SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	SYMMIO
Prepared by:	Sherlock
Lead Security Expert:	xiaoming90
Dates Audited:	October 3 - October 8, 2024
Prepared on:	October 24, 2024

Introduction

An intent-centric clearing & settlement layer for synthetic derivatives, allowing users to swap exposure to various assets instead of tokens.

Scope

Repository: SYMM-IO/protocol-core

Branch: version_0.8.4

Audited Commit: 8b6d7208a8ac8d64b3ab313039fef882a03af0f4

Final Commit: bfd7bb2389022cd77492fc69950f5abfdb18d22b

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
5	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

xiaoming90

Issue M-1: `settleUpnl` function can be DOSed by other PartyBs/hedgers

Source: https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest-judging/issues/39

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

`settleUpnl` function can be DOSed. Users will not be able to close their positions, which could lead to a loss to the users due to the inability to close their positions. In the report's example, PartyA wants to close its losing position to stop its account from incurring further losses due to unfavorable market conditions. However, since PartyA is unable to close its losing position, PartyA continues to incur more losses and might eventually be liquidated. Furthermore, trading is a time-sensitive activity. Thus, any blockage or delay would eventually lead to a loss of assets.

Root Cause

- Other PartyBs/hedgers can front-run the transaction and increment their own nonce to block the `settleUpnl` transaction from executing.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

Assume the following scenario:

- PartyA has open positions with PartyB_1 and PartyB_2
- PartyA has one (1) losing position with PartyB_1 (PartyA loses, PartyB_1 wins)
- PartyA has one (1) winning position with Party_2 (PartyA wins, PartyB_2 loses)
- PartyA has zero or low allocated balance

PartyA wants to close its losing position to stop its account from incurring further losses due to unfavorable market conditions.

However, the close position transaction cannot go through due to an insufficient allocated balance on PartyA's account. This is a similar scenario described in the Symm IO v0.8.4 documentation.

In order for PartyB_1 to close PartyA's position, it first has to execute the `settleUpnl` function to settle PartyA's unrealized profit so that PartyA's allocated balance will be "refilled" with more funds. Once there are sufficient funds in PartyA's allocated balance, PartyB_1 can proceed to fulfill PartyA's closing request.

The only option to increase the PartyA's allocated balance is to close the PartyA's winning position with PartyB_2.

When the `settleUpnl` function is executed, it will verify the signature provided by Muon via the `verifySettlement` function below. Line 20 below shows that the encoded data in the signature includes the nonce of the PartyBs of the positions/quotes where the PnL is to be settled. In this case, PartyB_2's nonce will be included in the signature.

Assume that the time when PartyB_1 fetches the signature is T0. At T0, PartyB_2's nonce is 890. Thus, the signature is signed against encoded data where PartyB_2's nonce is 890.

At T1, PartyB_1 executes the `settleUpnl` function along with the signature. However, PartyB_2 could front-run the transaction and perform certain actions (e.g., open/close position, charge funding rate) that will increment its own nonce to 891. In this case, the nonce in PartyB_1's signature and PartyB_2's nonce will differ, and the `settleUpnl` transaction will revert. PartyB_2 could repeat this continuously DOS or block PartyB_1 attempts to settle UPNL and close PartyA's positions.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/libraries/muon/LibMuonSettlement.sol#L12

```
File: LibMuonSettlement.sol
12:     function verifySettlement(SettlementSig memory settleSig, address
↪  partyA) internal view {
13:         MuonStorage.Layout storage muonLayout = MuonStorage.layout();
14:         // == SignatureCheck( ==
15:         require(block.timestamp <= settleSig.timestamp +
↪  muonLayout.upnlValidTime, "LibMuon: Expired signature");
16:         // == ) ==
17:         bytes memory encodedData;
18:         uint256[] memory nonces = new
↪  uint256[](settleSig.quotesSettlementsData.length);
19:         for (uint8 i = 0; i < settleSig.quotesSettlementsData.length; i++) {
```

```
20:             nonces[i] = AccountStorage.layout().partyBNonces[QuoteStorage.la
↪  yout().quotes[settleSig.quotesSettlementsData[i].quoteId].partyB][partyA];
21:             encodedData = abi.encodePacked(
22:                 encodedData,  // Append the previously encoded data
23:                 settleSig.quotesSettlementsData[i].quoteId,
24:                 settleSig.quotesSettlementsData[i].currentPrice,
25:                 settleSig.quotesSettlementsData[i].partyBUpnlIndex
26:             );
27:         }
28:         bytes32 hash = keccak256(
29:             abi.encodePacked(
30:                 muonLayout.muonAppId,
31:                 settleSig.reqId,
32:                 address(this),
33:                 "verifySettlement",
34:                 nonces,
35:                 AccountStorage.layout().partyANonces[partyA],
..SNIP..
44:     }
```

