoteId in a certain set of methods

xiaoming9090

Fix Confirmed. Fixed as per recommendation. Position can only be closed if:

- The subaccount is not under liquidation (Reference) or when the liquidation is completed.

- All opened positions and pending quotes of the subaccount are closed (Reference)

Issue M-1: The `forceCancelQuote()` function should support the `EXPIRED` quote status

Source:
https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/15

Summary

If the quote status is set to `CANCEL_PENDING` after calling the `requestToCancelQuote()` function, the quote can be updated not only by PartyB (to `CANCELLED` or `OPENED`) but also by anyone using the `expireQuote()` function in PartyA (to `EXPIRED`).

Vulnerability Detail

When the `requestToCancelQuote()` function is called while the quote is `LOCKED` for filling but has not yet expired, its status is updated to `CANCEL_PENDING`.

There is a potential edge case where a position in the `CANCEL_PENDING` state can expire before the required wait time for it to be available for force-closure.

In such a situation, anyone can grief another user by calling the `expireQuote()` function (Reference) in PartyA, locking the user out of their position and funds.

Impact

User funds may become locked in the subAccount.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L693

Tool Used

Manual Review

Recommendation

The `forceCancelQuote()` function should support this scenario and correctly handle expired quotes:

```
    function forceCancelQuote(
        bool shouldRefund_,
        uint256 id_
    ) external gasRefund(shouldRefund_) onlyOwnerOrKeeper {
..SNIP..
-       _checkQuoteStatus(quoteStatus, QuoteStatus.CANCELED);
+       if (
+           quoteStatus != QuoteStatus.EXPIRED &&
+           quoteStatus != QuoteStatus.CANCELED
+       ) {
+           revert InvalidQuoteStatus();
+       }
```

Discussion

aegas-io

Will fix

0xklapouchy

Fix Confirmed.

The `_checkQuoteStatus()` check was removed from the `forceCancelQuote()` function, and the function was updated to support positions with multiple quote IDs.

The `EXPIRED` quote status is now correctly supported within the `forceCancelQuote()` function for positions with `QUOTE_CANCEL_PENDING` and `POSITION_OPENED` statuses.

Issue M-2: User premature calling of `closeSymmioPosition()` and `forceCloseSymmioPosition()` functions leads to fund lock

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/16

Summary

A user can call the `closeSymmioPosition()` and `forceCloseSymmioPosition()` functions before the Keeper correctly updates positions by calling the `confirmOpenPosition()` function, which is required to provide all information and confirmation of position fulfillment in SYMMIO.

Vulnerability Detail

If these functions are called by the user before the position is moved to `POSITION_OPENED` by the Keeper, it results in a situation where the user's funds become locked.

In such a case, the Keeper cannot open the position because the `confirmOpenPosition()` function will revert on the quote `OPENED` check:

```
File: Account.sol
917:     function confirmOpenPosition(
918:         bool shouldRefund_,
..SNIP..
933:
934:         _checkQuoteStatus(
935:             _getQuoteStatus(position.quoteId),
936:             QuoteStatus.OPENED
937:         );
```

At the same time, the position cannot proceed further in its lifecycle by calling `confirmClosePosition()`, as it will revert on the position status check:

```
File: Account.sol
1173:     function confirmClosePosition(
1174:         bool shouldRefund_,
..SNIP..
1178:         _checkStatus(
1179:             position.status,
1180:             DeltaNeutralPositionStatus.POSITION_OPENED
1181:         );
```

Impact

User funds may become locked in the subAccount.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1095-L1099

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L934-L937

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1178-L1181

Tool Used

Manual Review

Recommendation

Add a sanity check within the `closeSymmioPosition()` and `forceCloseSymmioPosition()` functions:

```
    function closeSymmioPosition(
        bool shouldRefund_,
        address subAccount_,
        CloseRequestPositionParams calldata closeRequestParams_
    ) external gasRefund(shouldRefund_) onlyOwnerOrKeeper {
+       DeltaNeutralPosition memory position = positionsInfo[id_];
+       _checkStatus(
+           position.status,
+           DeltaNeutralPositionStatus.POSITION_OPENED
+       );
+
        _simpleMultiAccountCall(
```

Discussion

aegas-io

Will fix

0xklapouchy

Partially Fixed.

