



SHERLOCK

SHERLOCK SECURITY REVIEW FOR



PERENNIAL

Prepared For:

Perennial

Prepared By:

Sherlock

Lead Security Experts:

[WatchPug](#), [OxRajeev](#)

Security Researchers:

[Secureum](#) Bootcamp participants

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 security review for Perennial Protocol that was prepared by Sherlock Watsons [WatchPug](#) and [OxRajeev](#), with assistance from [Secureum](#) participants, in the context of Secureum CASE (Collaborative Assessment & Security Evaluation) during 2-15 June, 2022.

A follow-up mini-audit + fix review was conducted from 22-24 August, 2022 by [WatchPug](#).

Scope

Branch: Master (<https://github.com/equilibria-xyz/perennial-mono>)

Commit: 45cb2edbf07bb03cac70e711c7f5b5445a438615

(<https://github.com/equilibria-xyz/perennial-mono/tree/45cb2edbf07bb03cac70e711c7f5b5445a438615/packages>)

Contracts:

- perennial/contracts/collateral/Collateral.sol
- perennial/contracts/collateral/types/OptimisticLedger.sol
- perennial/contracts/controller/Controller.sol
- perennial/contracts/controller/UControllerProvider.sol
- perennial/contracts/forwarder/Forwarder.sol
- perennial/contracts/incentivizer/Incentivizer.sol
- perennial/contracts/incentivizer/types/ProductManager.sol
- perennial/contracts/incentivizer/types/Program.sol
- perennial/contracts/interfaces/*.sol
- perennial/contracts/interfaces/types/*.sol
- perennial/contracts/oracle/ChainlinkOracle.sol
- perennial/contracts/oracle/types/*.sol
- perennial/contracts/product/Product.sol
- perennial/contracts/product/types/position/*.sol
- perennial/contracts/product/types/accumulator/*.sol

Branch: Master (<https://github.com/equilibria-xyz/root>)

Commit: 9378a9538a495a04283c3420d697b0d7728b7ace

(<https://github.com/equilibria-xyz/root/tree/9378a9538a495a04283c3420d697b0d7728b7ace>)

Contracts:

- root/contracts/control/unstructured/*.sol
- root/contracts/curve/*.sol
- root/contracts/curve/immutable/*.sol
- root/contracts/curve/unstructured/*.sol



SHERLOCK

- root/contracts/curve/types/*.sol
- root/contracts/number/types/*.sol
- root/contracts/token/types/*.sol
- root/contracts/storage/UStorage.sol

After the initial review, the following contracts/folders were removed from scope:

- perennial/contracts/lens/PerennialLens.sol
- perennial-provider/contracts/oracle/*.sol
- perennial-provider/contracts/product/*.sol

Mini-Audit Scope

The main scope of this mini-audit was the code changes made for M-03 and M-04 issues in the initial audit report.

The diff is found here: <https://github.com/equilibria-xyz/perennial-mono/pull/35>

Additionally, an audit of the [ReservoirFeedOracle.sol](#) contract was also conducted.

No major issues were identified during the mini-audit, and only issues I-09 and I-10 are related to the mini-audit.

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.



SHERLOCK

- High issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
10	15	4	3



Issue H-01

Bank run is possible when the product is insolvent.

Summary

When a product becomes insolvent, the last user(s) will not be able to withdraw their full balance.

Severity

High

Vulnerability Detail

When one or more accounts go bankrupt, the bad debt will be accumulated in the product as the shortfall. When $\text{shortfall} > 0$ it means the product is now insolvent i.e. the total assets (product.total) is lower than the total liabilities (sum of all takers and makers' product accounts).

If a protocol becomes insolvent, the protocol owner (or a backer e.g. re-insurance fund as commented in the code) is expected to re-capitalize by resolving the shortfall. But there is nothing in the protocol to guarantee the protocol's shortfall to be resolved in time for all users' withdrawals to be made whole. Furthermore, when a user withdraws, the product's shortfall is not taken into consideration.

Impact

Early withdrawals will be able to close their position without being impacted by the insolvency but the protocol may not have sufficient funds for later withdrawals of the insolvent product. Therefore early users can be made whole as there are still enough assets for them to exit, but late users or at least the last user will not be able to withdraw their entire balance.

This will make the users of the insolvent product rush to withdraw their balances, i.e. a bank run scenario.

Code Snippet

[resolveShortfall](#) [withdrawTo](#) [debitAccount](#) [settleAccount](#)

Tool used

Manual Review

Recommendation

Withdrawals in insolvent products should account for any shortfall in proportion to their account's open positions.

One possible solution is to introduce a mapping(address \Rightarrow UFixed18) frozenBalance; to the OptimisticLedger of the product, and the frozenBalance will be settled for the shortfall accumulated and resolved in the account settlement flywheel. We believe it's



SHERLOCK

better if the shortfall accumulated and resolved is based on per unit of open position instead of per unit of collateral, because a user with some balance in the product but no open position should not bear the negative impact from any shortfall caused by bankrupt accounts.

Perennial Comment

Acknowledged.

Working as expected. Perennial only increments a shortfall when a position is settled (via user closing or force closing by liquidation), so in order to calculate the total shortfall for a product all positions would need to be settled. Essentially, the shortfall value is a lower bound of the shortfall, but can always increase as more positions are settled. This isn't feasible on-chain, so we cannot socialize the shortfall based on user positions.

Sherlock Comment

Ok.



SHERLOCK

Issue H-02

Missing accumulatedFee debit on the paying side may make the product insolvent.

Summary

The platform/product fee (accumulatedFee) component of the funding fee (accumulatedFunding) is missing a debit on the paying side.

