SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Perennial
Prepared by:	Sherlock
Lead Security Expert:	rogueréddwarf
Dates Audited:	May 22 - June 15, 2023
Prepared on:	August 31, 2023

Introduction

Perennial is built from first-principles to be a powerful, flexible, and composable primitive that can scale to meet the needs of DeFi traders, liquidity providers, and developers.

Scope

Repository: equilibria-xyz/perennial-mono

Branch: dev2

Commit: b06d5145db62a312dd88dfcafef0f8e2166c5e32

Repository: equilibria-xyz/root

Branch: dev

Commit: 914838c1cb2532325ecf5659807f9fca61d635e9

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
16	1

Security experts who found valid issues

roguereddwarf	0xGoodess	Phantasmagoria
cergyk	Emmanuel	ast3ros
mstpr-brainbot	rvierdiiev	branch_indigo

AkshaySrivastav	SolidityATL	Bauchibred
tvdung94	BLACK-PANDA-REACH	simon135

Issue H-1: BalancedVault.sol: loss of funds + global settlement flywheel / user settlement flywheels getting out of sync

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/45

Found by

0xGoodess, cergyk, roguereddwarf

Summary

When an epoch has become "stale", the `BalancedVault` will treat any new deposits and redemptions in this epoch as "pending". This means they won't get processed by the global settlement flywheel in the next epoch but one epoch later than that.

Due to the fact that anyone can push a pending deposit or redemption of a user further ahead by making an arbitrarily small deposit in the "intermediate epoch" (i.e. the epoch between when the user creates the pending deposit / redemption and the epoch when it is scheduled to be processed by the global settlement flywheel), the user can experience a DOS.

Worse than that, by pushing the pending deposit / pending redemption further ahead, the global settlement flywheel and the user settlement flywheel get out of sync.

Also users can experience a loss of funds.

Vulnerability Detail

Let's assume we are currently in epoch `1` and it is `stale`. User1 calls the `deposit` function and we set `_pendingEpochs[user1] = context.epoch + 1 = 2` https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L160-L164

We move one epoch ahead and are in epoch `2` now. By looking at the global settlement flywheel we see that `_deposit = _pendingDeposit` and `_redemption = _pendingRedemption` which means that the deposit we made will be processed in the global settlement flywheel in the next epoch (epoch `3`). https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L378-L403

By looking into the user settlement flywheel we see that the `if (accountContext.epoch > _pendingEpochs[account])` condition to process the pending deposit is not fulfilled yet (`2 !> 2`). It will be fulfilled in epoch `3`.

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L413-L422

So far so good. The global settlement flywheel and the user settlement flywheel are in sync and will process the pending deposit in epoch `3`.

Now here's the issue. A malicious user2 or user1 unknowingly (depending on the specific scenario) calls `deposit` for user1 again in the current epoch `2` once it has become `stale` (it's possible to deposit an arbitrarily small amount). By doing so we set `_pendingEpochs[user1] = context.epoch + 1 = 3`, thereby pushing the processing of the deposit in the user settlement flywheel one epoch ahead.

It's important to understand that the initial deposit will still be processed in epoch `3` in the global settlement flywheel, it's just being pushed ahead in the user settlement flywheel.

Thereby the global settlement flywheel and user settlement flywheel are out of sync now.

The total supply will be increased by an amount calculated based on the epoch `3` context: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L379

The user balance will be increased by an amount calculated based on the epoch `4` context: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L407

An example for a loss of funds that can occur as a result of this issue is when the PnL from epoch `3` to epoch `4` is positive. Thereby the user1 will get less shares than he is entitled to.

Similarly it is possible to push pending redemptions ahead, thereby the `_totalUnclaimed` amount would be increased by an amount that is different from the amount that `_unclaimed[account]` is increased by.

Coming back to the case with the pending deposit, I wrote a test that you can add to `BalancedVaultMulti.test.ts`:

```
it('pending deposit pushed by 1 epoch causing shares difference', async () => {
      const smallDeposit = utils.parseEther('1000')
      const smallestDeposit = utils.parseEther('0.000001')

      await updateOracleEth() // epoch now stale
      // make a pending deposit
      await vault.connect(user).deposit(smallDeposit, user.address)
      await updateOracleBtc()
      await vault.sync()

      await updateOracleEth() // epoch now stale
      /*
      user2 deposits for user1, thereby pushing the pending deposit ahead and
↪   causing the
      global settlement flywheel and user settlement flywheel to get out of sync
      */
      await vault.connect(user2).deposit(smallestDeposit, user.address)
      await updateOracleBtc()
      await vault.sync()

      await updateOracle()
      // pending deposit for user1 is now processed in the user settlement
↪   flywheel
      await vault.syncAccount(user.address)

      const totalSupply = await vault.totalSupply()
      const balanceUser1 = await vault.balanceOf(user.address)
      const balanceUser2 = await vault.balanceOf(user2.address)

      /*
      totalSupply is bigger than the amount of shares of both users together
      this is because user1 loses out on some shares that he is entitled to
      -> loss of funds
      */
      console.log(totalSupply);
      console.log(balanceUser1.add(balanceUser2));

})
```

The impact that is generated by having one pending deposit that is off by one epoch is small. However over time this would evolve into a chaotic situation, where the state of the Vault is significantly corrupted.

Impact

The biggest impact comes from the global settlement flywheel and user settlement flywheel getting out of sync. As shown above, this can lead to a direct loss of funds for the user (e.g. the amount of shares he gets for a deposit are calculated with the wrong context).

Apart from the direct impact for a single user, there is a subtler impact which can be more severe in the long term. Important invariants are violated:

- Sum of user balances is equal to the total supply
- Sum of unclaimed user assets is equal to total unclaimed assets

Thereby the impact is not limited to a single user but affects the calculations for all users.

Less important but still noteworthy is that users that deposit into the Vault are partially exposed to PnL in the underlying products. The Vault does not employ a fully delta-neutral strategy. Therefore by experiencing a larger delay until the pending deposit / redemption is processed, users incur the risk of negative PnL.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L156-L175

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L184-L205

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L375-L424

Tool used

Manual Review

Recommendation

My recommendation is to implement a queue for pending deposits / pending redemptions of a user. Pending deposits / redemptions can then be processed independently (without new pending deposits / redemptions affecting when existing ones are processed).

Possibly there is a simpler solution which might involve restricting the ability to make deposits to the user himself and only allowing one pending deposit / redemption to exist at a time.

The solution to implement depends on how flexible the sponsor wants the deposit / redemption functionality to be.

Discussion

KenzoAgada

Also note duplicate issue #74 which mentions how a user can redeem more assets than he's entitled to.

Issue M-1: ChainlinkAggregator: binary search for roundId does not work correctly and Oracle can even end up temporarily DOSed

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/4

Found by

roguereddwarf

Summary

When a phase switchover occurs, it can be necessary that phases need to be searched for a `roundId` with a timestamp as close as possible but bigger than `targetTimestamp`.

Finding the `roundId` with the closest possible timestamp is necessary according to the sponsor to minimize the delay of position changes:

The binary search algorithm is not able to find this best `roundId` which thereby causes unintended position changes.

Also it can occur that the `ChainlinkAggregator` library is unable to find a valid `roundId` at all (as opposed to only not finding the "best").

This would cause the Oracle to be temporarily DOSed until there are more valid rounds.

Vulnerability Detail

Let's look at the binary search algorithm:
https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L123-L156

The part that we are particularly interested in is:
https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L139-L149

Let's say in a phase there's only one valid round (`roundId=1`) and the timestamp for this round is greater than `targetTimestamp`

We would expect the `roundId` that the binary search finds to be `roundId=1`.

The binary search loop is executed with `minRoundId=1` and `maxRoundId=1001`.

All the above conditions can easily occur in reality, they represent the basic scenario under which this algorithm executes.

`minRoundId` and `maxRoundId` change like this in the iterations of the loop:

```
minRoundId=1
maxRoundId=1001

->

minRoundId=1
maxRoundId=501

->

minRoundId=1
maxRoundId=251

->

minRoundId=1
maxRoundId=126

->

minRoundId=1
maxRoundId=63

->

minRoundId=1
maxRoundId=32

->

minRoundId=1
maxRoundId=16

->

minRoundId=1
maxRoundId=8

->

minRoundId=1
maxRoundId=4

->

minRoundId=1
```

```
maxRoundId=2

Now the loop terminates because
minRoundId + 1 !< maxRoundId
```

Since we assumed that `roundId=2` is invalid, the function returns `0` (`maxTimestamp=type(uint256).max`):

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L153-L155

In the case that `latestRound.roundId` is equal to the `roundId=1` (i.e. same phase and same round id which could not be found) there would be no other valid rounds that the `ChainlinkAggregator` can find which causes a temporary DOS.

Impact

As explained above this would result in sub-optimal and unintended position changes in the best case. In the worst-case the Oracle can be temporarily DOSed, unable to find a valid `roundId`.

This means that users cannot interact with the perennial protocol because the Oracle cannot be synced. So they cannot close losing trades which is a loss of funds.