It is recommended to add sanity check in both the `closeSymmioPosition()` and `forceCloseSymmioPosition()` functions.

However, recommended sanity check was only added to `closeSymmioPosition()` here, but not to `forceCloseSymmioPosition()`.

0xklapouchy

Fix Confirmed.

Missing sanity check was added to the `forceCloseSymmioPosition()` function here.

Issue M-3: The `CANCEL_CLOSE_PENDING` quote status is not supported

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/20

Summary

When `requestToCancelCloseRequest` is called, the quote status may update to `CANCEL_CLOSE_PENDING`. This status is not supported within the `Account` contract flow.

Vulnerability Detail

If `requestToCancelCloseRequest` is called for a quote whose original deadline has not yet expired, its status will update to `CANCEL_CLOSE_PENDING` (Reference 1). In the current `Account` contract, only acceptance from PartyB can set the status back to `OPENED` (Reference 2). However, relying solely on PartyB is insufficient–if the quote remains in the `CANCEL_CLOSE_PENDING` status, user funds will remain locked.

Within the `Account` contract, the PartyA `forceCancelCloseRequest()` function is not utilized to prevent this limbo state.

Impact

User funds can become locked in an unsupported quote state.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1128-L1132

Tool Used

Manual Review

Recommendation

Add support for the PartyA `forceCancelCloseRequest()` function for quote in `CANCEL_CLOSE_PENDING` status.

Discussion

aegas-io

Will fix

xiaoming9090

Fix Confirmed. Fixed here as per recommendation.

Issue M-4: No option to cancel a quote for the remaining unfilled amount of a position

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/22

Summary

When SYMMIO partially fills the original quote for a position, there is no option to request the cancellation of the quote for the remaining unfilled amount.

Vulnerability Detail

Note

One crucial point is that when the original quote is partially filled, a new quote with a **different quote ID** will be created/spawned (Reference).

Since `requestToCancelQuote()` only supports the original quote ID for cancellation, there is no option to cancel any `PENDING` new quote that spawns when the original position quote is partially filled.

```
File: Account.sol
607:     function requestToCancelQuote(
608:         bool shouldRefund_
..SNIP..
617:         QuoteStatus quoteStatus = _getQuoteStatus(position.quoteId);
```

This leads to a situation where the unfilled part of the position can remain in the `PENDING` status until it is either filled or expires.

Impact

Users are unable to cancel `PENDING` quotes, causing their positions to remain in a limbo state.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L607

Tool Used

Manual Review

Recommendation

The `requestToCancelQuote()` function should support multiple quote IDs per position.

Discussion

aegas-io

Will fix

0xklapouchy

Fix Confirmed.

The reimplementation of the `requestToCancelQuote()` function allows determining the quote for cancellation by leveraging the quoteId_ function parameter, as such supporting multiple quote IDs per position.

Issue L-1: Incorrect scaling of `amount_` during withdrawal

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/8

Summary

The `amount_` is not scaled appropriately during withdrawal, leading to incorrect validation and users withdrawing fewer assets than expected.

Vulnerability Detail

The comment of this function (Line 840 below) mentioned that the `amount_` must be in the collateral token's native decimals. In this case, the collateral is USDC, which will be in 6 decimals precision.

However, the `availableBalance` variable in Line 871 is assigned to the value returned from `_symmio().balanceOf(position.subAccount)` function, which is denominated in SYMMIO's 18 decimals precision.

Thus, the logic and conditional check in Line 873 below will be incorrect since one variable (`availableBalance`) is denominated in SYMMIO's 18 decimals while the other (`amount_`) is denominated in native precision.

Note: All balance and allocatedBalance within a SYMMIO's subaccount are scaled to 18 decimals.