Severity

High

Vulnerability Detail

The current implementation does not debit the platform/product fee accumulatedFee on the paying side. Given:

Example scenario: Alice is the only Maker of the product, holding a position of 20 with a collateral of 100; Bob is the only Taker, holding a position of 10 with a collateral of 100; rateAccumulated = 10%; The price is 10; product.total = 200; FundingFeeRate is 0.1 (10%);

In product.settle():

```
takerNotional = 10 * 10 == 100;
fundingAccumulated = 100 * 10% == 10;
accumulatedFee = 10 * 0.1 == 1;
fundingIncludingFee = +(10-1) == 9;
```

Total accumulatedFunding on the maker side is 9; total accumulatedFunding on the taker side is -9;

As a result:

```
product.total = 199;
Alice's account = 100+9 == 109;
Bob's account = 100-9 == 91;
```

The product is now insolvent as the total liabilities is $109 + 91 == 200$, but the total assets is only 199.

Impact

The product will become insolvent right after the product is settled because the accumulatedFee will be deducted from the total assets (product.total) but the total liabilities (sum of all takers and makers' product accounts) remain unchanged. In other words, a fee is taken from the product but no one is paying for that.

Code Snippet

[_accumulateFunding](#)



SHERLOCK

Tool used

Manual Review

Recommendation

Add below to the end of function `_accumulateFunding`:

```
    if (fundingAccumulated.sign() == -1) {  
        accumulatedFunding.maker =  
        accumulatedFunding.maker.sub(accumulatedFee);  
    }  
    else {accumulatedFunding.taker =  
        accumulatedFunding.taker.sub(accumulatedFee);}
```

Perennial Comment

Acknowledged and fixed.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/33>

Sherlock Comment

Confirmed.



SHERLOCK

Issue H-03

Unconventional PnL settlement can cause unexpected user liquidation.

Summary

P&L is settled directly to the user's account and can be triggered by anyone which could lead to unexpected liquidations.

Severity

High

Vulnerability Detail

As per the documentation: *"P&L is settled every oracle version globally on a per-position basis. Account-level P&L is synced when a user makes any position-related updates to their account (i.e. opening or closing a position)."*

In conventional perpetual futures markets, PnL will only be settled when positions are closed. For open positions, "unrealized" PnLs are tracked and accounted for, but only realized/settled upon closing of positions. However, the current implementation of Perennial protocol settles PnL directly to the user's account every time `settleAccountInternal` is called (by anyone), without differentiating between unrealized and realized PnL.

Furthermore, `settleAccount` flow is permissionless which means that anyone can trigger PnL settlement for anyone else. This combination could lead to unexpected liquidations.

Example Scenario: Let's say that the product is ETH3x. Maintenance is 0.5 (50%) and the current price of the product is \$3000 ($3 * 1000$).

1. Alice deposits \$2000 into her collateral account
2. Alice opens a maker (short) position of 1 ETH3x
3. As an experienced perpetual futures trader, Alice sets an alert for the liquidation price of \$4000 for ETH3x or ~\$1333 for ETH
4. After a few rounds, the price of ETH3x rises to \$3600 (ETH price: \$1200)
5. Mallory calls `settleAccount(Alice)` and the PnL of Alice's open position \$-600 was settled to Alice's collateral account, and the updated balance is \$1400
6. Alice's position gets liquidated due to insufficient maintenance collateral \$1800 ($\$3600 * 0.5$) which is $>$ Alice's balance of \$1400
7. After a few rounds, the price of ETH3x plunges to \$2400
8. Alice returns to check her earnings prepared to close the position and collect the profit when she discovers that her position had already been liquidated
9. Alice's liquidation alert was never triggered but Alice's position has been liquidated

Impact

Users get unexpectedly liquidated if they assume that Perennial protocol behaves similar to conventional perpetual futures market.



SHERLOCK

Code Snippet

[Documentation](#) [settleAccountInternal](#) [settleAccount](#)

Tool used

Manual Review

Recommendation

Consider a combination of introducing the concept of unrealized PnL in the protocol and/or limiting unrestricted access to the settlement flow.

Perennial Comment

Acknowledged but won't fix. We have found a static function to determine the liquidation price given a position, and will display/adequately warn users on the frontend.

$$LiquidationPrice = \frac{InitialCollateral + Position * InitialPrice}{Position * (MaintenanceRatio + 1)}$$

We will additionally warn (or fully prevent) users on the frontend when their maintenance leaves no buffer for price movements (i.e. they would be immediately liquidated) as is consistent with other defi-native leveraged protocols.

Sherlock Comment

Ok.



SHERLOCK

Issue M-01

Griefing attack by front-running to prevent user withdrawals.

Summary

An attacker can revert a user's `withdrawTo` by front-running to deposit a dust amount of collateral in the user's account.

Severity

Medium

Vulnerability Detail

The `collateralInvariant` modifier is used to check whether the account is either empty or above the minimum collateral at the end of each `withdrawTo`. Meanwhile, `depositTo` allows deposits to an arbitrary address with no minimum deposit amount required. This opens a griefing attack vector where an attacker can frontrun any user's `withdrawTo` to deposit a dust amount of collateral for the victim and make the victim's `withdrawTo` revert because the account is not empty after the withdrawal and the amount of remaining funds cannot meet the minimum collateral.

Example scenario:

1. Alice has 1M collateral
2. Alice submits `withdrawTo` 1M (all the collateral)
3. Attacker frontruns Alice's transaction to deposit 1 Wei in her account
4. Alice's withdrawal will revert with `CollateralUnderLimitError`

The attacker can repeatedly front-run Alice's `withdrawal` and effectively prevent Alice from withdrawing her collateral.

Impact

This causes a denial-of-service to users' withdrawals.

Code Snippet

[`withdrawTo`](#) [`collateralInvariant`](#) [`depositTo`](#)

Tool used

Manual Review

Recommendation

Consider requiring the deposit amount to be greater than `controller.minCollateral` when `depositTo` account `!= msg.sender`.

Perennial Comment

Acknowledged and fixed.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/18>



SHERLOCK

Sherlock Comment

Confirmed.



SHERLOCK

Issue M-02

Protocol owner is a single point of failure.

Summary

Protocol owner can arbitrarily and unilaterally create coordinators/products, update product/incentizer/collateral addresses and change the various fees and minimum collateral requirements used by all the products and users of the protocol. This presents a critical single point of failure.