The DOS can occur since the while loop searching the phases does not terminate: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L88-L91

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/types/ChainlinkAggregator.sol#L123-L149

Tool used

Manual Review

Recommendation

I recommend to add a check if `minRoundId` is a valid solution for the binary search. If it is, `minRoundId` should be used to return the result instead of `maxRoundId`:

```
        // If the found timestamp is not greater than target timestamp or no
↪   max was found, then the desired round does
        // not exist in this phase
```

```
-         if (maxTimestamp <= targetTimestamp || maxTimestamp ==
↪   type(uint256).max) return 0;
+         if ((minTimestamp <= targetTimestamp || minTimestamp ==
↪   type(uint256).max) && (maxTimestamp <= targetTimestamp || maxTimestamp ==
↪   type(uint256).max)) return 0;

+         if (minTimestamp > targetTimestamp) {
+             return _aggregatorRoundIdToProxyRoundId(phaseId,
↪   uint80(minRoundId));
+         }
          return _aggregatorRoundIdToProxyRoundId(phaseId, uint80(maxRoundId));
     }
```

After applying the changes, the binary search only returns `0` if both `minRoundId` and `maxRoundId` are not a valid result.

If this line is passed we know that either of both is valid and we can use `minRoundId` if it is the better result.

Discussion

roguereddwarf

Escalate for 10 USDC

I think this should be a "High" severity finding. The binary search lies at the core of the protocol. All functionality for users to open / close trades and liquidations relies on the Chainlink oracle.

By not finding a valid `roundId` obviously many users are put at risk of losing funds (-> not being able to close trades).

Similarly when an unintended (i.e. sub-optimal) `roundId` is found this leads to a similar scenario where settlements / liquidations occur at unintended prices.

In summary, the fact that the binary search algorithm lies at the core of the protocol and there is a very direct loss of funds makes me think this should be "High" severity.

sherlock-admin

> Escalate for 10 USDC
>
> I think this should be a "High" severity finding. The binary search lies at the core of the protocol. All functionality for users to open / close trades and liquidations relies on the Chainlink oracle.
>
> By not finding a valid `roundId` obviously many users are put at risk of losing funds (-> not being able to close trades).

> Similarly when an unintended (i.e. sub-optimal) `roundId` is found this leads to a similar scenario where settlements / liquidations occur at unintended prices.
>
> In summary, the fact that the binary search algorithm lies at the core of the protocol and there is a very direct loss of funds makes me think this should be "High" severity.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

securitygrid

Comment from watson: Chainlink: ETH/USD has been deployed for 3 years, and the current phaseID is only 6. The binary search is only triggered when this condition is met, that is, when a new phaseID is generated. This is infrequent. This is from the chainlink docs: `phaseId` is incremented each time the underlying aggregator implementation is updated.

KenzoAgada

The issue is dependent upon a Chainlink changing of phase (which is quite infrequent) and the new phase having only 1 round. The impact as stated in the finding is that in those conditions, the product flywheel is jammed until further rounds are issued. (Then, the algorithm will correctly return round 2.) So the impact is only very temporary and rare DOS (which can impact user funds).

I think escalation is invalid and medium severity is appropriate.

arjun-io

To add here - the binary search is also a backup solution to simply checking if roundId + 1 exists in the previous phase (code). So additionally the last seen round in the previous phase has to be the very last roundID in that phase

jacksanford1

Agree with Medium due to Kenzo's reasons (unlikely and temporary situation which only indirectly could result in user funds loss due to liquidations). cc @roguereddwarf

roguereddwarf

Agreed

jacksanford1

Result: Medium Unique Unlikely and temporary situation which only indirectly could result in user funds loss due to liquidations is part of the reason why this is not being upgraded to a High.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- roguereddwarf: rejected

arjun-io

Fixed in: https://github.com/equilibria-xyz/perennial-mono/pull/207 as per the recommended fix

Issue M-2: Missing Sequencer Uptime Feed check can cause unfair liquidations on Arbitrum

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/37

Found by

roguereddwarf

Summary

When the Arbitrum sequencer is down and then comes back up, all Chainlink price updates will become available on Arbitrum within a very short time.

This leaves users no time to react to the price changes which can lead to unfair liquidations.

Vulnerability Detail

Chainlink explains their Sequencer Uptime Feeds here.

Quoting from the documentation:

> To help your applications identify when the sequencer is unavailable, you can use a data feed that tracks the last known status of the sequencer at a given point in time. This helps you prevent mass liquidations by providing a grace period to allow customers to react to such an event.

Users are still able in principle to avoid liquidations by interacting with the Arbitrum delayed inbox via L1, but this is out of reach for most users.

Impact

Users can get unfairly liquidated because they cannot react to price movements when the sequencer is down and when the sequencer comes back up, all price updates will immediately become available.

Code Snippet

This issue can be observed in both the `ChainlinkOracle` and `ChainlinkFeedOracle`, which do not make use of the sequencer uptime feed to check the status of the sequencer:

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/ChainlinkOracle.sol#L59-L64 https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-oracle/contracts/ChainlinkFeedOracle.sol#L94-L97

Tool used

Manual Review

Recommendation

The Chainlink documentation contains an example for how to check the sequencer status: https://docs.chain.link/data-feeds/l2-sequencer-feeds

There can be a grace period when the sequencer comes back up for users to act on their collateral (increase collateral to avoid liquidation).

Discussion

roguereddwarf

Escalate for 10 USDC

This issue is marked as a duplicate of #13.

This is wrong and I argue that my report is a legitimate Medium (speaking only of my report here, other dupes must be checked as well).

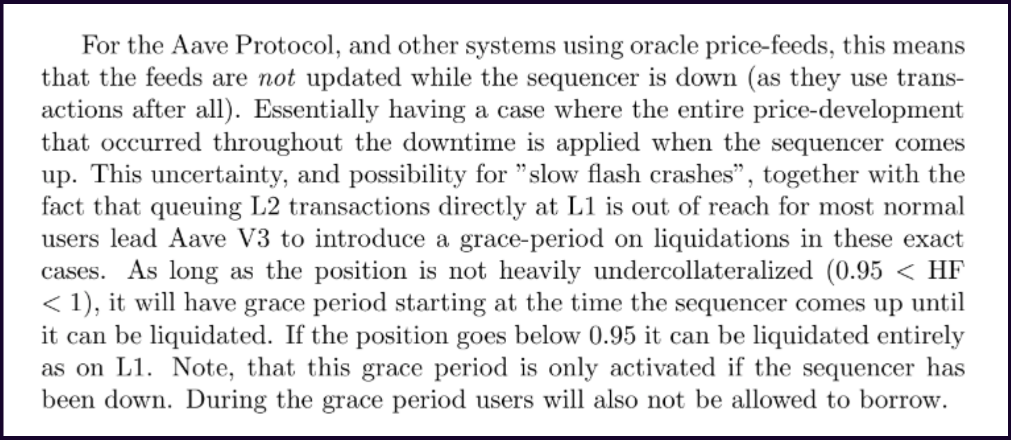
#13 argues that outdated prices would be used when the sequencer is down. The sponsor correctly explained that:

> From our understanding, when the sequencer is down the prices can't be updated by the data feed so therefore settlement can't occur. This means that effectively each product is paused as no state changes can occur

My point on the other hand is that when the sequencer comes back up, all the old prices will be processed at once and users have no time to react to price changes, which can lead to unfair liquidations.

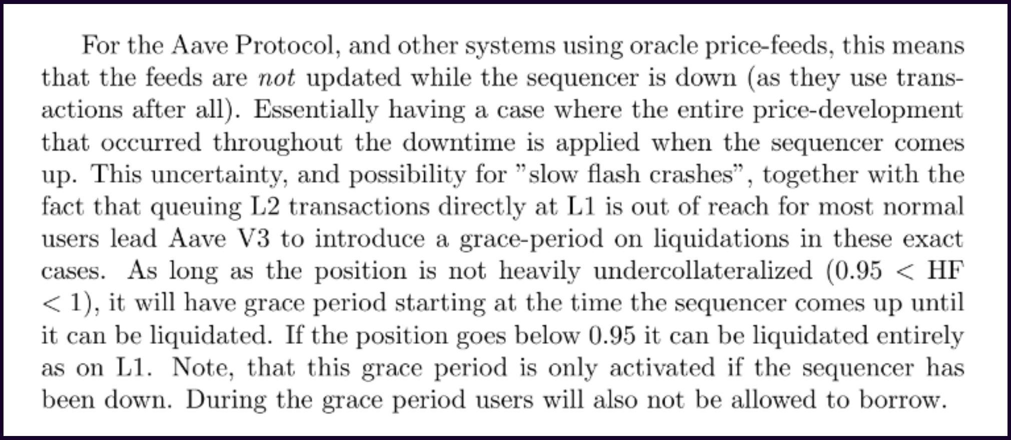
Therefore I had the suggestion for a grace period.