In addition, the `amount_` parameter in the Symmio.internalTransfer function in Line 882 has to be denominated in SYMMIO's 18 decimals precision. Thus, if the `amount_` is denominated in the collateral token's native decimals, users will withdraw fewer assets than expected.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/e8d7835f69dfcbc0d617d737fdf92a8c42a93c66/sf-core-contracts/contracts/funding/Account.sol#L830

```
File: Account.sol
840:      * @param amount_ The amount of collateral to withdraw, specified in the
↪  collateral token's native decimals.
..SNIP..
File: Account.sol
857:     function withdrawFromSubAccount(
858:         uint256 id_,
859:         address recipient_,
860:         uint256 amount_
861:     ) external onlyOwner {
..SNIP..
```

```
871:         uint256 availableBalance = _symmio().balanceOf(position.subAccount);
872:
873:         if (availableBalance < amount_ || availableBalance == 0) {
874:             revert InsufficientSymmioBalance();
875:         }
876:
877:         _simpleMultiAccountCall(
878:             position.subAccount,
879:             abi.encodeWithSelector(
880:                 ISymmio.internalTransfer.selector,
881:                 recipient_,
882:                 amount_
883:             )
884:         );
```

Impact

Incorrect validation and users withdrawing fewer assets than expected.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/e8d7835f69dfcbc0d617d737fdf92a8c42a93c66/sf-core-contracts/contracts/funding/Account.sol#L830

Tool Used

Manual Review

Recommendation

Consider scaling the `amount_` accordingly, depending on the operation performed:

- If the amount is to be passed into the `Symmio.internalTransfer` function, it has to be scaled up SYMMIO's 18 decimals precision.
- When a comparison is performed, both variables must be scaled to a similar scale.

Discussion

aegas-io

Will fix

The expected behavior is a transfer in 18 decimals always in this case, errors in natspec, no correct description. In the case of transferring exactly the collateral decimals, we

face the problem of dust on the symmio balance, which can be up to 9.[9]e12 in the case of collateral with decimals 6

xiaoming9090

Fix Confirmed. The NatSpec has been updated here to document that the `_amount` must be denominated in 18 decimals.

Issue L-2: Computation of `toTransferAmount` can be skipped if `fundingShortAmount_ == toShortSymmioPosition`

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/11

Summary

The computation of `toTransferAmount` can be skipped if `fundingShortAmount_` is equal to `toShortSymmioPosition`.

Vulnerability Detail

`toTransferAmount` only needs to be computed if `fundingShortAmount_` is insufficient to cover the entire amount of `toShortSymmioPosition`. If `fundingShortAmount_` and `toShortSymmioPosition` are equal, the computation of `toTransferAmount` can be skipped.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1307

```
-       if (fundingShortAmount_ <= toShortSymmioPosition) {
+       if (fundingShortAmount_ < toShortSymmioPosition) {
            toTransferAmount += toShortSymmioPosition - fundingShortAmount_;
        }
```

Impact

Unnecessary operation/Gas efficiency

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/e8d7835f69dfcbc0d617d737fdf92a8c42a93c66/sf-core-contracts/contracts/funding/Account.sol#L1266

Tool Used

Manual Review

Recommendation

Consider making the changes as shown above.

Discussion

aegas-io

Will fix

xiaoming9090

Fixed Confirmed. Fixed as per recommendation (Reference)

Issue L-3: Dust amount left in `totalScheduledPositionsCollateralReserv`

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/13

Summary

The dust amount in the `totalScheduledPositionsCollateralReserv` is not cleared after all the scheduled positions are fulfilled, leading to `totalScheduledPositionsCollateralReserv` to be inflated.