Severity

Medium

Vulnerability Detail

If a protocol owner becomes malicious or compromised, the entire protocol is immediately at risk for all existing/future products and users. While the updation of critical parameters emit events, that only lets product owners and users react after the fact because these changes are not time-delayed.

Impact

A scenario that affects future products/users is the updation of product/incentivizer/collateral contract addresses to malicious ones and creation of malicious/biased products/coordinators.

A DoS scenario, for existing products/users, is the protocol owner increasing the minimum collateral to a high enough value which prevents deposits and withdrawals for certain accounts/products. The owner may increase the protocol fee split to 100% taking away the incentives for existing product owners. The owner may decrease the liquidation fee to a very low value making it uneconomical to liquidate under-collateralized accounts.

Code Snippet

[createCoordinator](#) [createProduct](#) [updateCollateral](#) [updateIncentivizer](#)
[updateProductBeacon](#) [updateProtocolFee](#) [updateLiquidationFee](#) [updateMinCollateral](#)

Tool used

Manual Review

Recommendation

While it has been communicated that the initial protocol ownership on product creation and other critical protocol aspects is part of the guarded launch strategy, the protocol owner should nevertheless be a reasonable threshold multisig (e.g. 4/7, 5/9) with diverse owners and (cold/hardware) wallets until it is backed by token-holder



SHERLOCK

governance, i.e., it should certainly never be an EOA. The highest possible operational security measures should be taken for all multisig owners and wallets.

Perennial Comment

Acknowledged, Perennial will eventually transition to token-backed governance. Until then we will use a 4/7 or stronger multisig.

Sherlock Comment

Ok.



SHERLOCK

Issue M-03

Product owner can collude to force liquidations.

Summary

A malicious/compromised product owner can collude with liquidators to force liquidations by suddenly increasing maintenance.

Severity

Medium

Vulnerability Detail

Product owners are semi-trusted in the protocol (per protocol design) because they can provide custom code and arbitrarily change their product parameters. While this is recognized in the protocol to implement safeguards against fee ranges of funding, maker and taker, the maintenance and maker limits are not protected against malicious/compromised owners of products whose providers allow modifications of product parameters.

Impact

A malicious/compromised product owner can collude with liquidators to force liquidations by suddenly increasing maintenance requirements for their product. Given that this change can be effected immediately without any time-delay, users may not have the time to react to increase their collateral deposits and prevent liquidations of their positions.

Code Snippet

[ProductProvider](#) [updateMaintenance](#)

Tool used

Manual Review

Recommendation

Consider a design choice where product maintenance can only be increased in certain predefined amounts if it does not lead to immediate liquidations. Add a timelock to this function so users have enough time to react if they are susceptible to liquidations because of this change.

Perennial Comment

Acknowledged, we have made extensive changes to the way parameters and oracles work. Namely, we are moving all parameters (maintenance, utilization curve, funding/maker/taker fees, maker limit) from the ProductProvider to the Product directly, so we can set constraints on them. We have also moved the Oracle to the Product and made it immutable - the payoff function is now only optionally a call (via staticCall) to an external contract. Please see the PR for a detailed description of



SHERLOCK

changes. We will only show high quality products on the main front-end that we operate (i.e. products owned by the protocol owner, or specific teams with secure ownership setups).

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/35>

Sherlock Comment

These changes were reviewed separately as part of the follow-up mini-audit from August 22nd-24th.

Overall, Sherlock confirmed the fixes and only flagged two Informational issues with respect to the fixes (I-09 and I-10).



SHERLOCK

Issue M-04

Malicious/compromised product owners can manipulate users.

Summary

Malicious/compromised product owners can create products with payoff functions, parameters and oracles which deceive users taking a position in those products to benefit product owners or a subset of product users.

Severity

Medium

Vulnerability Detail

ProductProvider.sol comments that "Product providers are semi-untrusted as they contain custom code from the product owners." While the protocol owner technically has an opportunity (as part of the initial guarded launch) to check the implementation/effects of custom code and allowlist a provider adding a product via createProduct, the extent of these due-diligence checks possible and enforced is not clear beyond the dynamic safe range checks for fundingFee, makerFee and takerFee implemented in ProductProviderLib.

The provider can provide custom code including the provision of a custom oracle (which may even be changed later) which is a risk. Moreover, a misbehaving product can only be paused by its product owner or, in the case of a malicious/compromised product owner, by the protocol owner only by pausing the entire protocol.

Impact

A malicious/compromised product owner can deceive users via rigged payoff functions, manipulated values of maintenance/fees/limit or use of untrustworthy oracles.

A malicious product that misbehaves to deceive users under certain conditions may be hard to detect apriori and once deployed within the protocol may require the protocol owner to pause the entire protocol to stop it from cheating its users and damaging protocol reputation.

The risk of interacting with potentially malicious untrusted products is taken by product users who typically are not informed enough to make a judgement on a product's trustworthiness without the availability of any data, tools or techniques to help them with this decision-making.

Code Snippet

[ProductProvider](#) [createProduct](#) [updateOracle](#) [ShortEther](#)

Tool used



SHERLOCK

Manual Review

Recommendation

Evaluate providing data, interfaces, tools or techniques to help users make an informed judgement on a product's trustworthiness before engaging.

Perennial Comment

Acknowledged, we have made extensive changes to the way parameters and oracles work. Namely, we are moving all parameters (maintenance, utilization curve, funding/maker/taker fees, maker limit) from the ProductProvider to the Product directly, so we can set constraints on them. We have also moved the Oracle to the Product and made it immutable - the payoff function is now only optionally a call (via staticCall) to an external contract. Please see the PR for a detailed description of changes. We will only show high quality products on the main front-end that we operate (i.e. products owned by the protocol owner, or specific teams with secure ownership setups).

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/35>

Sherlock Comment

These changes were reviewed separately as part of the follow-up mini-audit from August 22nd-24th.

Overall, Sherlock confirmed the fixes and only flagged two Informational issues with respect to the fixes (I-09 and I-10).



SHERLOCK

Issue L-01

Missing Zero-address Validation.

Summary

Lack of zero-address validation on address parameters will lead to reverts and may force contract redeployments in the protocol.

