SHERLOCK

SHERLOCK SECURITY REVIEW FOR

PERENNIAL

Prepared For:	Perennial
Prepared By:	Sherlock
Lead Security Expert:	[WatchPug]()
Dates:	November 7th - 14th, 2022

Introduction

“Perennial is a cash-settled synthetic derivatives protocol. It allows developers to launch any synthetic market with just a few lines of code.”

This report is a follow-up security review for Perennial Protocol that was prepared by WatchPug from November 7th - 14th, 2022.

Scope

Branch: Dev (https://github.com/equilibria-xyz/perennial-mono)
Commit: b2bed03b16ceb3bce5930f54a7db1aed60dcf483 (https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages)
Change PR: https://github.com/equilibria-xyz/perennial-mono/pull/75
Contracts:

- packages/perennial/contracts/collateral/Collateral.sol
- packages/perennial/contracts/controller/Controller.sol
- packages/perennial/contracts/controller/UControllerProvider.sol
- packages/perennial/contracts/incentivizer/Incentivizer.sol
- packages/perennial/contracts/interfaces/types/PayoffDefinition.sol
- packages/perennial/contracts/interfaces/types/PendingFeeUpdates.sol
- packages/perennial/contracts/interfaces/types/Position.sol
- packages/perennial/contracts/interfaces/types/PrePosition.sol
- packages/perennial/contracts/multiinvoker/MultiInvoker.sol
- packages/perennial/contracts/product/Product.sol
- packages/perennial/contracts/product/UParamProvider.sol
- packages/perennial/contracts/product/types/accumulator/VersionedAccumulator.sol
- packages/perennial/contracts/product/types/position/AccountPosition.sol
- packages/perennial/contracts/product/types/position/VersionedPosition.sol

Protocol Info

Language: Solidity

Blockchain: Ethereum
L2s: None

Tokens used: USDC, DSU, Reward ERC20 tokens

Findings

Each issue has an assigned severity:

- Informational issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- Low issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- 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.

Total Issues

Informational	Low	Medium	High
5	0	0	0

Issue I-01 (invalid issue)

boundedFundingFee should be reloaded after _settleFeeUpdates()

Summary

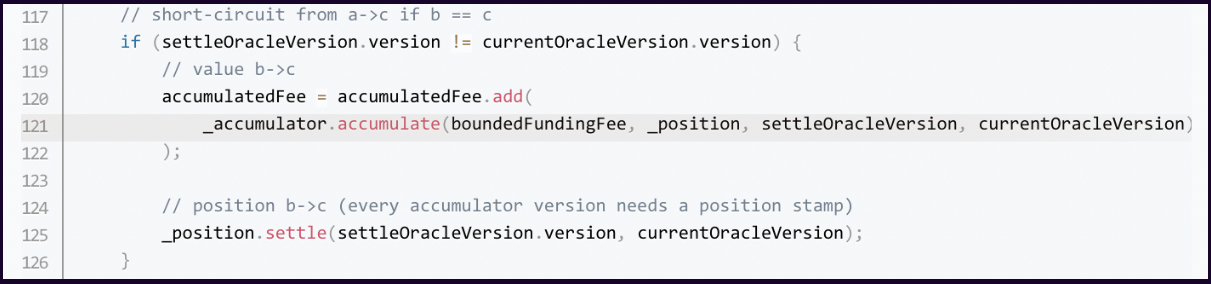
Once pending fee updates are applied with _settleFeeUpdates(), boundedFundingFee may get updated to a new value, thus before settle position b→c, the boundedFundingFee should be reloaded.

Severity

This was reported as Medium, but later confirmed to not be an issue, so it’s left as Informational

Issue Detail

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/product/Product.sol#L84-L132