You can see a detailed explanation for my argument in the Aave V3 technical paper: https://github.com/aave/aave-v3-core/blob/master/techpaper/Aave_V3_Technical_Paper.pdf (section 4.6)


For the Aave Protocol, and other systems using oracle price-feeds, this means that the feeds are *not* updated while the sequencer is down (as they use transactions after all). Essentially having a case where the entire price-development that occurred throughout the downtime is applied when the sequencer comes up. This uncertainty, and possibility for "slow flash crashes", together with the fact that queuing L2 transactions directly at L1 is out of reach for most normal users lead Aave V3 to introduce a grace-period on liquidations in these exact cases. As long as the position is not heavily undercollateralized ($0.95 < HF < 1$), it will have grace period starting at the time the sequencer comes up until it can be liquidated. If the position goes below 0.95 it can be liquidated entirely as on L1. Note, that this grace period is only activated if the sequencer has been down. During the grace period users will also not be allowed to borrow.

sherlock-admin

> Escalate for 10 USDC
>
> This issue is marked as a duplicate of #13.
>
> This is wrong and I argue that my report is a legitimate Medium (speaking only of my report here, other dupes must be checked as well).
>
> #13 argues that outdated prices would be used when the sequencer is down. The sponsor correctly explained that:
>
> > From our understanding, when the sequencer is down the prices can't be updated by the data feed so therefore settlement can't occur. This means that effectively each product is paused as no state changes can occur
>
> My point on the other hand is that when the sequencer comes back up, all the old prices will be processed at once and users have no time to react to price changes, which can lead to unfair liquidations.
>
> Therefore I had the suggestion for a grace period.
>
> You can see a detailed explanation for my argument in the Aave V3 technical paper: https://github.com/aave/aave-v3-core/blob/master/techpaper/Aave_V3_Technical_Paper.pdf (section 4.6)



For the Aave Protocol, and other systems using oracle price-feeds, this means that the feeds are *not* updated while the sequencer is down (as they use transactions after all). Essentially having a case where the entire price-development that occurred throughout the downtime is applied when the sequencer comes up. This uncertainty, and possibility for "slow flash crashes", together with the fact that queuing L2 transactions directly at L1 is out of reach for most normal users lead Aave V3 to introduce a grace-period on liquidations in these exact cases. As long as the position is not heavily undercollateralized ($0.95 < \text{HF} < 1$), it will have grace period starting at the time the sequencer comes up until it can be liquidated. If the position goes below 0.95 it can be liquidated entirely as on L1. Note, that this grace period is only activated if the sequencer has been down. During the grace period users will also not be allowed to borrow.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

- Watson is correct in saying that his issue is different than the rest of the "Arbitrum sequencer downtime" issues with regards to the impact described.
- While all other issues speak about stale prices, this issue says that the problem is unfair liquidations as users will not have time to react when oracle data feeds are updated again.
- Therefore watson suggests a grace period.
- So this is similar issue to #190, which mentions the lack of grace period when unpausing.
- There is an ongoing escalation there. The protocol team disputed the issue, but the escalator wrote that the *validity* of the issue should be accepted. Please see that issue for all the context and comments.

I think that this issue should have a similar fate to #190. If that escalation is accepted, this one should be accepted as well.

jacksanford1

Believe #190 is trending towards being accepted. This issue has a different root case (Arbitrum sequencer down vs. protocol team pausing contracts) but the effect is the same (depositors can't salvage their position before they get liquidated).

I think it should be a valid Medium, even though the root case is a temporary freezing.

jacksanford1

Result: Medium Unique Reasoning can be found in previous message.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- rogueredwarf: accepted

Issue M-3: Market Allows Zero Amount Positions Which Can Cause Vault Rebalancing to Revert

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/39

Found by

mstpr-brainbot

Summary

When adjusting positions, it correctly calculates the amount and respects limits, but it runs into a problem when dealing with a zero amount in the openMake and closeMake functions. This problem occurs when the Vault doesn't wish to change any position but still communicates a "0" amount to the product, resulting in revert and halting the rebalancing process.

Vulnerability Detail

When decreasing a position, the Market Vault calculates the available room and permits only the closing of positions up to the point where the maker equals the taker. If the market's taker is greater than the maker, the amount is deemed as zero. Conversely, when increasing a position, the Vault checks if the product's maker limit is exceeded. If it is, the Vault assigns the amount as zero. Following this, the Vault executes the closeMake and openMake functions using the calculated amount.

However, an issue arises with the Vault's openMake and closeMake functions operating with a zero amount. Even though the Vault might not want to open or close any position, it still calls the product with a "0" amount. This leads to the modifiers throwing an error, and the rebalancing process gets stuck.

Impact

In such cases where maker limit is exceeded in product or takers > makers vaults rebalancing will be stucked. Therefore, I'll call it as high

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L502-L509

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L274-L355

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L525-L544C6 https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L499-L519

Tool used

Manual Review

Recommendation

If the calculated openMake/closeMake amount is "0" don't call the products functions. Just exit the function.

Discussion

mstpr

Escalate for 10 USDC

This is a valid issue.

The problem stated here is not that the products allow "0" amount positions to be created. It is the opposite, it shouldn't allow the "0" amount positions. When vault decides that the maker limit is reached, it sets the position to be opened as "0" and calls the product. If the position to be opened calculated as "0" the product will revert because of the modifiers.

Example:

Imagine the product has more takers than makers and the position needs to be decreased in vault. Vault will calculate the closeMake amount as 0 as seen here, and calls the product with closeMake(0) https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L502-L509

closeMake function has a modifier called takerInvariant which will be executed after the function body and it checks the socialization factor which is maker/taker, and since takers are greater than makers this modifier will revert because socialization factor indeed lesser than 1.

https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544

So, vault knew that its going to revert and that's why it calculated it as 0 but it shouldn't have called the product because product allows "0" amount positions and modifiers will revert.

same is applicable with openMake. If the makerLimit is exceeded in product, vault calculates the amount of positions to be opened as "0" but it calls the product with openMake("0") and it reverts in the makerInvariant modifier.

The correct behaviour would be exiting the function if the position to be opened or closed found to be "0".

sherlock-admin

> Escalate for 10 USDC
>
> This is a valid issue.
>
> The problem stated here is not that the products allow "0" amount positions to be created. It is the opposite, it shouldn't allow the "0" amount positions. When vault decides that the maker limit is reached, it sets the position to be opened as "0" and calls the product. If the position to be opened calculated as "0" the product will revert because of the modifiers.
>
> Example:
>
> Imagine the product has more takers than makers and the position needs to be decreased in vault. Vault will calculate the closeMake amount as 0 as seen here, and calls the product with closeMake(0) https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L502-L509
>
> closeMake function has a modifier called takerInvariant which will be executed after the function body and it checks the socialization factor which is maker/taker, and since takers are greater than makers this modifier will revert because socialization factor indeed lesser than 1.
>
> https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544
>
> So, vault knew that its going to revert and that's why it calculated it as 0 but it shouldn't have called the product because product allows "0" amount positions and modifiers will revert.
>
> same is applicable with openMake. If the makerLimit is exceeded in product, vault calculates the amount of positions to be opened as "0" but it calls the product with openMake("0") and it reverts in the makerInvariant modifier.
>
> The correct behaviour would be exiting the function if the position to be opened or closed found to be "0".

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

arjun-io

We're looking into this for the socialization case with a 0 value position. However, in general an openMake/closeMake call (even if 0) is *required* for each rebalance because a settlement version needs to be stamped for that action. This ensures that valueAtVersion(version + 1) is available for an given stamped version in the vault.

arjun-io

This does appear to be a valid issue although the fix recommended is incorrect. The vault should be opening 0 value positions in some way when the socialization or makerLimit cases are hit. We'll take a closer look into the right fix here

We would say this is a medium issue though because it is an edge case in the markets and does not result in loss of user funds

KenzoAgada

I didn't fully understand the original issue, but the escalation made it clear. Agreeing with sponsor that issue is a valid medium.

jacksanford1

Ok, valid medium

jacksanford1

Result: Medium Unique Based on consensus from Lead Judge and Arjun after seeing escalator's comment.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- mstpr: accepted

Issue M-4: Payoff definitions that can cross zero price are not supported

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/40

Found by

roguereddwarf

Summary

The price that is returned by an Oracle can be transformed by a "Payoff Provider" to transform the Oracle price and implement different payoff functions (e.g. 3x Leveraged ETH, ETH*ETH).

According to the documentation, any payoff function is supported: https://docs.perennial.finance/mechanism/payoff

The problem is that this is not true.

Vulnerability Detail

Payoff functions that transform the Oracle price such that the transformed price can cross from negative to positive or from positive to negative are not supported.

This is because there is a location in the code, where there is division by `currentPrice`: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L487

Thus there is a risk of division-by-zero if the price was allowed to cross the 0 value.

Impact

In contrast to what is specified in the Docs, not all payoff functions are supported which can lead to a situation where it causes division-by-zero and thereby DOS.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L487

Tool used

Manual Review

Recommendation

Establish which payoff functions are supported and make it clear in the docs or find a meaningful way to handle the above case. If `currentPrice==0`, a meaningful value could be to set `currentPrice=1`

Discussion

arjun-io

We'll update the docs to clarify some payoffs which aren't supported

SergeKireev

Escalate for 10 USDC

This should be low/informational, since payoffs crossing zero price are indeed supported because absolute value of the price is taken in account to compute leverage: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L486

However the concept of leverage itself makes no sense if price == 0.

This is an undefined behavior in a very very unlikely edge case (Someone defines a zero crossing payoff, and this payoff ends up being exactly zero for a prolonged duration)

Additionally the recommendation seems to do more harm than good:

> If currentPrice==0, a meaningful value could be to set currentPrice=1

If market conditions mark an asset to zero, it seems best to fail than to adjust the price to another arbitrary value

sherlock-admin

> Escalate for 10 USDC
>
> This should be low/informational, since payoffs crossing zero price are indeed supported because absolute value of the price is taken in account to compute leverage: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L486
>
> However the concept of leverage itself makes no sense if price == 0.
>
> This is an undefined behavior in a very very unlikely edge case (Someone defines a zero crossing payoff, and this payoff ends up being exactly zero for a prolonged duration)
>
> Additionally the recommendation seems to do more harm than good:

> > If currentPrice==0, a meaningful value could be to set currentPrice=1
>
> If market conditions mark an asset to zero, it seems best to fail than to adjust the price to another arbitrary value

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

roguereddwarf

Escalate for 10 USDC

I disagree with the first escalation and think this should remain "Medium" severity.