Severity

Low

Vulnerability Detail

Many functions externally accessible by users and owners lack zero-address validation on address parameters.

Impact

This may lead to transaction reverts, waste gas, require resubmission of transactions and may even force contract redeployments in certain cases within the protocol.

Code Snippet

[updateCollateral](#) [updateIncentivizer](#) [updateProductBeacon](#) [updateCoordinatorTreasury](#)
[updateCoordinatorPauser](#) [Forwarder.constructor](#)

Tool used

Manual Review

Recommendation

Add explicit zero-address validation on input parameters of address type.

Perennial Comment

Acknowledged and fixed.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/19>

Sherlock Comment

Confirmed.



SHERLOCK

Issue L-02

Time-delayed change of critical parameters is absent.

Summary

Change of critical parameters should be enforced only after a time delay.

Severity

Low

Vulnerability Detail

When critical parameters of systems need to be changed, it is required to broadcast the change via event emission and recommended to enforce the changes after a time-delay. This is to allow system users to be aware of such critical changes and give them an opportunity to exit or adjust their engagement with the system accordingly.

None of the onlyOwner functions that change critical protocol addresses/parameters have a timelock for a time-delayed change to alert: (1) users and give them a chance to engage/exit protocol if they are not agreeable to the changes (2) team in case of compromised owner(s) and give them a chance to perform incident response.

Impact

Users may be surprised when critical parameters are changed or incentivizer programs completed early without notice. Furthermore, it can erode users' trust since they can't be sure the protocol rules won't be changed later on.

Compromised owner keys may be used to change protocol addresses/parameters to benefit attackers. Without a time-delay, authorised owners have no time for any planned incident response.

Code Snippet

[updateCollateral](#) [updateIncentivizer](#) [updateProductBeacon](#) [updateProtocolFee](#)
[updateMinFundingFee](#) [updateLiquidationFee](#) [updateIncentivizationFee](#)
[updateMinCollateral](#) [updateProgramsPerProduct](#) [complete](#)

Tool used

Manual Review

Recommendation

All access-controlled functions that set/change critical addresses/parameters in these contracts should apply a timelock. Consider evaluating the use of OpenZeppelin's TimelockController.

Perennial Comment



SHERLOCK

These parameters will be behind a Timelock (even in the multisig stage).

Sherlock Comment

Ok.



SHERLOCK

Issue L-03

Missing sanity/threshold checks.

Summary

Sanity/threshold checks (depending on their types) on input parameters are missing.

Severity

Low

Vulnerability Detail

Functions that update critical protocol parameters are missing sanity/threshold validation on some parameters.

Examples include:

- Missing sanity/threshold check on newMinCollateral of updateMinCollateral which allows it to be accidentally set to 0 (default initial value) or a very high value.
- Missing sanity/threshold check on amount field of programInfo in ProgramInfoLib.validate which may immediately revert if amount is less than the incentivizationFee that is deducted upon program creation.

Impact

Parameters may accidentally be initialized with invalid values in the context of the protocol or in relation to other parameters. This may lead to incorrect accounting and protocol malfunction.

Code Snippet

[updateMinCollateral](#) [validate](#)

Tool used

Manual Review

Recommendation

Add explicit sanity/threshold validation to check that input parameters fall within range or have values as expected in the context of the protocol or in relation to other parameters.

Perennial Comment

Won't fix - 0 is a valid value for minCollateral, and a very high minCollateral value can be used to put products into a close-position only state. incentivizationFee is a percentage (subtracted from UFixed18.ONE)

Sherlock Comment



SHERLOCK

Ok.



SHERLOCK

Issue L-04

Coordinator ownership cannot be removed.

Summary

A compromised or malicious coordinator owner cannot be removed.

Severity

Low

Vulnerability Detail

A malicious/compromised coordinator cannot be removed by protocol owner and neither can it be specifically paused (only coordinator pauser/owner can pause its products). There is no way for protocol owner to override and forcibly remove coordinator ownership.

Impact

If a malicious/compromised coordinator provides misleading/malicious products that could harm the protocol funds and/or reputation then such an emergency situation can only be addressed by pausing the entire protocol.

Code Snippet

[Coordinator ownership](#)

Tool used

Manual Review

Recommendation

Consider mechanisms for protocol owner to remove coordinators and/or pause specific coordinators/products implemented with a timelock. This may be triggered by token-holder protocol governance (if/when implemented) to reduce the centralized power of the protocol owner.

Perennial Comment

Won't fix - as designed, Perennial governance should not be able to forcefully take over control of products (similar to how Uniswap governance can't take over a pool).

Sherlock Comment

Ok.



SHERLOCK

Issue L-05

Missing equivalence checks in setters.

Summary

Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract.

Severity

Low

Vulnerability Detail

Setter functions are missing checks to validate if the new value being set is the same as the current value already set in the contract. Such checks will showcase mismatches between on-chain and off-chain states.

Impact

This may hinder detecting discrepancies between on-chain and off-chain states leading to flawed assumptions of on-chain state and protocol behavior.

Code Snippet

[updateCoordinatorPaused](#)

Tool used

Manual Review

Recommendation

Add equivalence checks to validate (and revert) if the new value being set is the same as the current value already set in the contract.

Perennial Comment

Won't fix, we'd prefer to not safety check users in this case.

Sherlock Comment

Ok.



SHERLOCK

Issue L-06

Creating a product with a non-existent coordinatorId is allowed.

Summary

The protocol owner may accidentally call createProduct with a non-existent coordinatorId which prevents it from being paused and could lead to other unexpected behavior.

Severity

Low

Vulnerability Detail

If the protocol owner accidentally calls createProduct with a non-existent coordinatorId, this is not validated against _coordinators.length. The product is created with coordinatorFor assigned with the non-existent coordinatorId.

Impact

A product associated with a non-existent coordinatorId is identified as an existing product by isProduct and therefore is functional within the context of the protocol but cannot be paused because the product's owner is evaluated as being the zero address. The owner, treasury or pauser addresses are all treated as zero addresses and cannot be updated. This cannot be reversed and will require the pausing of the entire protocol to prevent that product from being used.