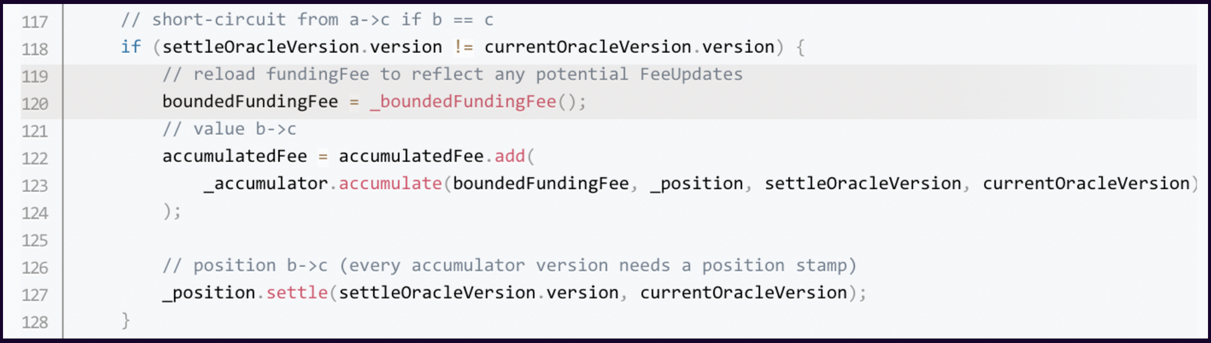
```
        // short-circuit from a->c if b == c
        if (settleOracleVersion.version != currentOracleVersion.version) {
            // value b->c
            accumulatedFee = accumulatedFee.add(
                _accumulator.accumulate(boundedFundingFee, _position, settleOracleVersion, currentOracleVersion)
            );

            // position b->c (every accumulator version needs a position stamp)
            _position.settle(settleOracleVersion.version, currentOracleVersion);
        }
```

Tool used

Manual Review

Recommendation

Change to:



```
        // short-circuit from a->c if b == c
        if (settleOracleVersion.version != currentOracleVersion.version) {
            // reload fundingFee to reflect any potential FeeUpdates
            boundedFundingFee = _boundedFundingFee();
            // value b->c
            accumulatedFee = accumulatedFee.add(
                _accumulator.accumulate(boundedFundingFee, _position, settleOracleVersion, currentOracleVersion)
            );

            // position b->c (every accumulator version needs a position stamp)
            _position.settle(settleOracleVersion.version, currentOracleVersion);
        }
```

Perennial Comment

We're unsure if this is actually an issue because fundingFee can't change in _settleFeeUpdates (code here: https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/product/UParamProvider.sol#L258-L267) so the boundedFundingFee value will stay constant.

Could you double check this? We acknowledge the comment in Product._settle a little misleading because it doesn't specify which fees might change, so we can update that.

WatchPug
Sorry for the false alarm. We just confirmed that it is indeed not an issue, as the fundingFee will not be changed the same way the other 3 fee rate settings (`makerFee`, takerFee, `positionFee`).

Perennial Comment
We'll update the comment because it is a little unspecific.

Issue I-02

Consider splitting depositTo() into two separate actions: PULL and DEPOSIT for a better action combination