The docs clearly state that ANY payoff function is supported which includes a payoff function that crosses zero.

This is an edge case in which the Vault does not work, i.e. users cannot interact with the Vault which puts the users' funds at risk because they are unable to withdraw them and thereby are at risk of liquidation / adverse price movements.

sherlock-admin

> Escalate for 10 USDC
>
> I disagree with the first escalation and think this should remain "Medium" severity.
>
> The docs clearly state that ANY payoff function is supported which includes a payoff function that crosses zero.
>
> This is an edge case in which the Vault does not work, i.e. users cannot interact with the Vault which puts the users' funds at risk because they are unable to withdraw them and thereby are at risk of liquidation / adverse price movements.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

This is indeed quite an edge case, but it is there, and part of an audit's job is to find the edge cases. Product owners can choose any payoff function, including one that returns 0 at some instances. The issue can cause temporary DOS due to product flywheel jamming. I think medium severity is appropriate.

mstpr

Didn't escalated it because @SergeKireev already explained in very good. This is indeed an informational or invalid issue.

As stated in the readme:

if the product payoff is dodgy, then its users responsibility to deposit in such product.

Also, how can currentPrice returns "0" ? If the price of something is "0" then how can you potentially trade and even leverage the asset? As @SergeKireev said, if the current price returns to be "0", then it's best to leave it because there is something wrong with the oracle.

Also, I don't think document needed to say "0 * ETH" positions are not allowed. It's clearly doesn't make sense to inform users in the document that this is not accepted.

Clearly, 0 payoff functions are not supported because they don't make sense. Also, if any of the products price is cross to 0 Perennial still allows that, it just reverts because that's the right thing to do.

jacksanford1

I'm leaning towards Invalid because the Sherlock judging rules state that temporary freezing/DOS is not a valid issue: https://docs.sherlock.xyz/audits/judging/judging#some-standards-observed

Could be valid if @roguereddwarf can come up with an attack vector beyond temporary freezing.

@mstpr also brings up a "trusted actor" problem which I haven't fully evaluated.

roguereddwarf

The impact beyond temporary freezing is that a user can e.g. not redeem his funds in an emergency event, risking further losses.

jacksanford1

Ok, I agree with the premise that an extended DOS could result in unintended behavior like liquidations which the user may not have a chance to prevent when the DOS ends.

@roguereddwarf @mstpr @KenzoAgada @SergeKireev How should we think about the realistic length of the DOS and the damage that could result to users because of it? Is this DOS going to last a few blocks maximum? Or a few months potentially?

SergeKireev

@jacksanford1 With all due respect, discussing the details of the DOS in this case is missing the bigger picture;

A market accepting a zero price is broken in more serious ways:

During the period in which the oracle returns zero, any trader Alice can open positions of arbitrary size without having any collateral deposited since maintenance is zero: https://github.com/sherlock-audit/2023-05-perennial-SergeKireev/blob/c90b017ad432b283bcfbec594f80e18204ee98c3/perennial-mono/packages/perennial/contracts/product/types/position/AccountPosition.sol#L68-L73

and collateral can be zero: https://github.com/sherlock-audit/2023-05-perennial-SergeKireev/blob/c90b017ad432b283bcfbec594f80e18204ee98c3/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L211-L227

So if the oracle ever updates again to a non-zero price, Alice creates a shortfall of an arbitrary size on the protocol.

In that regard, by reverting, `BalancedVault` has a more reasonable behavior than the underlying market when oracle price is zero.

However `BalancedVault` links many heterogenous markets together, and in the scenario in which one fails, the whole vault fails and funds may stay locked up. I would argue that this report is an example of the broader #232 and thus should be a valid duplicate.

mstpr

@jacksanford1 How can price be "0" ? I think we need the answer for this question.

Obviously creating a "0 * ETH" payoff product is not make sense and even though its created nobody would deposit to such product.

If price is "0" this can be possible because of an oracle error. I don't think Chainlink validations are a medium finding in Sherlock. If it is, then there are many other submissions regards to that. Example: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/62 I am sure there are many others aswell

KenzoAgada

> @roguereddwarf @mstpr @KenzoAgada @SergeKireev How should we think about the realistic length of the DOS and the damage that could result to users because of it? Is this DOS going to last a few blocks maximum? Or a few months potentially?

I would consider it few blocks maximum.

Anyway, @SergeKireev 's last comment pretty much convinced me that actually reverting is a relatively logical thing to do in this scenario.

My original thinking was more like, we can not know how funky the product owner would define his payout function. (shifting, steps, more inventions...). And if it's 0, accidentally or purposefully, indeed the product would be jammed. But as the escalators wrote, there's no real meaning to a payoff function returning 0.

I already considered this quite the edge case before... At the moment I agree with the escalators (love using that word) that medium severity is too much.

jacksanford1

Ok, the original issue impact was focused on DOS, so I'll stay on that topic. If Kenzo is correct and the length of the DOS is a "few blocks maximum" then this should be a low severity issue. But open to arguments from @rogueredddwarf or anyone else as to why it should stay a Medium.

roguereddwarf

I'd like to add that the price of an asset might just go to zero (look e.g. what we've seen with FTX and FTT).

In this case it should still be possible for users of the Vault that have redeemed their shares to call the claim function and get their share of the assets that were within the Vault and have not been lost (pointing this out because someone might argue price=0 implies that there are no funds left in the Vault):

https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L211-L228

Which would however revert when `_rebalance` is called which is my whole point in this report.

While the title of my report says "that can cross zero price" this technically includes the case when the price of an asset just goes to zero due to an issue with the asset.

In this case the DOS can be for an unlimited amount of time.

KenzoAgada

While the payoff function might transform the price and return 0 and then withdrawals would be bricked, I don't think the oracle price can return 0... Chainlink has a `minAnswer` which is bigger than 0.

jacksanford1

Fair point from Kenzo that @roguereddwarf can respond to. I also think FTT is a tough example because I'm not aware that it ever went to 0. And I can't think of another token that actually hit zero ever. I'm not saying that it's impossible, I'm saying that the likelihood is extremely low because there's always a chance that a token could have value in the future, so there is always some bid for it. And if the severity of this issue is based on the likelihood of a token hitting zero, then I think this is relevant.

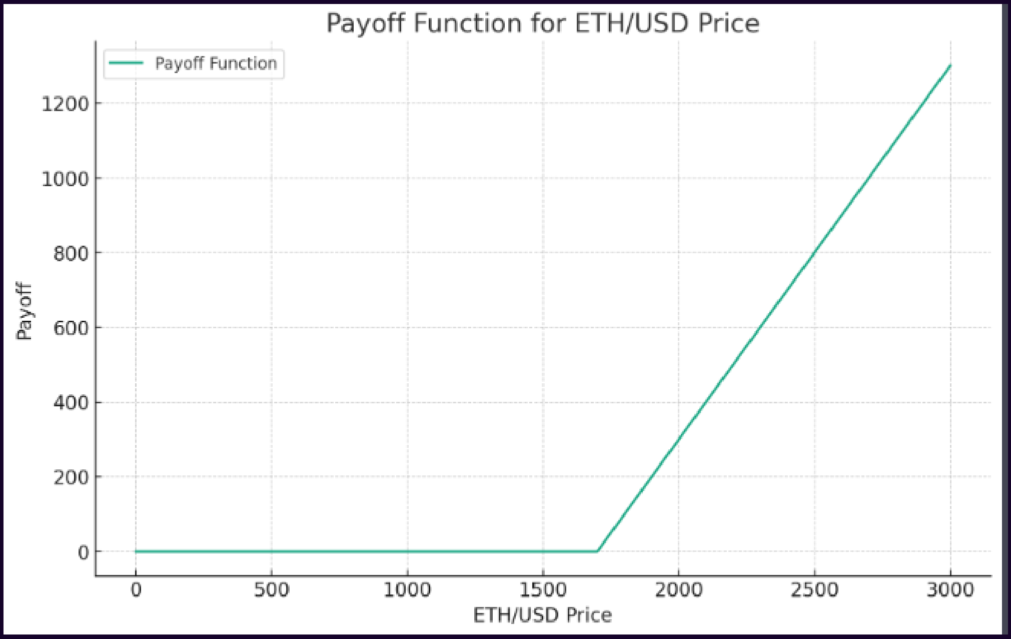
roguereddwarf

Fair points, and yeah there could be a small bid to get the price above 0.

Quoting from @jacksanford1 from a previous message.

> Ok, I agree with the premise that an extended DOS could result in unintended behavior like liquidations which the user may not have a chance to prevent when the DOS ends.
>
> How should we think about the realistic length of the DOS and the damage that could result to users because of it? Is this DOS going to last a few blocks maximum? Or a few months potentially?

Here's the exotic payoff function that can result in a DOS for a few months:



As the price goes below 1700 USD and stays there, the issue becomes permanent.

jacksanford1

Ok @SergeKireev this makes a very important distinction between "price returned by oracle" and "transformed price after the payoff function is applied."

I think the issue is saying that the transformed price is the one that is problematic if it hits zero. This would make more sense and could be a valid Medium.

@arjun-io Do you agree that the payoff function in the above graph is realistic?

arjun-io

@jacksanford1 It could be realistic since any payoff function is technically possible. I think the question here might be severity, since this isn't really a naturally occurring case and the payoff function would need to be explicitly designed in such a way that a `0` price is feasible for a long period of time.