Code Snippet

[createProduct](#)

Tool used

Manual Review

Recommendation

Validate that the coordinatorId used is a valid one by checking that it is less than _coordinators.length.

Perennial Comment

Will fix by changing onlyOwner(0) to onlyOwner(coordinatorId).

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/20>

Sherlock Comment

Confirmed.



SHERLOCK

Issue L-07

Missing validation and events in init functions.

Summary

None of the init functions perform zero-address validation on address parameters or emit events.

Severity

Low

Vulnerability Detail

None of the init functions perform zero-address validation on address parameters or emit init-specific events. They all however have the initializer modifier (from UInitializable) so that they can be called only once.

Impact

Accidental use of incorrect parameters will require contract redeployment. Offchain monitoring of calls to these critical functions is not possible.

Code Snippet

[Controller.initialize](#) [Collateral.initialize](#) [Incentiver.initialize](#) [Product.initialize](#)

Tool used

Manual Review

Recommendation

It is recommended to perform validation of input parameters and emit events.

Perennial Comment

Most of these should already have events by calling the underlying update* function, will double check these and perform 0 address validation as part of L01.

Fix PR: <https://github.com/equilibria-xyz/root/pull/33>

Sherlock Comment

Confirmed.



SHERLOCK

Issue L-08

Missing isProduct modifier may lead to unexpected behavior.

Summary

Few externally accessible functions that take a product parameter are missing the isProduct modifier which checks if the provided product is a valid one in the context of the protocol.

Severity

Low

Vulnerability Detail

Externally accessible functions in Collateral and Incentivizer that take a product parameter use the isProduct modifier (defined in UControllerProvider) to check if the provided product is a valid one in the context of the protocol. There are a few functions which do not use this modifier on the provided product parameter.

Impact

This may lead to unexpected behavior (e.g. reverts without an appropriate error) or user confusion when a non-existent product is used accidentally as a parameter for such functions.

Code Snippet

[collateral](#) [collateral](#) [shortfall](#) [liquidatable](#) [liquidatableNext](#) [resolveShortfall](#)

Tool used

Manual Review

Recommendation

Consider adding isProduct modifier to all externally accessible functions that take a product parameter.

Perennial Comment

Partial fix for resolveShortfall, other functions weren't worth the gas increase for an external call.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/21>

Sherlock Comment

Confirmed.



SHERLOCK

Issue L-09

Missing nonReentrant modifier may lead to unexpected behavior.

Summary

Function resolveShortfall is missing the nonReentrant modifier.

Severity

Low

Vulnerability Detail

Function resolveShortfall which is documented as “*This hook can be used by the product owner or an insurance fund to re-capitalize an insolvent market*”, modifies contract state, transfers tokens to the contract and can be called by anyone. However, this is missing the nonReentrant modifier unlike other similar functions in the codebase.

Impact

An exploitable reentrancy is likely not possible given the use of the CEI pattern.

Code Snippet

[resolveShortfall](#)

Tool used

Manual Review

Recommendation

Add the nonReentrant modifier for defensive programming (despite the use of the CEI pattern) and to be consistent with other similar functions in the codebase.

Perennial Comment

Won't fix, the only external call here is DSU which is trusted.

Sherlock Comment

Ok.



SHERLOCK

Issue L-10

Event spamming may grief system.

Summary

claimFee can be called by anyone leading to event spamming.

Severity

Low

Vulnerability Detail

claimFee is expected to be called only by protocolTreasury or one of the productTreasury addresses to claim their respective fees. However, there is no access control on the msg.sender and the code will work for any msg.sender (accounting and token transfer of 0 amount) to finally emit a FeeClaim event.

Impact

The emission of an event in a non-access-controlled public function allows grieving the system via event spamming, which could confuse/overwhelm off-chain monitoring tools about fees being claimed when they are actually not i.e. when msg.sender is not protocolTreasury or one of the productTreasury addresses.

Code Snippet

[claimFee](#)

Tool used

Manual Review

Recommendation

Consider adding access control to allow claimFee to be called only by protocolTreasury or one of the productTreasury addresses.

Perennial Comment

Won't fix, we'd prefer to not safety check users in this case.

Sherlock Comment

Ok.



SHERLOCK

Issue L-11

Missing equality check on array lengths of parameters.

Summary

The claim function of incentivizer takes two parameters of array types whose lengths should be the same but this input validation is missing.

Severity

Low

Vulnerability Detail

The claim function of incentivizer takes two parameters of array types products and programIds to claim rewards for programs corresponding to each product. The lengths of these two arrays should be the same but this input validation is missing.

Impact

Mismatched array lengths could lead to reverts or unexpected behavior.

Code Snippet

[claim](#)

Tool used

Manual Review

Recommendation

Perform input validation to check that the lengths of two parameter arrays are equal.

Perennial Comment

Fixed.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/23>

Sherlock Comment

Ok.



SHERLOCK

Issue L-12

Missing isProgram modifier may lead to unexpected behavior.

Summary

Externally accessible functions that take product and programId parameters are missing the isProgram modifier which checks if the provided programId is a valid program corresponding to the provided product.

Severity

Low

Vulnerability Detail

Externally accessible functions in Incentivizer that take product and programId parameters should use the isProgram modifier to check if the provided programId is a valid program corresponding to the provided product. The getter functions do not use this modifier on their parameters.

Impact

This may lead to unexpected behavior (e.g. reverts without an appropriate error) or user confusion on returned values when non-existent product/programId are used accidentally as parameters for such functions.

Code Snippet

[programInfos](#) [unclaimed](#) [available](#) [versionStarted](#) [versionComplete](#) [owner](#)

Tool used

Manual Review

Recommendation

Consider adding isProgram modifier to all externally accessible functions that take product and programId parameters so that IncentivizerInvalidProgramError is thrown on invalid values.

Perennial Comment

Won't fix, similar to L08 the gas tradeoffs aren't worth the slight usability improvements for view functions.

Sherlock Comment

Ok.



SHERLOCK

Issue L-13

Incentivizer program is incompatible with rebasing/deflationary/inflationary reward tokens.