Severity

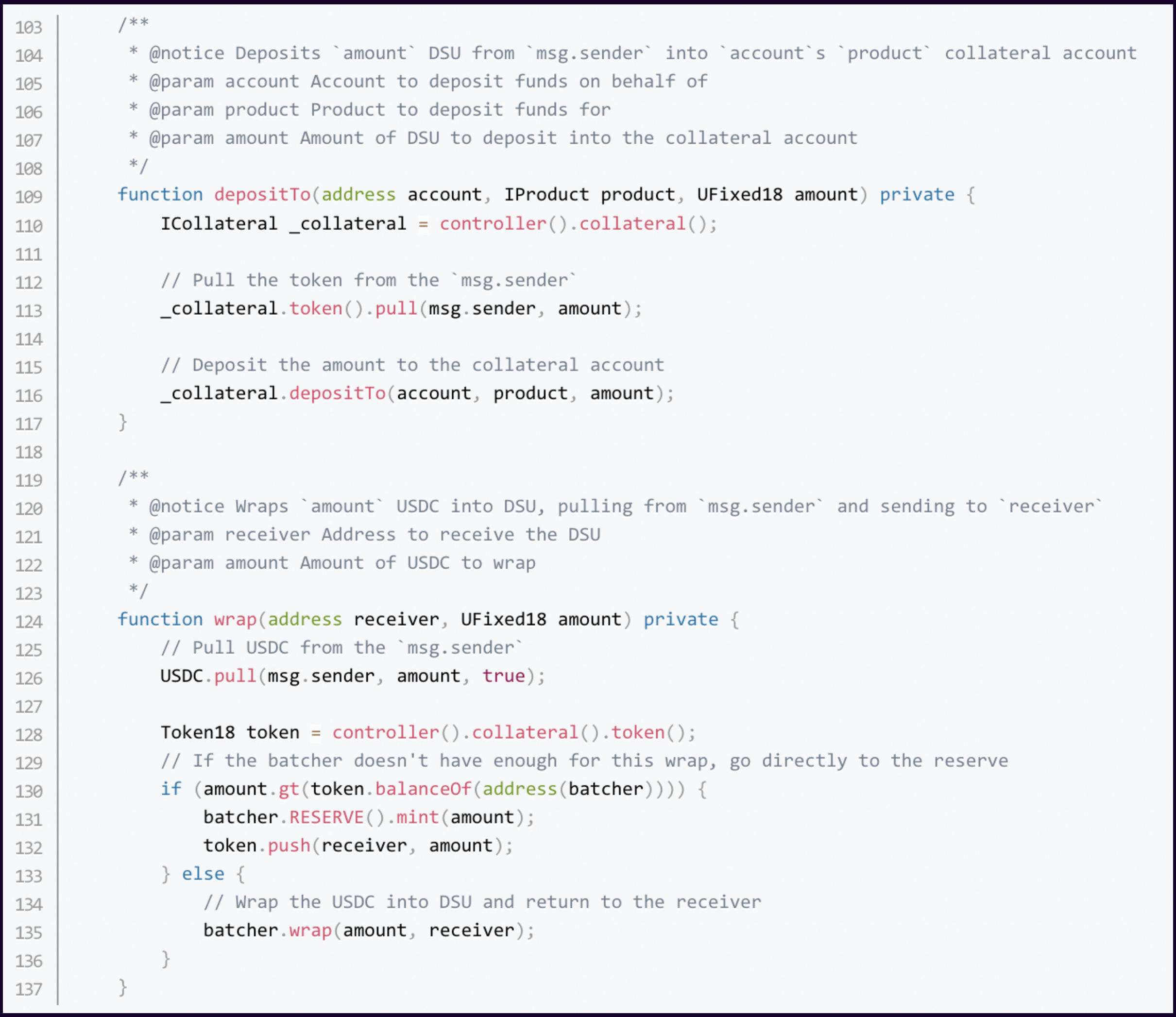
Informational

Issue Detail

Most users will need to wrap before deposit anyway, there is no need to push the DSU to the user's wallet in in wrap() and pull it from the user in depositTo() again.

It helps to save some gas for the end user, and users don't need to approve DSU anymore, which is also an enhancement in user experience.

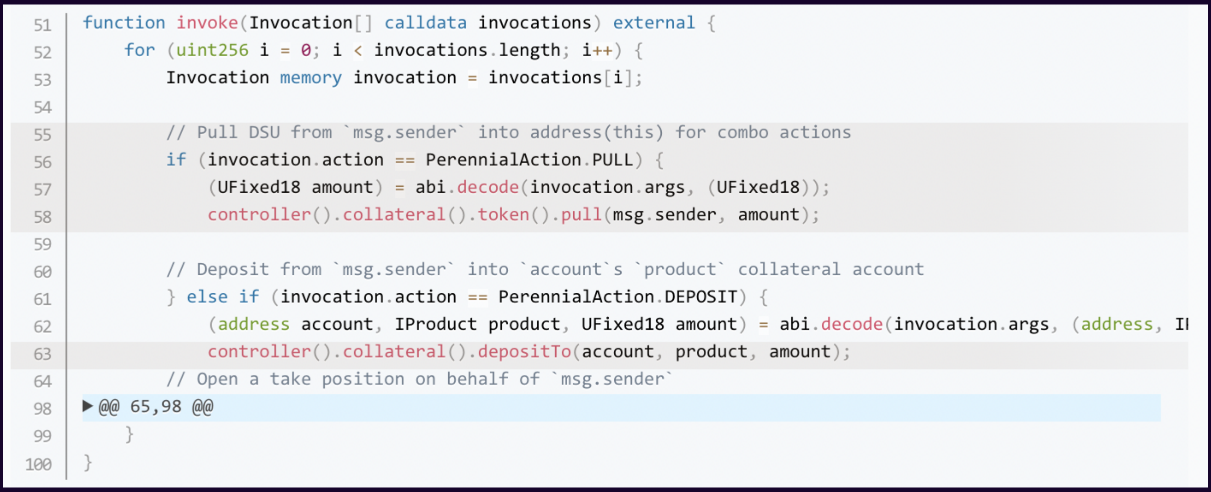
https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/multiinvoker/MultiInvoker.sol#L103-L137



```
    /**
     * @notice Deposits `amount` DSU from `msg.sender` into `account`s `product` collateral account
     * @param account Account to deposit funds on behalf of
     * @param product Product to deposit funds for
     * @param amount Amount of DSU to deposit into the collateral account
     */
    function depositTo(address account, IProduct product, UFixed18 amount) private {
        ICollateral _collateral = controller().collateral();

        // Pull the token from the `msg.sender`
        _collateral.token().pull(msg.sender, amount);

        // Deposit the amount to the collateral account
        _collateral.depositTo(account, product, amount);
    }

    /**
     * @notice Wraps `amount` USDC into DSU, pulling from `msg.sender` and sending to `receiver`
     * @param receiver Address to receive the DSU
     * @param amount Amount of USDC to wrap
     */
    function wrap(address receiver, UFixed18 amount) private {
        // Pull USDC from the `msg.sender`
        USDC.pull(msg.sender, amount, true);

        Token18 token = controller().collateral().token();
        // If the batcher doesn't have enough for this wrap, go directly to the reserve
        if (amount.gt(token.balanceOf(address(batcher)))) {
            batcher.RESERVE().mint(amount);
            token.push(receiver, amount);
        } else {
            // Wrap the USDC into DSU and return to the receiver
            batcher.wrap(amount, receiver);
        }
    }
```