SergeKireev

> Ok @SergeKireev this makes a very important distinction between "price returned by oracle" and "transformed price after the payoff function is

> applied."
>
> I think the issue is saying that the transformed price is the one that is problematic if it hits zero. This would make more sense and could be a valid Medium.
>
> @arjun-io Do you agree that the payoff function in the above graph is realistic?

Np, I don't think this changes our discussion above, I agree with the feasability of such a Payoff function.

It can make sense in implementing some derivative products such as options, as noted by @rogueredddwarf in latest comment. However these products don't work more broadly on perennial as they allow for unbounded sized positions with zero collateral (see my previous comment). As such, they wouldn't make sense in the context of a `BalancedVault` either, that's why I argue for this issue to be a low severity.

mstpr

regards to @rogueredddwarf payoff function,

I don't think anybody would deposit to such product that defines the payoff like this. Technically we can even create a payoff function where the price is always "0" but as long as there are no user depositing then nothing to worry about imo. Also, I think if price is ever "0" best thing to do is revert so even in such payoff it makes sense to revert.

jacksanford1

Agree with others that this payoff function is actually realistic and could mimic an options payoff, etc. So I'll assume there is a world where this exists and users deposit into it.

In that case, I'm sorry to drag this on, but I'd be interested in @rogueredddwarf's response to Serge's comment as probably the final thing holding this back from being Medium severity:

> However these products don't work more broadly on perennial as they allow for unbounded sized positions with zero collateral (see my previous comment). As such, they wouldn't make sense in the context of a BalancedVault either, that's why I argue for this issue to be a low severity.

rogueredddwarf

Isn't this then just saying more broadly that there are other things that break under the assumption of a realistic payoff function?

If what Serge is saying is correct, then part A of the protocol breaks before part B under the assumption of a realistic payoff function.

In the case that Serge is wrong, then part A does not break but part B breaks.

Either way there is an issue with realistic payoff functions (options payoff).

SergeKireev

> Isn't this then just saying more broadly that there are other things that break under the assumption of a realistic payoff function?
>
> If what Serge is saying is correct, then part A of the protocol breaks before part B under the assumption of a realistic payoff function.
>
> In the case that Serge is wrong, then part A does not break but part B breaks.
>
> Either way there is an issue with realistic payoff functions (options payoff).

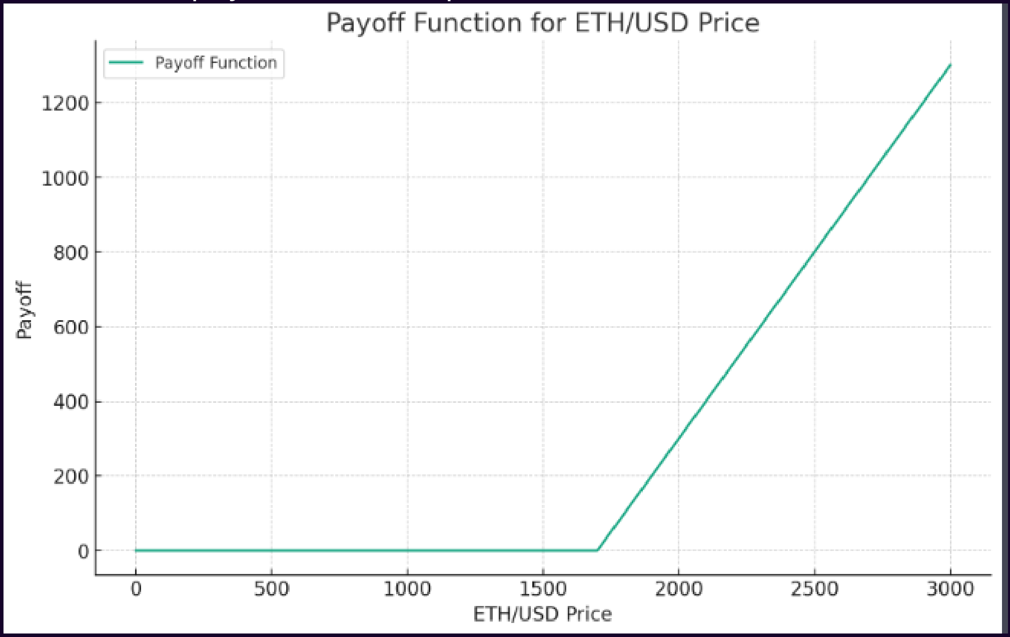
Yeah I agree with that, only thing is since it's not possible to create a functional market for this feasible payoff function, it is very unlikely somebody would do it at all.

Then this means that a BalancedVault created upon such a market would be even less likely to exist, which means that there is no use implementing the remediation suggested in this report, and the severity should be low

rogueredddwarf

Why is it not possible?

Assume the payoff function I posted above:

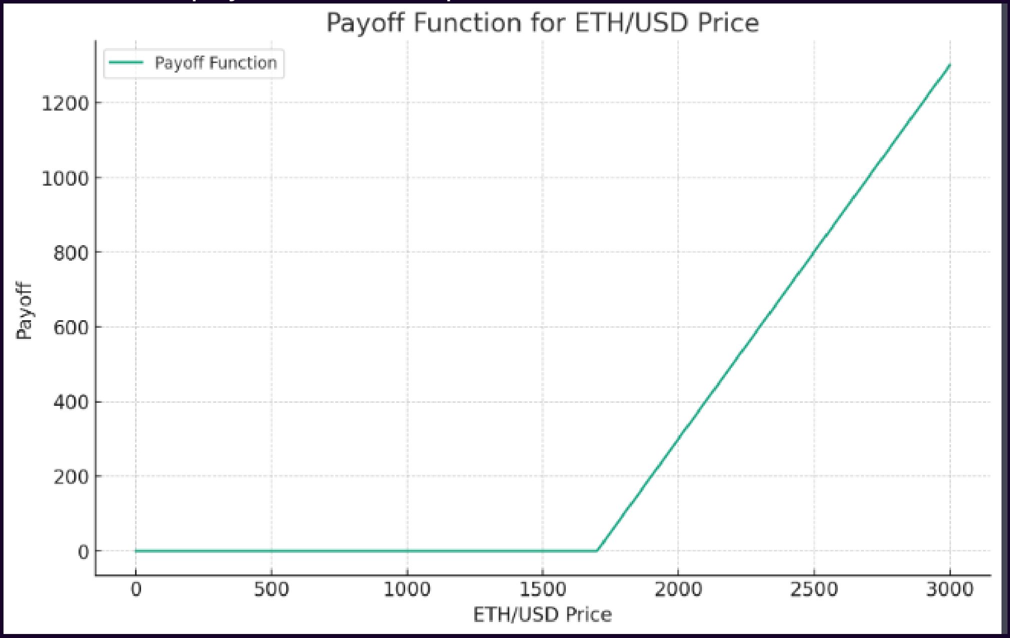


Assume the current price of ETH is 2000 USD.

Now the price drops below 1700 USD and we run into the issue.

Nothing prevents one from creating such a payoff function and it works fine as long as the price is above 1700 USD.

SergeKireev

> Why is it not possible?
>
> Assume the payoff function I posted above:
>
> 
>
>
> Assume the current price of ETH is 2000 USD.
>
> Now the price drops below 1700 USD and we run into the issue.
>
> Nothing prevents one from creating such a payoff function and it works fine as long as the price is above 1700 USD.

As I said earlier, the market would be completely broken, since it allows to take positions of any size without collateral

roguereddwarf

> @jacksanford1 With all due respect, discussing the details of the DOS in this case is missing the bigger picture;
>
> A market accepting a zero price is broken in more serious ways:
>
> During the period in which the oracle returns zero, any trader Alice can open positions of arbitrary size without having any collateral deposited since maintenance is zero: https://github.com/sherlock-audit/2023-05-perennial-SergeKireev/blob/c90b017ad432b283bcfbec594f80e18204ee98c3/perennial-mono/packages/perennial/contracts/product/types/position/AccountPosition.sol#L68-L73
>
> and collateral can be zero:

> https://github.com/sherlock-audit/2023-05-perennial-SergeKireev/blob/c90b017ad432b283bcfbec594f80e18204ee98c3/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L211-L227
>
> So if the oracle ever updates again to a non-zero price, Alice creates a shortfall of an arbitrary size on the protocol.
>
> In that regard, by reverting, `BalancedVault` has a more reasonable behavior than the underlying market when oracle price is zero.
>
> However `BalancedVault` links many heterogenous markets together, and in the scenario in which one fails, the whole vault fails and funds may stay locked up. I would argue that this report is an example of the broader #232 and thus should be a valid duplicate.

This is the previous comment you are referring to.

Maybe I have missed the broader picture in that I didn't realize that in addition to the DOS there are other problems as well (which as you rightfully point out are more serious).

However I identified the payoff function causing the issue in the first place and even the sponsor said this can be a realistic scenario.

As long as the price of ETH would not drop below 1700 USD in my example there would be no issue so this might only be discovered when it's too late.

I agree that this finding is probably best considered as a duplicate of #232.

Can we agree on this @SergeKireev @jacksanford1 ?

SergeKireev

Agree with this, thanks for acknowledging @roguereddwarf

jacksanford1

Result: Medium Unique It seems like this issue is likely enough and severe enough to be considered a Medium. However I don't believe it is similar enough to #232 to be considered a duplicate, so it will be a unique Medium.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- roguereddwarf: accepted
- SergeKireev: rejected

arjun-io

Fixed via a comment: https://github.com/equilibria-xyz/perennial-mono/pull/203

Issue M-5: If long and short products has different maker fees, vault rebalance can be spammed to eat vaults balance

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/41

Found by

mstpr-brainbot

Summary

In a market scenario where long and short products carry different position fees, continuous rebalancing could potentially deplete the vault's funds. This situation arises when closing positions incurs a maker fee, which gradually eats away at the collateral balance. The rebalancing cycle continues until the fee becomes insignificant or the collateral balance is exhausted.

