SHERLOCK

SHERLOCK SECURITY REVIEW FOR

PERENNIAL

Prepared For: Perennial

Prepared By: Sherlock

Lead Security Experts: WatchPug, 0xRajeev

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 CASEfile for Perennial Protocol that was prepared by Sherlock Watsons WatchPug and 0xRajeev, with assistance from Secureum participants, in the context of Secureum CASE (Collaborative Assessment & Security Evaluation) during 2-15 June, 2022.

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

- root/contracts/curve/types/*.sol
- root/contracts/number/types/*.sol
- root/contracts/token/types/*.sol
- root/contracts/storage/UStorage.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
8	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 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 ⇒ 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

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
TBD

Sherlock Comment
TBD

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

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
TBD

Sherlock Comment
TBD

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.

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
"Oh this is interesting.

I need to spend more time digging into this, but this might be something we want to address. I need to figure out exactly how this changes the p&l formula to see if there's an easy way to explain this to the end user. I'm thinking it's going to be too complicated to completely change the mechanism, but did have an idea:

What if instead we simply locked down the settleAccount() function so that it can't be called directly, only by actual position change actions internally (or privileged external ones)? That way this weirdness only occurs when a user is actually modifying their position (which can intuitively be mapped to closing and reopening their position) or getting liquidated. Does that sound reasonable, I think that solves this, but I'm not sure? Will definitely think through this more though."

"Yeah digging into this more I think this is right: my solution by itself won't actually help.

I think in order to "solve" this you'd have to both 1) lock down settleAccount() and (2) use only "settled" collateral in the maintenance check. Haven't fully dug through if there'd be downstream effects though.

The original idea was to keep those functions open to reduce code complexity since they do need to be called externally from Collateral, Incentivizer, and any Lens that gets developed.

Another direction though is that we also don't have to emulate tradperps directly - we are already designing somewhat of a new financial product here.

I math'd out how the liq price changes in this model and it's still static given the price and collateral of the user at time of open, so another option would be to clearly display this in the UI on position open (estimated) and after the position is confirmed, instead of just giving the tradition maintenance ratio by itself. This could also be supplemented with some explainer.

tradperps: liq. price = collateral / maintenance
perennial: liq. price = (initial collateral + initial price) / (maintenance + 1)

So it's still computable, and static over the course of the position, but the formula is different"

"hmm I think the unrealizedPNL might be deceivingly hard to implement since positions can be partially opened / closed over time. Like when a position is partially closed, it would be tough I think to attribute to correct proportion of the accumulated unrealizedPNL to that close specifically. Especially with lots of partial opens and closes over time."

"I originally thought this too - if the liquidation price does in fact move around depending on when and how often it's settle is called (especially if its death spiraling), then it would make sense to change.

However after running the math we figured out that the liquidation is still static per position (no matter when / how often settle is called) just like a traditional perp, it just has a little bit of a different formula to compute the liquidation price:

*liquidation price = (initial collateral + position * initial price) / (position * (maintenance ratio + 1))*

Given this, I think it makes sense to keep as is, but properly document how the "maintenance" ratio differs from a traditional perp. Would love for you all to double check that this is in fact static like we're thinking, or if there's other downsides we're not seeing."

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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

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

"It's best to think of Perennial as a neutral base layer here sort of like Uniswap. Anyone can come and launch a product, and they're free to code their ProductProvider however they wish, and so this does give them the flexibility to create a malicious provider.

The idea here is that Perennial does not explicitly endorse any individual product as safe, in the same way that a Uniswap pool can be created with maliciously-coded tokens. It will ultimately be up to the front-end to choose which products they end up showing (we imagine even each coordinator team having their own frontends in the future)

There's a few caveats though:
- Perennial pools must remain silo'd: a malicious product cannot affect any other product
- There's no reason to allow values totally outside the reasonable domain, so when we can validate (like in ProductProviderLib) we still should
- As a launch strategy, we will be locking down the createProduct method (as you can see in the current code) so that we can ensure an initial set of "team-trusted" products are available at launch, simplifying the user experience."

"This is kind of a large topic, so let me try and shed some light on our thinking here, but we absolutely would also like input and recommendations from the auditors on this piece as well.

There's four threat levels that I see for the coordinators' privileged role:

(1) coordinator can exfiltrate funds from the market arbitrarily, through invalid parameters or functions

(2) coordinators can break / lock their market through reverts caused by invalid parameters or functions

(3) coordinators can collect an unreasonable amount of funds, likely though high fees / quickly changing parameters etc, or can otherwise manipulate the market to the users' inconvenience

(4) coordinators can set risk parameters such that insolvency can occur through loose maintenance requirements.

Our intention here is to fully lock down any threats in the (1) and (2) buckets, and most if not all in the (3) bucket. I believe (3) is where there is the most room for improvement at the moment, so we'd love feedback that we can incorporate as part of our audit remediation process.

(Some ideas we've had on this, but not sure if they made sense from a threat model perspective)

- Moving params out of the provider into Product or Controller so that they can have more restricted update logic

- Forcing the "JumpRate" utilization curve model so that we can move that and its parameters similarly to the Product or Controller

(4) and potentially some parts of (3) are likely unavoidable due to the fact that we want to enable arbitrary payoff functions in our market. At this point, this is where the reputation of the coordinator is important (and why our initial coordinator set will be teams like Opyn, Gauntlet, etc), and it will be on the frontends to select which markets are reputable enough from a risk-management perspective to show. There's also a possibility that through the above recommendations, the Provider contract could be simplified significantly and can therefore be relatively easily spot-checked."

Sherlock Comment

TBD

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
TBD

Sherlock Comment
TBD

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

TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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 an 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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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 griefing 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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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 >= roundID and updatedAt != 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.*"

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment

TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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→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

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

Perennial Comment
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD

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
TBD

Sherlock Comment
TBD