Impact

Users will not be able to close their positions, which could lead to a loss to the users due to the inability to close their positions. In the above example, PartyA wants to close its losing position to stop its account from incurring further losses due to unfavorable market conditions. However, since PartyA is unable to close its losing position, PartyA continues to incur more losses and might eventually be liquidated. Furthermore, trading is a time-sensitive activity. Thus, any blockage or delay would eventually lead to a loss of assets.

PoC

No response

Mitigation

No response

Issue M-2: Unauthorized PartyB could settle PNL of other PartyBs and users in the system

Source: https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest-judging/issues/40

Found by

xiaoming90

Summary

The `settleUpnl` function does not include `notPartyB` modifier. As a result, unauthorized PartyB could settle PNL of other PartyBs and users in the system, causing disruption and breaking core protocol functionality. Unauthorized PartyB can prematurely settle PartyB's (victim) positions prematurely at times that are disadvantageous to PartyB, resulting in asset loss for them.

For example, in a highly volatile market, if PartyB's positions temporarily incur a loss due to sudden market fluctuations, the Unauthorized PartyB could immediately settle these positions before the market has a chance to recover. This premature settlement forces PartyB to realize losses that might have otherwise been avoided if the positions had remained open.

Root Cause

- The `settleUpnl` function does not include `notPartyB` modifier.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

The `settleUpnl` function can only be accessed by PartyB, as per the comment below. However, the function is not guarded by the notPartyB modifier.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/facets/Settlement/SettlementFacet.sol#L26

```
File: SettlementFacet.sol
16:     /**
17:      * @notice Allows Party B to settle the upnl of party A position for the
↪  specified quotes.
18:      * @param settlementSig The data struct contains quoteIds and upnl of
↪  parties and market prices
19:      * @param updatedPrices New prices to be set as openedPrice for the
↪  specified quotes.
20:      * @param partyA Address of party A
21:      */
22:     function settleUpnl(
23:         SettlementSig memory settlementSig,
24:         uint256[] memory updatedPrices,
25:         address partyA
26:     ) external whenNotPartyBActionsPaused notLiquidatedPartyA(partyA) {
27:         uint256[] memory newPartyBsAllocatedBalances =
↪  SettlementFacetImpl.settleUpnl(settlementSig, updatedPrices, partyA);
..SNIP..
35:     }
```

Instead, it depends on the `quoteLayout.partyBOpenPositions[msg.sender][partyA].length > 0` at Line 31 below, which is not a reliable method to determine whether the caller is a valid PartyB. The reason is that it is possible that a PartyB (e.g., one that might be removed due to malicious activities) that has already been removed from the system still has residual open positions. In this case, the position's length check will pass, and the unauthorized PartyB could continue to settle the PNL of other PartyBs and users in the system.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/libraries/LibSettlement.sol#L24

```
File: LibSettlement.sol
15:     function settleUpnl(
..SNIP..
30:         require(
31:             isForceClose ||
↪  quoteLayout.partyBOpenPositions[msg.sender][partyA].length > 0,
32:             "LibSettlement: Sender should have a position with partyA"
33:         );
```

Impact

Unauthorized PartyB could settle PNL of other PartyBs and users in the system, causing disruption and breaking core protocol functionality. Unauthorized PartyB can prematurely settle PartyB's (victim) positions prematurely at times that are disadvantageous to PartyB, resulting in asset loss for them.

For example, in a highly volatile market, if PartyB's positions temporarily incur a loss due to sudden market fluctuations, the Unauthorized PartyB could immediately settle these positions before the market has a chance to recover. This premature settlement forces PartyB to realize losses that might have otherwise been avoided if the positions had remained open.

PoC

No response

Mitigation

Include the `notPartyB` modifier to the `settleUpnl` function.