Summary

Incentivizer program allows arbitrary ERC20 reward tokens but does not handle rebasing/deflationary/inflationary tokens.

Severity

Low

Vulnerability Detail

Protocol allows arbitrary contracts to be used as Incentivizer rewards tokens. However, the reward logic does not appear to support rebasing/deflationary/inflationary tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred before and after the actual transfer to infer any fees/interest. These seem to be absent.

Impact

This will lead to miscalculations between protocol accounting and the actual amounts transferred when rebasing/deflationary/inflationary reward tokens are used.

Code Snippet

[ProgramInfo](#) [create](#) [_claimProgram](#) [Code4rena](#) [Finding](#)

Tool used

Manual Review

Recommendation

Either:

- 1) Implement allowlist of reward tokens while vetting for any rebasing/inflation/deflation tokens, or
- 2) Add support in contracts for such tokens before accepting arbitrary tokens, or
- 3) Document the supported reward token types

Perennial Comment

This restriction is mentioned in docs:

<https://docs.perennial.finance/developers/launching-a-new-product/launching-an-inc>
entive-program, we can also add this comment to the Solidity code.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/24>

Sherlock Comment



SHERLOCK

Confirmed.



SHERLOCK

Issue L-14

Token approvals may fail.

Summary

Token libraries implement approve as a wrapper over (the deprecated) safeApprove which will fail under some conditions.

Severity

Low

Vulnerability Detail

OpenZeppelin's SafeErc20 safeApprove, as documented, is meant to only be called when setting an initial allowance or when resetting it to zero. Its usage has also been deprecated because it is susceptible to the same race condition as ERC20 approve.

Impact

This will lead to approval failures when changing allowances.

Code Snippet

[Token6Lib.approve](#) [Token18Lib.approve](#) [TokenOrEther18Lib.approve](#)
[SafeERC20.safeApprove](#)

Tool used

Manual Review

Recommendation

Either:

- 1) Implement using safeIncreaseAllowance/safeDecreaseAllowance, or
- 2) Document the expected usage

Perennial Comment

Will fix by documenting.

Fix PR: <https://github.com/equilibria-xyz/root/pull/34>

Sherlock Comment

Confirmed.



SHERLOCK

Issue L-15

Protocol is susceptible to stale oracle data feeds.

Summary

Chainlink's latestRoundData and getRoundData might return stale/incomplete results which get used in the protocol.

Severity

Low

Vulnerability Detail

Chainlink's latestRoundData and getRoundData may return stale or incomplete round data. ChainlinkRegistryLib.getLatestRound uses the return values of latestRoundData and ChainlinkRegistryLib.getRound uses the return values of getRoundData without validating for round staleness or completeness because they ignore the answeredInRound value and do not validate the updatedAt timestamp.

Impact

Prices from historical or incomplete Chainlink rounds may allow stale data to be used in the protocol.

Given that settlements are round-based, the impact of stale oracle prices is delayed updates in future rounds. The Perennial team has been aware of this issue/impact for a while and has reportedly investigated this aspect thoroughly, even in discussions with the Chainlink team itself.

Code Snippet

[getLatestRound](#) [Code4rena Finding](#) [Chainlink Documentation](#)

Tool used

Manual Review

Recommendation

Consider adding validation checks on return values of latestRoundData and getRoundData by checking that answeredInRound \geq roundID and updatedAt \neq 0.

Perennial Comment

"This is the response we got from Chainlink when we chatted with them: LatestRoundData will not change once it is written to the chain. Only new rounds update it."

Perennial Follow-Up Comment



SHERLOCK

See comment above, our implementation relies on the fact (per Chainlink team comments) that the feed registry will not push stale data (i.e. rounds where `answeredInRound != roundID`). Can the auditing team confirm this by speaking with Chainlink? Our understanding is that stale rounds were only possible in a prior version of Chainlink.

Sherlock Comment

Confirmed.



SHERLOCK

Issue I-01

Push is susceptible to reentrancy risk for Ether transfers.

Summary

Library function push uses OpenZeppelin's Address.sendValue for Ether transfers which is susceptible to reentrancy risk.

Severity

Informational

Vulnerability Detail

Library function TokenOrEther18Lib.push uses OpenZeppelin's Address.sendValue for Ether transfers which is susceptible to reentrancy risk as documented by OpenZeppelin: *"IMPORTANT: because control is transferred to `recipient`, care must be taken to not create reentrancy vulnerabilities. Consider using ReentrancyGuard or the checks-effects-interactions (CEI) pattern."* The callers of this library function may become vulnerable if they do not apply appropriate mitigations.

Impact

Callers of library function TokenOrEther18Lib.push may become susceptible to reentrancy exploits on Ether transfers if they do not use the CEI pattern and/or reentrancy guards. There are currently no calls to this library function in the code within the scope of this review.

Code Snippet

[push](#)

Tool used

Manual Review

Recommendation

Consider adding reentrancy guard to the library function or document the risk and expected mitigations for library users.

Perennial Comment

Will fix via documenting re-entrancy.

Fix PR: <https://github.com/equilibria-xyz/root/pull/35>

Sherlock Comment

Confirmed.



SHERLOCK

Issue I-02

isDoubleSided does not allow users to change sides in the same round.

Summary

isDoubleSided does not allow users to change maker-taker sides in the same round which may not be optimal.

Severity

Informational

Vulnerability Detail

The positionInvariant modifier checks whether the account isDoubleSided at the end of each openTake and openMake. However this does not allow a user to change sides in one round.

Example scenario: Alice is a taker with a position of 10, in order to change side and open a maker position of 10, she must closeTake(10) followed by openMake(10). If Alice sends both transactions in the same round, the second transaction will fail because isDoubleSided will be true even though Alice's position in the next round won't be double sided.

Impact

This forces users to change sides only in two different rounds.