Vulnerability Detail

If a market's long and short products have different position fees, continuous rebalancing could be an issue. A significant position fee difference might deplete all funds after repeated rebalancing.

Consider a scenario with one market, where the long product has a 0% maker fee (position fee), and the short product a 10% maker fee. A vault holding 100 positions divided across these markets, with 50 short and 50 long positions, and collateral balances of 50-50 each, is presented.

Now, suppose the vault and the product are entirely in sync (meaning they're of the same version). If someone deposits 100 assets into the vault, it would distribute that deposit, adding 50 to both product's collateral balances, bringing them to 100 each. The vault then increases its position to 100 to maintain the 1x leverage. The long market would match this since it has no maker fee, but the short market would lose 5 from its collateral due to the fee charged by the openMake operation. As a result, the market now has 95 in the short market and 100 in the long market collateral. This imbalance triggers a rebalancing operation via sync().

After the second call to sync(), the collateral balances would be 97.5 in both markets, and the markets would reduce their positions to maintain the 1x leverage. The long market would reduce its position to 97.5, but when the short market does the same, the closeMake operation's fee charge reduces the short market collateral by 0.25, resulting in final balances of 97.5 in the long market and 97.25 in the short market. This imbalance again allows a call to settle...

As seen if the difference between the products are high the rebalance will start eating the vaults collateral balance by paying redundant maker fees.

The above scenario assumes only one market with two products. If there are two markets, the situation could be more troublesome due to the additional fee charges and potentially higher losses.

Impact

Since the vaults funds can be drained significantly if certain cases are met (high makerFee set by market owner and big collateral balance differences between markets or directional exposure on vault causing the collateral balance differences higher than usual so more funds to rebalance hence more fee to pay). I'll label it as high.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L137-L149

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L426-L533

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L341-L355

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L147-L153

Tool used

Manual Review

Recommendation

Consider accounting the positionFee if there are any on any of the markets or assert that both of the markets has the same positionFee.

Discussion

arjun-io

This is a good call - currently vault's don't support handling maker fees in a clean way. We probably won't fix this in the current version of the vaults but it's something we'd likely address in the future as maker fees might become necessary in certain markets

mstpr

Escalate for 10 USDC

I think this is a valid high since it can drain the vaults balance significantly.

As stated in the impact section of the issue, if the price movement to one side is high, there will be a difference between the vaults short and long products collateral balance hence, the more fee to apply. Therefore, calling rebalancing few more times will result in more losses and this will start taking the funding fees that the vault generating. Although at some point it will converge to very small numbers, this will lead to loss of funds for the vault depositors.

sherlock-admin

> Escalate for 10 USDC
>
> I think this is a valid high since it can drain the vaults balance significantly.
>
> As stated in the impact section of the issue, if the price movement to one side is high, there will be a difference between the vaults short and long products collateral balance hence, the more fee to apply. Therefore, calling rebalancing few more times will result in more losses and this will start taking the funding fees that the vault generating. Although at some point it will converge to very small numbers, this will lead to loss of funds for the vault depositors.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

Hmm... I think there is a case for this to be high severity as this will result in a continuous eating up of user funds, if vaults for products with maker fees are launched. On the other hand, Perennial doesn't have any products with maker fees at the moment, and so Perennial says that this current iteration of vaults was intentionally not including the handling of that.

arjun-io

Similar to #43 this is intended as the added complexity to support something that wasn't likely to be turned on wasn't worth it. These two can likely be judged similarly, although I will say for this issue there is a malicious attack vector where other makers can force the vault to rebalance and thus generate maker fees which they get a cut of, this would necessitate quicker action to add some sort of rebalance/vault fee

jacksanford1

Seems like there's a chance that a significant amount of funds could be lost due to coordinated effort here.

There are enough contingencies to make it a Medium:

> high makerFee set by market owner and big collateral balance differences between markets or directional exposure on vault causing the collateral balance differences higher than usual so more funds to rebalance hence more fee to pay

And enough of a risk of material funds being lost in a coordinated attack scenario.

jacksanford1

Result: Medium Unique See previous comment.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- mstpr: rejected

Issue M-6: BalancedVault.sol: Early depositor can manipulate exchange rate and steal funds

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/46

Found by

Phantasmagoria, roguereddwarf

Summary

The first depositor can mint a very small number of shares, then donate assets to the Vault. Thereby he manipulates the exchange rate and later depositors lose funds due to rounding down in the number of shares they receive.

The currently deployed Vaults already hold funds and will merely be upgraded to V2. However as Perennial expands there will surely be the need for more Vaults which enables this issue to occur.

Vulnerability Detail

You can add the following test to `BalancedVaultMulti.test.ts`. Make sure to have the `dsu` variable available in the test since by default this variable is not exposed to the tests.

The test is self-explanatory and contains the necessary comments:

```
it('exchange rate manipulation', async () => {
      const smallDeposit = utils.parseEther('1')
      const smallestDeposit = utils.parseEther('0.000000000000000001')

      // make a deposit with the attacker. Deposit 1 Wei to mint 1 Wei of shares
      await vault.connect(user).deposit(smallestDeposit, user.address)
      await updateOracle();
      await vault.sync()

      console.log(await vault.totalSupply());

      // donating assets to Vault
      await dsu.connect(user).transfer(vault.address, utils.parseEther('1'))

      console.log(await vault.totalAssets());

      // make a deposit with the victim. Due to rounding the victim will end up
↪   with 0 shares
      await updateOracle();
```

SHERLOCK

```
      await vault.sync()
      await vault.connect(user2).deposit(smallDeposit, user2.address)
      await updateOracle();
      await vault.sync()

      console.log(await vault.totalAssets());
      console.log(await vault.totalSupply());
      // the amount of shares the victim receives is rounded down to 0
      console.log(await vault.balanceOf(user2.address));

      /*
      at this point there are 2000000000000000001 Wei of assets in the Vault and
    ↪ only 1 Wei of shares
      which is owned by the attacker.
      This means the attacker has stolen all funds from the victim.
      */
    })
```

Impact

The attacker can steal funds from later depositors.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L775-L778

Tool used

Manual Review

Recommendation

This issue can be mitigated by requiring a minimum deposit of assets. Thereby the attacker cannot manipulate the exchange rate to be so low as to enable this attack.

Discussion

arjun-io

The inflation attack has been reported and paid out on Immunefi (happy to provide proof here if needed) - we have added a comment describing this attack here: https://github.com/equilibria-xyz/perennial-mono/pull/194

mstpr

Escalate for 10 USDC I think this is an informational issue.

The finding is correct. However, in order this to be applicable, the first depositor needs to be the only depositor in the epoch (first epoch) they deposited. So, it is way harder to pull off this attack than a regular 4626 vault. Considering oracles are updating every 3 hours minimum (heartbeat of chainlink, assuming no price deviation) the attacker needs to be the first depositor for the epoch not the actual first depositor. Protocol team can easily deposit some considerable amount after deployment and mitigate this attack.

sherlock-admin

> Escalate for 10 USDC I think this is an informational issue.
>
> The finding is correct. However, in order this to be applicable, the first depositor needs to be the only depositor in the epoch (first epoch) they deposited. So, it is way harder to pull off this attack than a regular 4626 vault. Considering oracles are updating every 3 hours minimum (heartbeat of chainlink, assuming no price deviation) the attacker needs to be the first depositor for the epoch not the actual first depositor. Protocol team can easily deposit some considerable amount after deployment and mitigate this attack.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

roguereddwarf

Escalate for 10 USDC

I think this is a valid Medium and disagree with the first escalation.

First I'd like to comment on the issue that was submitted via Immunefi that the sponsor has linked to. The commit has been made on June 16th and the contest ended on June 15th. So it is clear that I did not just copy the attack vector from the repo, and this is indeed a valid finding. (Just want to make sure there can be no suspicion of me copying the issue)

Furthermore the first escalation explains that this is a tricky attack scenario. While true, there is a valid scenario which can occur with realistic on-chain conditions as new Vaults get deployed.

Also the first escalation pointed out that this could be mitigated by seeding the Vault with some initial funds. While this could be a solution, clearly this is not something the sponsor had in mind, specifically since the issue was accepted on Immunefi and there is no mention of seeding the Vault in the docs.

What remains therefore is a valid attack path (even though unlikely) leading to a loss of funds. So I think this should be a valid Medium.

sherlock-admin

> Escalate for 10 USDC
>
> I think this is a valid Medium and disagree with the first escalation.
>
> First I'd like to comment on the issue that was submitted via Immunefi that the sponsor has linked to. The commit has been made on June 16th and the contest ended on June 15th. So it is clear that I did not just copy the attack vector from the repo, and this is indeed a valid finding. (Just want to make sure there can be no suspicion of me copying the issue)
>
> Furthermore the first escalation explains that this is a tricky attack scenario. While true, there is a valid scenario which can occur with realistic on-chain conditions as new Vaults get deployed.
>
> Also the first escalation pointed out that this could be mitigated by seeding the Vault with some initial funds. While this could be a solution, clearly this is not something the sponsor had in mind, specifically since the issue was accepted on Immunefi and there is no mention of seeding the Vault in the docs.
>
> What remains therefore is a valid attack path (even though unlikely) leading to a loss of funds. So I think this should be a valid Medium.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

mstpr

Thinking more on this, I think I agree this is a valid medium. Although it is harder to make this attack because of the epoch things it is still almost free for attacker to try. So, attacker can just deposit 1 Wei and hope they're the first depositor.