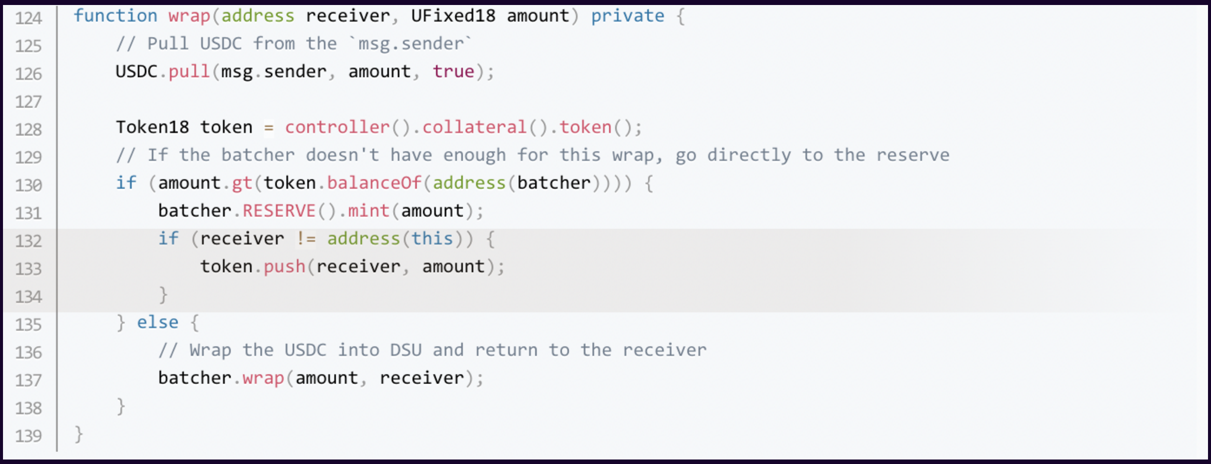
Recommendation

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/multiinvoker/MultiInvoker.sol#L51-L59



```solidity
function invoke(Invocation[] calldata invocations) external {
    for (uint256 i = 0; i < invocations.length; i++) {
        Invocation memory invocation = invocations[i];

        // Pull DSU from `msg.sender` into address(this) for combo actions
        if (invocation.action == PerennialAction.PULL) {
            (UFixed18 amount) = abi.decode(invocation.args, (UFixed18));
            controller().collateral().token().pull(msg.sender, amount);

        // Deposit from `msg.sender` into `account`s `product` collateral account
        } else if (invocation.action == PerennialAction.DEPOSIT) {
            (address account, IProduct product, UFixed18 amount) = abi.decode(invocation.args, (address, II
            controller().collateral().depositTo(account, product, amount);
        // Open a take position on behalf of `msg.sender`
▶ @@ 65,98 @@
    }
}
```