Code Snippet

[isDoubleSided](#)

Tool used

Manual Review

Recommendation

Consider changing to always checking the next round's position. For e.g.:

```
function isDoubleSided (AccountPosition storage self) internal view
returns (bool) {
    Position memory nextPosition = self.position.next(self.pre);
    return !nextPosition.maker.isZero() &&
    !nextPosition.taker.isZero();
}
```

Perennial Comment

Won't fix, the math is fixable but the settlement flow has certain assumptions about single sides which would be complex to fix.



SHERLOCK

Sherlock Comment

Ok.



SHERLOCK

Issue I-03

Floating pragma allows use of buggy compiler versions.

Summary

Contracts in the root repository use pragma ^0.8.13 which allows compilation with versions 0.8.13 or 0.8.14 which have multiple unfixed compiler bugs.

Severity

Informational

Vulnerability Detail

Contracts in the root repository use pragma ^0.8.13 which allows compilation with version 0.8.13 or 0.8.14 when more recent versions 0.8.14 and 0.8.15 were recently released which fix multiple important bugs — see <https://blog.soliditylang.org/2022/05/18/solidity-0.8.14-release-announcement> and <https://blog.soliditylang.org/2022/06/15/solidity-0.8.15-release-announcement>

Impact

Compiling with 0.8.13 makes code susceptible to unfixed bugs if it uses the affected features.

Code Snippet

[UInitializable.sol](#) and others

Tool used

Manual Review

Recommendation

Consider using a fixed pragma 0.8.15 which prevents susceptibility to the known bugs fixed from 0.8.13 and 0.8.14 with the understanding that using the most recent versions may increase susceptibility to unknown compiler bugs.

Perennial Comment

We will upgrade Perennial to 0.8.15, but will keep root at ^0.8.13 as it is a library (see OpenZeppelin/contracts).

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/29>

Sherlock Comment

Confirmed.



SHERLOCK

Issue I-04

Unused error declarations.

Summary

Unused errors may be indicative of unused code or of missing logic.

Severity

Informational

Vulnerability Detail

Errors `AlreadyInitializedError` and `NotProductOwnerError` are not used to throw in any exceptional conditions. This is potentially indicative of missing checks for exceptional conditions.

Impact

Errors that are declared but not used may be indicative of unused declarations leading to reduced readability/maintainability/auditability, or worse, indicative of missing logic that may be missing exceptional condition flows with these errors.

Code Snippet

[AlreadyInitializedError](#) [NotProductOwnerError](#) [ControllerAlreadyInitializedError](#)

Tool used

Manual Review

Recommendation

Remove error declarations or add missing logic that will throw these errors.

Perennial Comment

Will fix.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/25>

Sherlock Comment

Confirmed.



SHERLOCK

Issue I-05

Code, comments & documentation.

Summary

Discrepancies in/between or inaccuracies/deficiencies in code, comment and documentation can be misleading and could indicate the presence of inaccurate implementation or documentation.

Severity

Informational

Vulnerability Detail

1. Missing detailed specification and documentation for all contracts. The current protocol documentation is high-level, user-focussed and minimal. This forces reviewers to rely on inline code comments and NatSpec thus making assumptions about the rationale behind the implementation and their intended behavior.
2. Typo: NatSpec @notice for createCoordinator and _createCoordinator should say "Creates a new coordinator with coordinatorOwner as the owner" and not `msg.sender`.
3. Typo: Natspec @dev for onlyOwner says "Only allow the protocol owner to call" but should instead say "coordinator owner" not "protocol owner" because the modifier check applies to even coordinatorId != 0.
4. Typo: Natspec in ProductManager says "Program manager" instead of "Product manager".
5. TODO comments are indicative of missing functionality which should be added or the stale comment removed.
6. Typo: Inline comment "short-circuit if $a \rightarrow c$ " is incomplete
7. Incorrect NatSpec of from function used for unpack functions
8. Missing NatSpec for storage libraries

Impact

Reduced code comprehension, auditability and maintainability.

Code Snippet

[createCoordinator](#) [_createCoordinator](#) [onlyOwner](#) [ProductManager](#) [PerennialLens](#)
[TokenOrEther18](#) [settleAccountInternal](#) [unpack](#) [unpack](#)

Tool used

Manual Review

Recommendation



SHERLOCK

Code, comments and documentation should all be complete, accurate and consistent before security review.

Perennial Comment

We have fixed all mentioned issues except for the detailed, contract level NatSpec comments (#1).

Fix PRs: <https://github.com/equilibria-xyz/perennial-mono/pull/27> and <https://github.com/equilibria-xyz/root/pull/36>

Sherlock Comment

Ok.



SHERLOCK

Issue I-06

Code structure deviates from best-practice.

Summary

It is a best practice to order different contract constructs in a certain widely-used layout for better readability and auditability.

Severity

Informational

Vulnerability Detail

The best-practice layout for a contract should follow the following order: state variables, events, modifiers, constructor and functions.

Function ordering helps readers identify which functions they can call and find constructor and fallback functions easier. Functions should be grouped according to their visibility and ordered as: constructor, receive function (if exists), fallback function (if exists), external, public, internal, private.

Some constructs deviate from this recommended best-practice: Modifiers are at the end of contracts. External/public functions are mixed with internal/private ones.

Impact

Reduced readability, auditability and maintainability.

Code Snippet

[Modifiers Private and External Functions](#)

Tool used

Manual review

Recommendation

Consider adopting recommended best-practice for code structure and layout.

Perennial Comment

Won't fix for now, the changeset here would be very large.

Sherlock Comment

Ok.



SHERLOCK

Issue I-07

Inconsistent naming convention.

Summary

Function naming convention is inconsistent in a few places.

Severity

Informational

Vulnerability Detail

One of the naming conventions is for internal/private functions to start with a leading underscore. While this is followed in some places, there are functions where this is missing.

Impact

Reduced readability and auditability.

Code Snippet

[creditAccount](#) [debitAccount](#) and many others

Tool used

Manual review

Recommendation

Use the naming convention of starting with an underscore for internal/private functions consistently.

Perennial Comment

Won't add underscores for libraries, but we will for contracts.

Fix PR: <https://github.com/equilibria-xyz/perennial-mono/pull/28>

Sherlock Comment

Confirmed.