Vulnerability Detail

When all the schedule positions are fulfilled, the unused dust (`nativeSpotLongAssetReserved - nativeSpotLongAssetInvolved`) will be removed from the `totalSpotLongAssetInvolved`. This is to prevent `totalSpotLongAssetInvolved` from slowly becoming inflated due to accumulating dust, which will affect the `_checkOnAvailableSpotLongAssetBalance` function's calculation.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L587

```
File: Account.sol
488:     function tryOpenPositionFromKeeper(
489:         bool shouldRefund_,
..SNIP..
577:
578:         if (info.count == info.openedCount + 1) {
579:             uint256 restSpotLongAssetReserved = info
580:                 .nativeSpotLongAssetReserved -
581:                 scheduledPositionRequests[scheduledPositionRequestId_]
582:                     .nativeSpotLongAssetInvolved;
583:
584:             if (restSpotLongAssetReserved > 0) {
585:                 totalSpotLongAssetInvolved[
586:                     info.spotLongAsset
587:                 ] -= restSpotLongAssetReserved;
588:             }
589:         }
```

However, it is also possible that there will be unused reserved collateral in the `totalScheduledPositionsCollateralReserv`.

Assume that the `amount` is 9 and `count` is 4. In this case, `totalAmount` will be 36. If

`nativeSpotLongAssetReserved_` > 0, then `totalAmount` will become 9 (divided by 4).

`totalScheduledPositionsCollateralReserv` will be 9. When a scheduled position is fulfilled, only 2 (info.amount / 4 = 9 / 4 = 2) deducted from `totalScheduledPositionsCollateralReserv` each time.

When all the four (4) scheduled positions are fulfilled, only 8 (2 * 4) will be deducted from `totalScheduledPositionsCollateralReserv`, leaving behind 1 wei of dust.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L547

```
File: Account.sol
546:         if (nativeSpotLongAssetInvolved_ > 0) {
547:             totalScheduledPositionsCollateralReserv -= info.amount / 4;
..SNIP..
553:         } else {
554:             totalScheduledPositionsCollateralReserv -= info.amount;
555:         }
```

Thus, the remaining dust from the reserved collateral also needs to be removed from `totalScheduledPositionsCollateralReserv`.

Impact

Dust amount will slowly accumulate in the `totalScheduledPositionsCollateralReserv`, leading to `totalScheduledPositionsCollateralReserv` to be inflated.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L547

Tool Used

Manual Review

Recommendation

Consider clearing the dust amount in the`totalScheduledPositionsCollateralReserv` after all the scheduled positions have been fulfilled.

Discussion

aegas-io

Will fix

xiaoming9090

Fixed Confirmed. Fixed in here.

`totalScheduledPositionsCollateralReserv` will be reset to zero once all scheduled positions have been fulfilled (`openSchedulePositionCount == 0`) to clear any remaining dust.

Issue L-4: Position is closed even if liquidation is still on-going

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/17

Summary

The position is closed even though the liquidation process has not yet been completed. As a result, the subaccount is wrongly marked as available, leading to a revert if users use the subaccount to open a new position.

Vulnerability Detail

The liquidation in SYMMIO involves five (5) stages:

1. liquidatePartyA
2. setSymbolsPrice
3. liquidatePendingPositionsPartyA
4. liquidatePositionsPartyA (Quote will be marked with `QuoteStatus.LIQUIDATED` in this stage)
5. settlePartyALiquidation

When the quote's status is set to `QuoteStatus.LIQUIDATED`, it does not mean that the liquidation has been completed.

After the liquidation, the subaccount's allocated balance will usually be zero. However, in some instances, it is possible that after the liquidation, the subaccount's allocated balance will be a non-zero as it receives reimbursement (Reference).

The reimbursement can happen if:

1. There are pending quotes that prepaid the trading fee. During liquidation, the pending quote will be cancelled and trading fee will be refunded back. If any PartyB performs a partial fill, there will be a pending quote, and this scenario will happen.
2. SYMMIO has a feature called Deferred Liquidation. In short, assume that Bob's account is subjected to liquidation at T0. If at T1, Bob deposits 100 USDC collateral to its account and its account becomes healthy, the liquidator can ”go back in time” and liquidate Bob based on his account's state at T0 (This is assuming he does not perform any action that increases his account's nonce). In this case, the 100 USDC additional collateral he deposited will be reimbursed back to his account after his account is liquidated (Reference)

The reimbursement will only occur at the last stage of the liquidation process (settlePartyALiquidation).