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/multiinvoker/MultiInvoker.sol#L124-L137



```solidity
function wrap(address receiver, UFixed18 amount) private {
    // Pull USDC from the `msg.sender`
    USDC.pull(msg.sender, amount, true);

    Token18 token = controller().collateral().token();
    // If the batcher doesn't have enough for this wrap, go directly to the reserve
    if (amount.gt(token.balanceOf(address(batcher)))) {
        batcher.RESERVE().mint(amount);
        if (receiver != address(this)) {
            token.push(receiver, amount);
        }
    } else {
        // Wrap the USDC into DSU and return to the receiver
        batcher.wrap(amount, receiver);
    }
}
```

Perennial Comment

Acknowledged and fixed (https://github.com/equilibria-xyz/perennial-mono/pull/81). This is not exactly the recommended fix, but achieves the same result. We introduce two new actions WRAP_AND_DEPOSIT and WITHDRAW_AND_UNWRAP. Both of these combine two actions to circumvent the extraneous push/pull, saving a significant amount of gas.

WatchPug Comment

Fix confirmed.

Issue I-03

Caching external call results can save gas

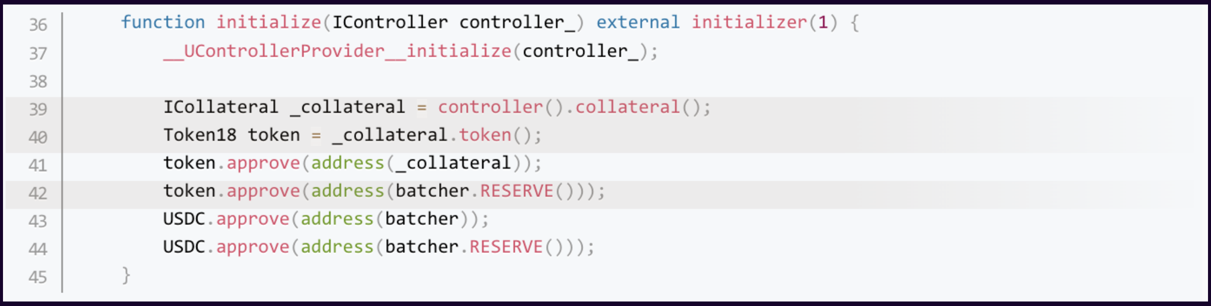
Severity

Informational

Issue Detail

Every call to an external contract costs a decent amount of gas. For optimization of gas usage, external call results should be cached if they are being used for more than one time.

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/multiinvoker/MultiInvoker.sol#L36-L45



```
    function initialize(IController controller_) external initializer(1) {
        __UControllerProvider__initialize(controller_);

        ICollateral _collateral = controller().collateral();
        Token18 token = _collateral.token();
        token.approve(address(_collateral));
        token.approve(address(batcher.RESERVE()));
        USDC.approve(address(batcher));
        USDC.approve(address(batcher.RESERVE()));
    }
```

controller().collateral(), controller().collateral().token(), and batcher.RESERVE() can be cached in storage to save the external call.

Perennial Comment

Acknowledged and partially fixed (https://github.com/equilibria-xyz/perennial-mono/pull/85). We cached the `reserve` address usage in the `initialize` function but opted to still perform call-time reads for the values during the `invoke` stage, as it is possible that these addresses can change.

WatchPug Comment

Fix confirmed.

Issue I-04

Unused imports

Severity

Informational

Issue Detail

The following source units are imported but not referenced in the contract:

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/multiinvoker/MultiInvoker.sol#L4

```
4 | import "hardhat/console.sol";
```

Recommendation

Check all imports and remove all unused/unreferenced and unnecessary imports.

Perennial Comment

Acknowledged and fixed (https://github.com/equilibria-xyz/perennial-mono/pull/82).

WatchPug Comment

Fix confirmed.

Issue I-05

Consider renaming Product.positionFee to Product.positionFeeShare() to avoid shadowed state var and misleading

Severity

Informational

Issue Detail

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/interfaces/IParamProvider.sol#L36-L37

```
    function positionFee() external view returns (UFixed18);
    function updatePositionFee(UFixed18 newPositionFee) external;
```