SHERLOCK

Issue I-08

Potential gas optimizations

Summary

There are some opportunities to optimize gas usage.

Severity

Informational

Vulnerability Detail

There are categories of gas optimizations possible across contracts:

- Caching of variables in memory instead of using storage variables
- Avoiding initialisations of loop indices to their default values
- Caching loop iteration count
- Using pre-increment instead of post-increment for loop indices
- Unnecessary checked arithmetic for loop index increment but readability trade-off with unchecked block

Impact

Increased gas usage.

Code Snippet

Examples

- Caching: [sign](#), [_claimProgram](#) and others
- Loop index: [sync](#) and others
- Loop iteration count: [syncResults.length](#) [activeProgramIds.length](#) and others
- Loop index post-increment: [i++](#) and others
- Loop increment: [i++](#) and others

Tool used

Manual review

Recommendation

Consider optimizing for gas where possible.

Perennial Comment

Won't fix, after digging into this we discovered that the gas savings aren't worth the code readability losses.

Sherlock Comment

Ok.



SHERLOCK

Issue I-09 (from mini-audit)

Lack of constraints for key parameters.

Severity

Informational

Vulnerability Detail

Per the Perennial Audit Report Responses and Updates on M-03 & M-04:

"Acknowledged, we have made extensive changes to the way parameters and oracles work. Namely, we are moving all parameters (maintenance, utilization curve, funding/maker/taker fees, maker limit) from the ProductProvider to the Product directly, so we can set constraints on them."

However, we found that there are still no constraints on maintenance in the updated code:

[UParamProvider.sol#L73-L85](#)

```
function _updateMaintenance(UFixed18 newMaintenance) private {
    _maintenance.store(newMaintenance);
    emit MaintenanceUpdated(newMaintenance);
}

/**
 * @notice Updates the maintenance to `newMaintenance`
 * @dev only callable by product owner
 * @param newMaintenance new maintenance value
 */
function updateMaintenance(UFixed18 newMaintenance) external
onlyProductOwner {
    _updateMaintenance(newMaintenance);
}
```

There is an upper bound on fundingFee, makerFee, and takerFee (UFixed18Lib.ONE).

But that's merely a type check, it can not stop a malicious product owner from ripping the users off with suddenly increased fundingFee, makerFee, and takerFee.

[UParamProvider.sol#L87-L104](#)

```
/**
 * @notice Updates the funding fee to `newFundingFee`
 * @param newFundingFee new funding fee value
 */
```



SHERLOCK


```

function _updateFundingFee(UFixed18 newFundingFee) private {
    if (newFundingFee.gt(UFixed18Lib.ONE)) revert
    ParamProviderInvalidFundingFee();
    _fundingFee.store(newFundingFee);
    emit FundingFeeUpdated(newFundingFee);
}

/**
 * @notice Updates the funding fee to `newFundingFee`
 * @dev only callable by product owner
 * @param newFundingFee new funding fee value
 */
function updateFundingFee(UFixed18 newFundingFee) external
onlyProductOwner {
    _updateFundingFee(newFundingFee);
}

```

UParamProvider.sol#L106-L123

```

/**
 * @notice Updates the maker fee to `newMakerFee`
 * @param newMakerFee new maker fee value
 */
function _updateMakerFee(UFixed18 newMakerFee) private {
    if (newMakerFee.gt(UFixed18Lib.ONE)) revert
    ParamProviderInvalidMakerFee();
    _makerFee.store(newMakerFee);
    emit MakerFeeUpdated(newMakerFee);
}

/**
 * @notice Updates the maker fee to `newMakerFee`
 * @dev only callable by product owner
 * @param newMakerFee new maker fee value
 */
function updateMakerFee(UFixed18 newMakerFee) external onlyProductOwner
{
    _updateMakerFee(newMakerFee);
}

```



UParamProvider.sol#L125-L142

```
/**
 * @notice Updates the taker fee to `newTakerFee`
 * @param newTakerFee new taker fee value
 */
function _updateTakerFee(UFixed18 newTakerFee) private {
    if (newTakerFee.gt(UFixed18Lib.ONE)) revert
    ParamProviderInvalidTakerFee();
    _takerFee.store(newTakerFee);
    emit TakerFeeUpdated(newTakerFee);
}

/**
 * @notice Updates the taker fee to `newTakerFee`
 * @dev only callable by product owner
 * @param newTakerFee new taker fee value
 */
function updateTakerFee(UFixed18 newTakerFee) external onlyProductOwner
{
    _updateTakerFee(newTakerFee);
}
```

Tool used

Manual Review

Recommendation

Consider introducing a timelock for all the changes that worsen the terms for the users, and only allow the changes that make the terms better for the users to be effective immediately.

For example, updateMaintenance() to a lower value will be effective immediately, but increasing it will need to wait for a 72 hours timelock.

This allows the users to react and choose whether to accept the new terms or exit the product before the new terms become effective.

Perennial Comment

Acknowledged.

Sherlock Comment

Ok.



SHERLOCK

Issue I-10 (from mini-audit)

Using upgradeable contracts for the Oracle and PayoffDefinitionContract should be considered dangerous.

Severity

Informational

Vulnerability Detail

First of all, we welcome the changes which moved the Oracle to the Product and made it immutable and made the payoff function only optionally a call (via staticCall) to an external contract.

While you already mentioned that: *"We will only show high quality products on the main front-end that we operate (i.e. products owned by the protocol owner, or specific teams with secure ownership setups)."*

We believe it's still noteworthy to consider that if the Oracle and/or PayoffDefinitionContract is upgradeable, they should still be considered dangerous.

Even if the product owner is trustworthy, their private key may get compromised one day, and then the Oracle and PayoffDefinitionContract can be upgraded to a malicious implementation, and all the users' funds will be at risk.

Tool used

Manual Review

Recommendation

Consider checking if the Oracle and/or PayoffDefinitionContract is an upgradeable contract (eg, by checking if the DELEGATECALL opcode is used within the contract), and display warnings accordingly for the users on the frontend.

Perennial Comment

Acknowledged.

Sherlock Comment

Ok.



SHERLOCK