Discussion

MoonKnightDev

the check in the libSettlement.sol file prevents the scenario mentioned: https://github.com/SYMM-IO/protocol-core/blob/eac73bf1d97df96bcd5b19bcc972792ef96c70e1/contracts/libraries/LibSettlement.sol#L30

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/SYMM-IO/protocol-core/pull/57

Issue M-3: Force Close can be DOSed by exploiting `settleUpnl` function

Source: https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest-judging/issues/42

Found by

xiaoming90

Summary

PartyB can abuse the newly implemented `settleUpnl` function to increment the nonce to block PartyA's force close action.

Root Cause

1. A `SettlementSig` signature with `quotesSettlementsData` and/or `partyBs` array being empty can be passed into `settleUpnl` function
2. No cooldown if PartyB is settling its own positions
3. No minimum PnL to be settled (Even 1 wei of PnL can be settled)

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

This report is similar to the report from the previous contest, but with a different attack vector.

Assume that PartyA requests to close a quote via `requestToClosePosition` function. If PartyB does not respond within the cooldown period, PartyA can call the `PartyAFacetImpl.forceClosePosition` to close the quote forcefully.

Within the `forceClosePosition` function:

- In Line 86 below, the verifyHighLowPrice function will verify that the signature is valid. The function will use the nonce of PartyA and PartyB to generate the hash.

- In Line 88 below, the verifySettlement function verifies that the signature is valid. Similarly, it also relies on the nonce of PartyA and PartyB to generate the hash.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/facets/ForceActions/ForceActionsFacetImpl.sol#L86

```
File: ForceActionsFacetImpl.sol
51:     function forceClosePosition(
52:         uint256 quoteId,
53:         HighLowPriceSig memory sig,
54:         SettlementSig memory settlementSig,
55:         uint256[] memory updatedPrices
56:     ) internal returns (uint256 closePrice, bool isPartyBLiquidated, int256
 ↪  upnlPartyB, uint256 partyBAllocatedBalance) {
..SNIP..
86:         LibMuonForceActions.verifyHighLowPrice(sig, quote.partyB,
 ↪  quote.partyA, quote.symbolId);
87:         if (updatedPrices.length > 0) {
88:             LibMuonSettlement.verifySettlement(settlementSig, quote.partyA);
89:         }
```

Attack Path 1 - Incrementing PartyA's nonce

The following describes the attack path:

1. At this point, PartyA's nonce is 200. PartyA fetches the signatures from Muon and bundles them with the `forceClosePosition` transaction, and submits to the mempool to close the quote forcefully as PartyB did not respond to PartyA's close request for an extended period.
2. PartyB front-run PartyA's transaction and exploit the new `settleUpnl` function to increment its nonce to 201.
3. When PartyA's transaction gets executed, since the signature's PartyA nonce is 200, while the PartyA's nonce on-chain is 201, the transaction will revert.
4. Party B could continuously grieve Party A, preventing them from being forced to close their positions.

There are several ways that PartyB can increment the PartyA's nonce via the `settleUpnl` function with no cost or minimum cost:

1. Passing a `SettlementSig` signature with `quotesSettlementsData` and/or `partyBs` array being empty. In this case, the code at Line 34 will increment PartyA's nonce, but the rest of the code that involves the for-loop will be bypassed since the array (`settleSig.quotesSettlementsData.length = 0`) is empty

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/libraries/LibSettlement.sol#L40

```
File: LibSettlement.sol
15:     function settleUpnl(
16:         SettlementSig memory settleSig,
..SNIP..
34:         accountLayout.partyANonces[partyA] += 1;
..SNIP..
40:         for (uint8 i = 0; i < settleSig.quotesSettlementsData.length; i++) {
```

2. The tricks mentioned in "Attack Path 2" section below for incrementing PartyB's nonce can also be applied here to increment PartyA's nonce. Refer to the next section for more details.

Attack Path 2 - Incrementing PartyB's nonce

The following describes the attack path:

1. At this point, PartyB's nonce is 800. PartyA fetches the signatures from Muon and bundles them with the `forceClosePosition` transaction, and submits to the mempool to close the quote forcefully as PartyB did not respond to PartyA's close request for an extended period.
2. PartyB front-run PartyA's transaction and exploit the new `settleUpnl` function to increment its nonce to 801.
3. When PartyA's transaction gets executed, since the signature's PartyB nonce is 800, while the PartyB's nonce on-chain is 801, the transaction will revert.
4. Party B could continuously grieve Party A, preventing them from being forced to close their positions.