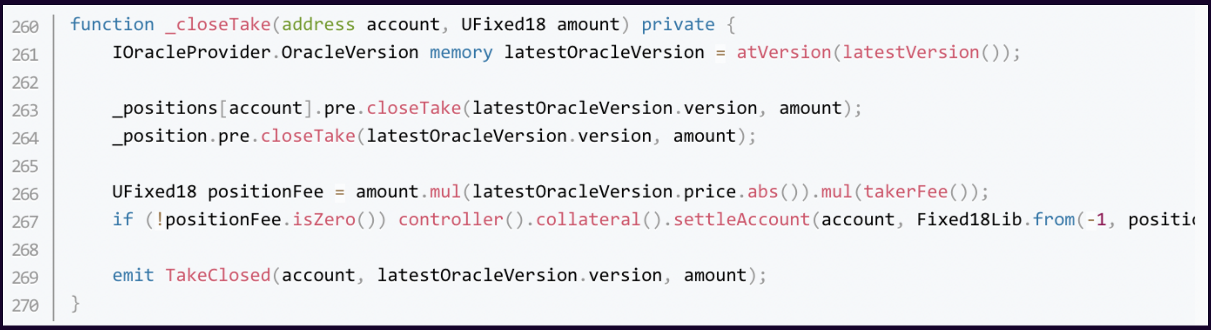
The name of Product.positionFee is quite misleading, while it's actually the share of positionFee that belongs to the product, in many contexts it can be misunderstood as the amount of position fee.

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/product/types/accumulator/VersionedAccumulator.sol#L180-L205

```
function _accumulatePositionFee(
    Position memory latestPosition,
    PrePosition memory pre,
    IOracleProvider.OracleVersion memory latestOracleVersion
) private view returns (Accumulator memory accumulatedPosition, UFixed18 fee) {
    if (pre.isEmpty()) return (accumulatedPosition, fee);

    Position memory positionFee = pre.computeFee(latestOracleVersion);
    Position memory protocolFee = positionFee.mul(_product().positionFee());
    positionFee = positionFee.sub(protocolFee);
    fee = protocolFee.sum();

    // If there are makers to distribute the taker's position fee to, distribute. Otherwise give it to the
    if (!latestPosition.maker.isZero()) {
        accumulatedPosition.maker = Fixed18Lib.from(positionFee.taker.div(latestPosition.maker));
    } else {
        fee = fee.add(positionFee.taker);
    }

    // If there are takers to distribute the maker's position fee to, distribute. Otherwise give it to the
    if (!latestPosition.taker.isZero()) {
        accumulatedPosition.taker = Fixed18Lib.from(positionFee.maker.div(latestPosition.taker));
    } else {
        fee = fee.add(positionFee.maker);
    }
}
```

And sometimes, the local variable will shadow Product.positionFee():

https://github.com/equilibria-xyz/perennial-mono/blob/b2bed03b16ceb3bce5930f54a7db1aed60dcf483/packages/perennial/contracts/product/Product.sol#L260-L270



```
function _closeTake(address account, UFixed18 amount) private {
    IOracleProvider.OracleVersion memory latestOracleVersion = atVersion(latestVersion());

    _positions[account].pre.closeTake(latestOracleVersion.version, amount);
    _position.pre.closeTake(latestOracleVersion.version, amount);

    UFixed18 positionFee = amount.mul(latestOracleVersion.price.abs()).mul(takerFee());
    if (!positionFee.isZero()) controller().collateral().settleAccount(account, Fixed18Lib.from(-1, positi

    emit TakeClosed(account, latestOracleVersion.version, amount);
}
```

Perennial Comment

Acknowledged but won't fix. We will clarify documentation around this parameter. For clarity, our thinking is as follows:

Product parameters for market economics (part of p&l directly):

- makerFee → product's fee for makers
- takerFee → product's fee for takers
- utilizationCurve → product's fee for funding

Traditional fee parameters (coordinator and protocol):

- fundingFee → coordinator's fee on funding interest
- positionFee → coordinator's fee on (maker / taker fees)
- protocolFee → protocol's fee on (funding and positions fees)

WatchPug Comment

Ok.

Additional Fix 1

Mismatch variable names in Collateral and ICollateral

Issue Detail

Fixed in https://github.com/equilibria-xyz/perennial-mono/pull/83

WatchPug Comment

Fix confirmed.

Additional Fix 2

Revert unnecessary initializer change

Issue Detail

For simplicity and ease of upgrade, we are reverting the initialize changes in Controller. https://github.com/equilibria-xyz/perennial-mono/pull/86. Our plan is to upgrade the implementation and call `updateMultiInvoker` separately

WatchPug Comment

Fix confirmed.