Not deleting my escalation just in case @roguereddwarf escalation stands unresponded and lead him to lose 10 USDC.

KenzoAgada

Issue should remain medium. The escalator also agrees to that, as seen in the previous comment.

jacksanford1

Result: Medium Has duplicates See mstpr comment above.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- roguereddwarf: accepted
- mstpr: rejected

arjun-io

As stated above we've updated the comment to reflect share inflation is possible: https://github.com/equilibria-xyz/perennial-mono/pull/194

We won't be adding a solidity level fix at this time, but we will update our deploy scripts to create an initial deposit

Issue M-7: BalancedVault.sol: claim can be impossible due to unsigned integer underflow

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/57

Found by

roguereddwarf

Summary

When a user redeems his shares, he then needs to call the `claim` function to actually claim his assets.

This will downstream call the `_rebalancePosition` function with the `claimAmount` as an argument.

Under some circumstances the `_rebalancePosition` function can revert due to unsigned integer underflow, making it impossible to claim assets.

This condition is not permanent but can only be solved by sending additional assets to the Vault causing a loss of funds.

Vulnerability Detail

Assume the following situation:

```
_assets() = 5e18
_totalUnclaimed = 10e18

unclaimed of user1 = 5e18
```

When user1 calls `claim`, pro-rata claiming applies and the actual `claimAmount` is equal to `5e18*10e18/10e18=5e18`.

Now let's look at the vulnerable line: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L474

`_totalAssetsAtEpoch` would be equal to `0`: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L684-L690

Thereby the line in `_rebalancePosition` reverts due to unsigned integer underflow (`0 - claimAmount`),

You can add the following test to `BalancedVaultMulti.test.ts` which is a modified version of the existing `gracefully unwinds upon insolvency` test:

```
it('reverts upon pro rata claim', async () => {
        // 1. Deposit initial amount into the vault
        await vault.connect(user).deposit(utils.parseEther('50000'),
↪   user.address)
        await vault.connect(user2).deposit(utils.parseEther('50000'),
↪   user2.address)
        await updateOracle()
        await vault.sync()

        // 2. Redeem most of the amount, but leave it unclaimed

        await vault.connect(user).redeem(utils.parseEther('40000'), user.address)
        await vault.connect(user2).redeem(utils.parseEther('20000'),
↪   user2.address)
        await updateOracle()
        await vault.sync()

        // 3. An oracle update makes the long position liquidatable, initiate
↪   take close
        await updateOracle(utils.parseEther('20000'))
        await long.connect(user).settleAccount(vault.address)
        await short.connect(user).settleAccount(vault.address)
        await long.connect(perennialUser).closeTake(utils.parseEther('700'))
        await collateral.connect(liquidator).liquidate(vault.address,
↪   long.address)

        // // 4. Settle the vault to recover and rebalance
        await updateOracle() // let take settle at high price
        await updateOracle(utils.parseEther('1500')) // return to normal price
↪   to let vault rebalance
        await vault.sync()
        await updateOracle()
        await vault.sync()

        // 6. Claim should be pro-rated
        await vault.claim(user2.address)
})
```

Impact

The user is at least temporarily unable to claim his assets and the situation needs to be solved by sending additional assets to the Vault (of which the user most likely does not get the majority back with the claim) -> loss of funds

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L211-L228

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L472-L492

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L684-L690

Tool used

Manual Review

Recommendation

In the `_rebalancePosition` function it must be ensured that the following line does not execute when `claimAmount > _totalAssetsAtEpoch(context)`:
https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L474

Discussion

sherlock-admin

> Escalate for 10 USDC I also noticed this issue, thanks to roguereddwarf for good explanation. The reasons for not submitting are as follows:
>
> 1. This situation is not permanent. `_rebalance` is called by multiple functions, but only in the `claim` function that `claimAmount` is not 0. So it will not affect `syncAccount/deposit/redeem`.
> 2. This happens only after the vault is liquidated. This does make the user lose funds (unable to redeem) if no funds are subsequently deposited(from other users). But shouldn't this user take the risk of liquidation?
>
> So this is not a valid M. I would like to hear everyone's opinion.

```
You've deleted an escalation for this issue.
```

roguereddwarf

Escalate for 10 USDC

I disagree with the first escalation and argue that this should remain a Medium severity issue. The situation is not permanent but can only be resolved by the user by sending additional funds (of which the user most likely does not get the majority back because they will also be allocated to other users).

And yes this only happens when the Vault got liquidated as can be seen from my test case as well. But this does not take away from the fact that this is a valid scenario. Liquidations are part of the normal operation of the Vault, I'm not making any unrealistic assumptions in this report.

Also the user DOES take the risk of liquidation which is reflected by only being able to claim pro-rata. But as I have showed the claiming fails altogether so he cannot claim the funds that he is entitled to after the liquidation.

sherlock-admin

> Escalate for 10 USDC
>
> I disagree with the first escalation and argue that this should remain a Medium severity issue. The situation is not permanent but can only be resolved by the user by sending additional funds (of which the user most likely does not get the majority back because they will also be allocated to other users).
>
> And yes this only happens when the Vault got liquidated as can be seen from my test case as well. But this does not take away from the fact that this is a valid scenario. Liquidations are part of the normal operation of the Vault, I'm not making any unrealistic assumptions in this report.
>
> Also the user DOES take the risk of liquidation which is reflected by only being able to claim pro-rata. But as I have showed the claiming fails altogether so he cannot claim the funds that he is entitled to after the liquidation.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

securitygrid

You convinced me, I agree. Thanks. @roguereddwarf

KenzoAgada

The original escalation has been deleted. All that remains is roguerddwarf's escalation which is valid (as it explained why original escalation was invalid).

No change is needed in issue's status.

jacksanford1

Ok, should remain a Medium.

jacksanford1

Result: Medium Unique Escalation was deleted.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- roguereddwarf: accepted

arjun-io

Fixed: https://github.com/equilibria-xyz/perennial-mono/pull/205

Issue M-8: Malicious trader can bypass utilization buffer

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/75

Found by

ast3ros, cergyk

Summary

A malicious trader can bypass the Utilisation buffer, and push utilisation to 1 on any product.

Vulnerability Detail

Perenial products use a utilization buffer to prevent taker side to DOS maker by taking up all the liquidity: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L547-L556

Makers would not be able to withdraw if utilization would reach 100% because of `takerInvariant` which is enforced during `closeMake`: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544

A malicious trader can bypass the utilisation buffer by first opening a maker position, open taker position and close previous maker position. The only constraint is that she has to use different accounts to open the maker positions and the taker positions, since perennial doesn't allow to have maker and taker positions on a product simultaneously.

So here's a concrete example:

Let's say the state of product is `900 USD` on the maker side, `800 USD` on the taker side, which respects the 10% utilization buffer.

Example

Using account 1, Alice opens up a maker position for `100 USD`, bringing maker total to `1000 USD`.

Using account 2, Alice can open a taker position for `100 USD`, bringing taker to `900 USD`, which respects the 10% utilization buffer still.

Now using account 1 again, Alice can close her `100 USD` maker position and withdraw collateral, clearing account 1 on perennial completely.

This brings the utilization to `100%`, since taker = maker = `900 USD`.

> This is allowed, since only `takerInvariant` is checked when closing a maker position, which enforces that utilization ratio is lower than or equal to `100%`.

Impact

Any trader can bring the utilization up to 100%, and use that to DoS withdrawals from Products and Balanced vaults for an indefinite amount of time. This is especially critical for vaults, since when any product related to any market is fully utilized, all redeems from the balanced vault are blocked.

Code Snippet

Tool used

Manual Review

Recommendation

If a safety buffer needs to be enforced for the utilisation of products, it needs to be enforced on closing make positions as well.

Discussion

arjun-io

This is a good callout and a valid issue but since the *impact* existed before this utilization buffer feature was added we disagree with the severity. The utilization buffer is not intended to totally fix the 100% utilization issue against malicious users, but instead provide a buffer in the normal operating case.

KenzoAgada

In addition to the sponsor's comment, also worth noting that a malicious user will not gain anything from doing this. Indeed seems like medium severity is more appropriate. Downgrading to medium.

sherlock-admin

> Escalate for 10 USDC
>
> The assumption here is that Alice can close position immediately. However, in order to close a position Alice needs to wait the settlement and the meanwhile Alice can occur losses or profit. This is not a safe operation for Alice to do. Alice can do this if she is willing to take the risk. I think the utilization buffer works as intented in this scenario, where it forces Alice to create a maker position in order to create a taker position

> first and since Alice can't withdraw immediately she is subject to her maker positions pnl for at least an oracle version.
>
> I think this an invalid issue

```
You've deleted an escalation for this issue.
```

SergeKireev

> The assumption here is that Alice can close position immediately. However, in order to close a position Alice needs to wait the settlement and the meanwhile Alice can occur losses or profit. This is not a safe operation for Alice to do. Alice can do this if she is willing to take the risk. I think the utilization buffer works as intented in this scenario, where it forces Alice to create a maker position in order to create a taker position first and since Alice can't withdraw immediately she is subject to her maker positions pnl for at least an oracle version.
>
> I think this an invalid issue

The described operation can be done atomically because the invariant `takerInvariant` which is bypassed here checks only next amounts, and so Alice does not in fact need to expose herself to the market (She can just fill all maker capacity before Bob's transaction, and close after the tx and before next settlement) https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544

mstpr

>> The assumption here is that Alice can close position immediately. However, in order to close a position Alice needs to wait the settlement and the meanwhile Alice can occur losses or profit. This is not a safe operation for Alice to do. Alice can do this if she is willing to take the risk. I think the utilization buffer works as intented in this scenario, where it forces Alice to create a maker position in order to create a taker position first and since Alice can't withdraw immediately she is subject to her maker positions pnl for at least an oracle version.
>>
>> I think this an invalid issue
>
> The described operation can be done atomically because the invariant `takerInvariant` which is bypassed here checks only next amounts, and so Alice does not in fact need to expose herself to the market (She can just fill all maker capacity before Bob's transaction, and close after the tx and before next settlement) https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544

You are correct, sorry for the misunderstanding of the scenario you described above. Deleting my escalations comment

mstpr

Escalate for 10 USDC

I rethought of the issue and I think this is a low issue.