There are several ways that PartyB can increment its own nonce via the `settleUpnl` function with no cost or minimum cost:

1. PartyA is permissionless. PartyB can create a PartyA account and open some dummy positions with its account. When PartyB needs to increment its nonce, simply settle a minimum possible PnL between these two accounts. There is no cooldown since PartyB is settling its own positions.
2. If PartyB already has existing positions, PartyB can settle a minimum possible PnL on these positions. There is no cooldown since PartyB is settling its own positions.

Impact

This impact is the same as the impact of a report in the previous Symm IO contest. Thus, the risk rating should be aligned to Medium.

PartyB can take advantage of it against PartyA, making themselves always profitable. For instance:

1. If the current price goes for PartyB, then the quote is closed, and PartyB makes a profit.
2. If the current price goes against PartyB, PartyB can front-run `forceClosePosition` and call `settleUpnl` to increase PartyA's nonces. In this way, PartyA's `forceClosePosition` will inevitably revert because the nonces are incorrect.

In addition, when PartyA cannot close its position promptly, it exposes PartyA to unnecessary market risks and potential losses and gives PartyB an unfair advantage by giving PartyB an opportunity to turn the table (e.g., loss -> profit).

PoC

No response

Mitigation

Consider implementing the following measures to mitigate the root causes:

1. Ensure that `SettlementSig` signature with `quotesSettlementsData` and/or `partyBs` array being empty are rejected within `settleUpnl` function
2. Implement some cooldown even if PartyB is settling its own positions
3. Implement a minimum PnL to be settled so that no one can attempt to abuse the settle upnl feature by only settling 1 wei or small amount of PnL

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/SYMM-IO/protocol-core/pull/58

Issue M-4: Emergency close might be blocked due to insufficient allocated balance

Source: https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest-judging/issues/50

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

Emergency close might be blocked due to insufficient PartyA's allocated balance. During urgent situations where emergency mode is activated, the positions need to be promptly closed to avoid negative events that could potentially lead to serious loss of funds (e.g. the protocol is compromised, and the attacker is planning to or has started draining funds from the protocols). However, if the emergency closure of positions is blocked or delayed due to the above-mentioned issue, it might lead to unrecoverable losses.

Root Cause

- Emergency close does not handle an edge case where PartyA has insufficient allocated balance

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

Assuming a similar scenario mentioned in the Symm IO v0.8.3 documentation.

Bob (Party A) has three open positions and zero allocated balance:

1. Position 1 with Rasa Hedger (Party B1):
    - Unrealized Profit: $300
    - Quote ID: 1

2. Position 2 with PerpsHub Hedger (Party B2):
    - Unrealized Profit: $100
    - Quote ID: 2
3. Position 3 with PerpsHub Hedger (Party B2):
    - Unrealized Loss: $250
    - Quote ID: 3

PartyB wants to emergency close Bob's Position 3 via the `emergencyClosePosition` function, which has an unrealized loss of $250. However, since Bob has zero allocated balance, the loss in Position 3 cannot cover this loss upon closing, and an attempt to emergency close will revert.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/facets/PartyBPositionActions/PartyBPositionActionsFacetImpl.sol#L99

```
File: PartyBPositionActionsFacetImpl.sol
099:     function emergencyClosePosition(uint256 quoteId, PairUpnlAndPriceSig
↪  memory upnlSig) internal {
100:         AccountStorage.Layout storage accountLayout =
↪  AccountStorage.layout();
101:         Quote storage quote = QuoteStorage.layout().quotes[quoteId];
102:         Symbol memory symbol =
↪  SymbolStorage.layout().symbols[quote.symbolId];
103:         require(
104:             GlobalAppStorage.layout().emergencyMode ||
↪  GlobalAppStorage.layout().partyBEmergencyStatus[quote.partyB] ||
↪  !symbol.isValid,
105:             "PartyBFacet: Operation not allowed. Either emergency mode must
↪  be active, party B must be in emergency status, or the symbol must be
↪  delisted"
106:         );
107:         require(quote.quoteStatus == QuoteStatus.OPENED || quote.quoteStatus
↪  == QuoteStatus.CLOSE_PENDING, "PartyBFacet: Invalid state");
108:         LibMuonPartyB.verifyPairUpnlAndPrice(upnlSig, quote.partyB,
↪  quote.partyA, quote.symbolId);
109:         uint256 filledAmount = LibQuote.quoteOpenAmount(quote);
110:         quote.quantityToClose = filledAmount;
111:         quote.requestedClosePrice = upnlSig.price;
112:         require(
113:
↪  LibAccount.partyAAvailableBalanceForLiquidation(upnlSig.upnlPartyA,
↪  accountLayout.allocatedBalances[quote.partyA], quote.partyA) >= 0,
114:             "PartyBFacet: PartyA is insolvent"
```

```
115:         );
116:         require(
117:
↪  LibAccount.partyBAvailableBalanceForLiquidation(upnlSig.upnlPartyB,
↪  quote.partyB, quote.partyA) >= 0,
118:             "PartyBFacet: PartyB should be solvent"
119:         );
120:         accountLayout.partyBNonces[quote.partyB][quote.partyA] += 1;
121:         accountLayout.partyANonces[quote.partyA] += 1;
122:         LibQuote.closeQuote(quote, filledAmount, upnlSig.price);
123:     }
```