In the current LODE design, it is possible that when the liquidation is still at Stage 4 (`liquidatePositionsPartyA`), the process of closing a subaccount (`confirmClosePosition -> deallocateFromSubAccount -> withdrawFromSubAccount`) can be completed. The subaccount will be marked as "clean/free" to be available for use for a new position, but in reality the liquidation process against this subaccount has not been completed yet and the subaccount's state in SYMMIO has not been reset.

Impact

LODE will consider a subaccount available for a new position while, in reality, the subaccount has not yet completed its liquidation. If anyone uses this subaccount to open a new position, it will revert as SYMMIO does not allow any operation to be performed against a liquidated account.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1173

Tool Used

Manual Review

Recommendation

In the event of a liquidation, the LODE's position should only be marked as closed AND the subaccount should only be released/freed when all the following conditions are met:

- The liquidation process is completed (After Stage 5 - settlePartyALiquidation). Once the last stage is completed, the subaccount's `liquidationStatus` will revert to `false`. (Reference)
- If the subaccount has multiple open positions, all its existing positions must be liquidated (`status = QuoteStatus.LIQUIDATED`). A subaccount can have multiple open positions if PartyB performs a partial fill.

This ensures that when a subaccount is marked as available, the state of the subaccount is a clean slate (liquidationStatus = false AND zero open positions)

Discussion

aegas-io

Will fix

xiaoming9090

Fix Confirmed. Fixed as per recommendation. Position can only be closed if:

- The subaccount is not under liquidation (Reference) or when the liquidation is completed.
- All opened positions and pending quotes of the subaccount are closed (Reference)

Issue L-5: The `LIQUIDATED_PENDING` quote status is not supported

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/19

Summary

The `LIQUIDATED_PENDING` quote status is not expected within the `Account` contract flow.

Vulnerability Detail

Note

During a partial fill in SYMMIO, a new quote with a new ID will be created. This issue highlights one of the problems associated with that process.

The `LIQUIDATED_PENDING` status was introduced in SYMMIO to handle the liquidation of pending quotes when a subAccount becomes insolvent.

Consider a scenario where a quote is partially filled, and after some time, the user's subAccount becomes insolvent due to a sharp price decline. If liquidation occurs, the new pending quote for the remaining unfilled portion will also be liquidated, resulting in the `LIQUIDATED_PENDING` status for the unfilled quote.

Additionally, if a quote cancellation is requested for the remaining new pending quote, the status can transition to `LIQUIDATED_PENDING` from `CANCEL_PENDING` in some situations.

Impact

A subAccount position within `Account` contract may enter an unexpected state where liquidation finalization becomes impossible. As a result, the long asset portion of the position could become inaccessible to the user.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1185-L1191

Tool Used

Manual Review

Recommendation

Support for the `LIQUIDATED_PENDING` status should be considered within the quote cancellation flow and the close position flow:

```
        QuoteStatus quoteStatus = _getQuoteStatus(position.quoteId);
        if (
            quoteStatus != QuoteStatus.CLOSED &&
+           quoteStatus != QuoteStatus.LIQUIDATED_PENDING &&
            quoteStatus != QuoteStatus.LIQUIDATED
        ) {
            revert InvalidQuoteStatus();
        }
```

Discussion

aegas-io

Won't fix

LIQUIDATED_PENDING refers to quotes that are in PENDING and have not been opened, in this case there must have be 2+ quotes:

1. An open position that will go to LIQUIDATED status
2. A quote that will go to the LIQUIDATED_PENDING status

confirmClosePosition implies the need to close only the main position, and cannot be used for pending/second... quote, in this case, the fate of secondary quotas is not interested if the main one is liquidated

The main position in this case cannot acquire the LIQUIDATED_PENDING status. Therefore, checking for this status is unnecessary

0xklapouchy

Fixed by introduction of fixes for the #17 issue.