Impact

During urgent situations where emergency mode is activated, the positions need to be promptly closed to avoid negative events that could potentially lead to serious loss of funds (e.g. the protocol is compromised, and the attacker is planning to or has started draining funds from the protocols). However, if the emergency closure of positions is blocked or delayed due to the above-mentioned issue, it might lead to unrecoverable losses.

PoC

No response

Mitigation

Consider implementing the `settleUpnl` function within the `emergencyClosePosition` function so that PartyB can proceed to settle PartyA's open positions to ensure that PartyA's allocated balance will have sufficient balance to allow the position to be emergency closed.

Discussion

MoonKnightDev

The emergency mode is designed for situations when a symbol is delisted or when a party B wants to exit. They can still use the settle unrealized profit and loss (uPnL) function separately to add to the user's allocated balance.

Issue M-5: Inconsistent in the liquidation fee leads to unfairness in liquidation process

Source: https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest-judging/issues/52

The protocol has acknowledged this issue.

Found by

xiaoming90

Summary

Reserve vault results in unfairness in the liquidation process. PartyA using force close mechanism to liquidate a position will receive more liquidation fees compared to normal liquidators as they have access to the funds in the reserve vault.

This creates unfairness to the protocol's liquidation process and to the existing liquidators, as they will receive lesser liquidation fees than Alice (PartyA). Without a fair and effective liquidation process, the protocol's solvent will be at risk, as bad positions will not be liquidated in a timely manner, and some liquidators might not want to participate in the process due to unfairness in the system.

Root Cause

- Inconsistent in the liquidation fee given to PartyA and liquidators as PartyA performs liquidation via force close mechanism has access to funds in liquidatee's reserve vault, while normal liquidators do not.

Internal pre-conditions

No response

External pre-conditions

No response

Attack Path

Assume that PartyB's reserve vault has 1000 USD. During force closing, if PartyB's account is underwater, even with the additional support from its reserve vault, PartyB's account will be liquidated, as per Lines 127-138 below.

In Lines 128 and 129, the protocol transfers all the existing funds (1000 USD) from PartyB's reserve vault to PartyB's allocated balance. Thus, before the liquidation process is executed at Line 137 below, the PartyB's allocated balance will increase by 1000 USD.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/facets/ForceActions/ForceActionsFacetImpl.sol#L127

```
File: ForceActionsFacetImpl.sol
051:     function forceClosePosition(
..SNIP..
112:         require(partyAAvailableBalance >= 0, "PartyAFacet: PartyA will be
↪  insolvent");
113:         if (partyBAvailableBalance >= 0) {
..SNIP..
118:         } else if (partyBAvailableBalance + int256(reserveAmount) >= 0) {
..SNIP..
127:         } else { // @audit-info Code block for liquidating position
128:             accountLayout.reserveVault[quote.partyB] = 0;
129:
↪  accountLayout.partyBAllocatedBalances[quote.partyB][quote.partyA] +=
↪  reserveAmount;
130:             emit SharedEvents.BalanceChangePartyB(quote.partyB,
↪  quote.partyA, reserveAmount, SharedEvents.BalanceChangeType.REALIZED_PNL_IN);
131:             int256 diff = (int256(quote.quantityToClose) *
↪  (int256(closePrice) - int256(sig.currentPrice))) / 1e18;
132:             if (quote.positionType == PositionType.LONG) {
133:                 diff = diff * -1;
134:             }
135:             isPartyBLiquidated = true;
136:             upnlPartyB = sig.upnlPartyB + diff;
137:             LibLiquidation.liquidatePartyB(quote.partyB, quote.partyA,
↪  upnlPartyB, block.timestamp);
138:         }
```

As a result, when the LibLiquidation.liquidatePartyB is executed at Line 137 above, the computed `availableBalance` will be 1000 USD smaller when the liquidation process is triggered via the `forceClosePosition` function compared to the standard liquidation process that is triggered via `LiquidationFacet.liquidatePartyB` by the liquidators, as shown below. This can be proven via the formula within partyBAvailableBalanceForLiquidation function.

```
**Normal scenario without access to reserve vault's funds**
availableBalance = partyBAllocatedBalances - lockedValue + PnL
availableBalance = 1000 - 100 - 2000 = -1100
```

```
**Force close with access the reserve vault's funds => partyBAllocatedBalances
↪  will 1000 USDC higher (1000 => 2000)**
availableBalance = partyBAllocatedBalances - lockedValue + PnL
availableBalance = 2000 - 100 - 2000 = -100
```

Assume Alice is entitled to execute the force close function against PartyB, while Bob is a normal liquidator that can only liquidate PartyB via the standard `LiquidationFacet.liquidatePartyB` function.

When Alice triggers the liquidation process, the `availableBalance` will be `-100` instead of `-1000`. Thus, when computing the liquidation fee that the liquidator is entitled to at Line 39 below, the computed liquidation fee (`remainingLf`) will be higher. This means that if Alice triggered the liquidation process, she would receive more liquidation fees compared to Bob because she has access to funds within PartyB's reserve vault. In this case, Alice is the liquidator and will receive the liquidation fee.

As a result, this creates unfairness within the system and to the existing liquidators as they (e.g., Bob) will receive lesser liquidation fees compared to Alice.

https://github.com/sherlock-audit/2024-09-symmio-v0-8-4-update-contest/blob/main/protocol-core/contracts/libraries/LibLiquidation.sol#L29

```
File: LibLiquidation.sol
23:     function liquidatePartyB(address partyB, address partyA, int256
↪  upnlPartyB, uint256 timestamp) internal {
24:         AccountStorage.Layout storage accountLayout =
↪  AccountStorage.layout();
25:         MAStorage.Layout storage maLayout = MAStorage.layout();
26:         QuoteStorage.Layout storage quoteLayout = QuoteStorage.layout();
27:
28:         // Calculate available balance for liquidation
29:         int256 availableBalance =
↪  LibAccount.partyBAvailableBalanceForLiquidation(upnlPartyB, partyB, partyA);
30:
31:         // Ensure Party B is insolvent
32:         require(availableBalance < 0, "LiquidationFacet: partyB is solvent");
..SNIP..
37:         // Determine liquidator share and remaining locked funds
38:         if (uint256(-availableBalance) <
↪  accountLayout.partyBLockedBalances[partyB][partyA].lf) {
39:             remainingLf =
↪  accountLayout.partyBLockedBalances[partyB][partyA].lf -
↪  uint256(-availableBalance);
40:             liquidatorShare = (remainingLf * maLayout.liquidatorShare) /
↪  1e18;
```

```
41:
42:             maLayout.partyBPositionLiquidatorsShare[partyB][partyA] =
43:                 (remainingLf - liquidatorShare) /
44:                 quoteLayout.partyBPositionsCount[partyB][partyA];
45:         } else {
46:             maLayout.partyBPositionLiquidatorsShare[partyB][partyA] = 0;
47:         }
```

Impact

This creates unfairness to the protocol's liquidation process and to the existing liquidators, as they will receive lesser liquidation fees than Alice. Without a fair and effective liquidation process, the protocol's solvent will be at risk, as bad positions will not be liquidated in a timely manner, and some liquidators might not want to participate in the process due to unfairness in the system.

PoC

No response

Mitigation

The liquidation process for Alice and existing liquidators should be consistent to ensure fairness. In the above case, consider either of the following solutions:

- Do not take into consideration the reserve vault's funds during liquidation within the `forceClosePosition` function; OR
- Take into consideration the reserve vault's funds within the `LiquidationFacet.liquidatePartyB` for consistency.

Discussion

MoonKnightDev

Normal liquidators can also call the settleAndForceClose functions.

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.