Issue L-6: The `AccountsCenter.collateral` value can mismatch with `ISymmio.getCollateral()`

Source:
https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/21

Summary

The `collateral` value is not sanity-checked against `ISymmio.getCollateral()` during initialization or updates.

Vulnerability Detail

In the `AccountsCenter` contract, the `collateral` value can be set to any address. This can lead to two issues:

1. The collateral does not match the expected value, new positions cannot be opened in the `Account` contract.
2. The `collateral` value was updated after positions have been opened, those positions may become unclosable, leaving users unable to exit their trades.

Impact

A mismatch between `AccountsCenter.collateral` and the collateral used by SYMMIO can disrupt position management.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/b59282f25fb82b3b6763b1941c03ac066377705d/AccountsCenter.sol#L176
https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/b59282f25fb82b3b6763b1941c03ac066377705d/AccountsCenter.sol#L197-L208

Tool Used

Manual Review

Recommendation

Implement a sanity check to ensure the `collateral` value matches `ISymmio.getCollateral()` during both initialization and updates.

Discussion

aegas-io

Will fix

xiaoming9090

Partially Fixed.

It is recommended that the check be added in both the `AccountsCenter.initialize()` and `AccountsCenter.setCollateral()` functions. However, it was only added to `AccountsCenter.setCollateral()`, but not `AccountsCenter.initialize()`.

- `AccountsCenter.initialize()` - Check not added.
- `AccountsCenter.setCollateral()` - Check added in here

0xklapouchy

Fix Confirmed.

Recommended check was added to the `AccountsCenter.initialize()` here.

Issue L-7: Opening a new position might revert due to insufficient allocated balance

Source: https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/issues/23

Summary

Opening a new position might revert due to insufficient allocated balance as the current check does not consider the trading fee that SYMMIO charges.

Vulnerability Detail

When a new quote is sent to SYMMIO, SYMMIO will charge a trade fee against the notional value of the position (Reference). If the notional value of the short position is 75 ETH, and the trade fee is 1%, 0.75 ETH will be deducted from the account's `allocatedBalances`.

However, in the `_checkOnAvailableCollateralBalance()` check in Line 1311 below, it does not consider the trade fee. If the `amount_` is 100 ETH, `toShortSymmioPosition` will be 25 ETH. With 3X leverage, the position's notional value will be 75 ETH. In this case, the `_checkOnAvailableCollateralBalance()` only checks if this contract's balance has 25 free ETH to open the position, but does not include the 0.75 ETH trading fee.

If we consider the trade fee, the `toTransferAmount` should be `25.75` ETH.

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1311

```
File: Account.sol
1273:     function _tryOpenPosition(
..SNIP..
1310:
1311:         _checkOnAvailableCollateralBalance(
1312:             (nativeSpotLongAssetInvolved_ > 0 ? 0 : toLongSpotPosition) +
1313:                 toTransferAmount
1314:         );
```

Impact

Opening a new position might revert due to insufficient allocated balance.

Code Snippet

https://github.com/sherlock-audit/2025-01-lode-funding-rate-product/blob/2c57a2ad5d46091910d9341d341b59d1afe3bce0/sf-core-contracts/contracts/funding/Account.sol#L1311

Tool Used

Manual Review

Recommendation

Consider taking into consideration SYMMIO's trading fee when opening a position.

Discussion

aegas-io

Won't fix

The trading fee is included within the allocated balance and is configured by reducing the position (parameters), send quote. Just as the swap fee is deducted from the collateral allocated for the spot long part, the trading fee will be deducted from the short allocated part within the allocated balance.

For example, 25 USD is allocated for the short part, the keeper/fe will calculate the quote size so that `(25 USD) >= position size (partyAmm + cva + lf) + trading fee`

The short part is the allocated available balance, the keeper tries to maximize the position size within this balance but also takes into account the trading fee, as without it it would lead to revert tx (symmio have enough checks for this cases)

xiaoming9090

Acknowledged.

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.