Issue does not add any benefits to the user and in fact, it is even discouraging for the user to not take this action due to the funding fee increase. Since the funding fee follows a curve model the funding fee will be taken at 100% utilization is quite high and it is not a good thing that any taker would want. There are no funds in danger and there isn't any benefits of doing such thing.

sherlock-admin

> Escalate for 10 USDC
>
> I rethought of the issue and I think this is a low issue.
>
> Issue does not add any benefits to the user and in fact, it is even discouraging for the user to not take this action due to the funding fee increase. Since the funding fee follows a curve model the funding fee will be taken at 100% utilization is quite high and it is not a good thing that any taker would want. There are no funds in danger and there isn't any benefits of doing such thing.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

Hmm... Rethinking about the issue, I admit it might not be enough for medium severity. Leaving for Sherlock to decide.

jacksanford1

Impact from issue:

> Any trader can bring the utilization up to 100%, and use that to DoS withdrawals from Products and Balanced vaults for an indefinite amount of time. This is especially critical for vaults, since when any product related to any market is fully utilized, all redeems from the balanced vault are blocked.

Based on Sherlock's DOS rules: https://docs.sherlock.xyz/audits/judging/judging#some-standards-observed

Doesn't seem like this should be a valid Medium. @cergyk would need to explain how the attack can be profitable for the attacker or show a difference between cost incurred and damage created that is many orders of magnitude.

SergeKireev

> @CergyK would need to explain how the attack can be profitable for the attacker or show a difference between cost incurred and damage created that is many orders of magnitude.

Due to the nature of BalancedVault, multiple markets of different nature are linked together. If one is fully utilized, it blocks the BalancedVault for all of the markets as stated in the initial report:

> This is especially critical for vaults, since when any product related to any market is fully utilized, all redeems from the balanced vault are blocked.

So there can indeed be a magnitude between cost incurred and damage created.

Suppose Balanced Vault on markets:

- $.PEPEwith100$ worth of liquidity
- $.ETHwith1.000.000$ of liquidity

Malicious user Alice would only have to fully utilize 100$ on the $.PEPEmarkettoblock1.000.000$ in redeemable assets on $ETH because the two markets are linked by the BalancedVault

jacksanford1

Thanks @SergeKireev. Agree in that case the cost could be very low for a DOS of significant magnitude. I don't think it's a High, but it seems like it's worthwhile to make this a Medium.

Open to arguments from @mstpr on why it should be Low instead.

jacksanford1

Result: Medium Has duplicates Based on argument from Serge and no follow-up response from mstpr, leaving this as a Medium.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- mstpr: rejected

Issue M-9: A trader close to liquidation risks being liquidated by trying to reduce her position

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/76

Found by

cergyk

Summary

A trader close to liquidation risks being liquidated by trying to reduce her position

Vulnerability Detail

We can see in the implementation of closeTake/closeMake that the `maintenanceInvariant` is not enforced: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L246-L256

and the takerFee is deducted immediately from traders collateral, whereas the position reduction only takes effect at next settlement: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L258-L273

This means that a legitimate user close to liquidation, who may want to reduce leverage by closing position will end up immediately liquidatable.

Example:

> Maintenance factor is 2% Liquidation fee is 5% Taker fee is 0.1% Funding fee is 0%

Alice has a 50x leverage `taker` position opened as authorized per maintenance invariant:

Alice collateral: 100 DSU Alice taker position: notional of 5000 DSU

Alice tries to reduce her position to a more reasonable 40x, calls `closeTake(1000)`. A 1 DSU taker fee is deducted immediately from her collateral. Since the reduction of the position only takes effect on next oracle version, her position becomes immediately liquidatable. A liquidator closes it entirely, making Alice lose an additional 4 DSU `takerFee` on next settlement and 250 DSU of liquidationFee.

Impact

A user trying to avoid liquidation is liquidated unjustly

Code Snippet

Tool used

Manual Review

Recommendation

Either:

- Do not deduce takerFee from collateral immediately, but at next settlement, as any other fees or
- Enforce `maintenanceInvariant` on closing of positions, to make transaction revert (except when closing is liquidation already)

Discussion

SergeKireev

Escalate for 10 USDC

Disagree with downgrading of severity. Although it can be argued the loss of funds is self-inflicted, reducing the position when near liquidation is the most reasonable choice to avoid liquidation. Instead of avoiding the liquidation, the user makes himself instantly liquidatable and risks losing 5% to 10% of collateral, which can be an unbounded loss of funds.

sherlock-admin

> Escalate for 10 USDC
>
> Disagree with downgrading of severity. Although it can be argued the loss of funds is self-inflicted, reducing the position when near liquidation is the most reasonable choice to avoid liquidation. Instead of avoiding the liquidation, the user makes himself instantly liquidatable and risks losing 5% to 10% of collateral, which can be an unbounded loss of funds.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

sherlock-admin

> Escalate for 10 USDC This is not valid M. In this situation on the verge of liquidation, no matter what you do, if the price moves in the direction of loss, the liquidation will occur. On the contrary, the position is safe. This is a normal phenomenon. The safe thing to do is to increase the margin.

```
You've deleted an escalation for this issue.
```

mstpr

Escalate for 10 USDC

I think this issue is invalid or informational.

First of all, that's a great point. However, I feel like that the users should be aware of this already. Maker/taker fees are actually the open/close position fees which they should be aware all the time. So, if a user closes some of their position they should know that there is a fee that they need to pay. If that fee takes them to liquidation level, this should be ok because that's how the system works.

Here how I see the case: In a product with both open/close position fees; If a user with a perfectly healthy position wants to leverage its position, they will pay an open position fee which is acceptable. When the same users wants to deleverage, it's also acceptable to take the close position fee. However, just because the deleverage now puts the user in liquidation, why is it not acceptable to take the close position fees? I think this is how the system works. Just because user is liquidatable after the position fees shouldn't mean that the protocol should not take fees from their action.

sherlock-admin

> Escalate for 10 USDC
>
> I think this issue is invalid or informational.
>
> First of all, that's a great point. However, I feel like that the users should be aware of this already. Maker/taker fees are actually the open/close position fees which they should be aware all the time. So, if a user closes some of their position they should know that there is a fee that they need to pay. If that fee takes them to liquidation level, this should be ok because that's how the system works.
>
> Here how I see the case: In a product with both open/close position fees; If a user with a perfectly healthy position wants to leverage its position, they will pay an open position fee which is acceptable. When the same users wants to deleverage, it's also acceptable to take the close position fee. However, just because the deleverage now puts the user in liquidation, why is it not acceptable to take the close position fees? I think this is how the system works. Just because user is liquidatable after the position fees shouldn't mean that the protocol should not take fees from their action.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

SergeKireev

> Escalate for 10 USDC
>
> I think this issue is invalid or informational.
>
> First of all, that's a great point. However, I feel like that the users should be aware of this already. Maker/taker fees are actually the open/close position fees which they should be aware all the time. So, if a user closes some of their position they should know that there is a fee that they need to pay. If that fee takes them to liquidation level, this should be ok because that's how the system works.
>
> Here how I see the case: In a product with both open/close position fees; If a user with a perfectly healthy position wants to leverage its position, they will pay an open position fee which is acceptable. When the same users wants to deleverage, it's also acceptable to take the close position fee. However, just because the deleverage now puts the user in liquidation, why is it not acceptable to take the close position fees? I think this is how the system works. Just because user is liquidatable after the position fees shouldn't mean that the protocol should not take fees from their action.

Great point as well. This report does not argue that the protocol should not take a fee in that case, just that it should be taken at next settlement (like funding fees for example). The user reduces position, fee is deducted upon next settlement, and the user is never liquidatable, everybody is happy

The situation described in this report arises from the fact that collateral handling is not consistent: open/close fee is subtracted right away, and all other collateral/position modifications occur at next settlement

KenzoAgada

I think the various points above show why a medium severity is appropriate.

On one hand it is self inflicted, limited in scope, and depends upon a user not leaving any room for error (price movements/fees) at all, so I don't think it deserves high severity.

On the other hand, it is unintuitive that partially closing a dangerous position should make you liquidatable right away. I think there *is* a seeming inconsistency in the fact that position fees are deducted right away and can make a position that was already safe in the current version liquidatable. When withdrawing collateral there is a check that the user has enough collateral for maintenance of current version, but there's no such check in the scenario this issue describes.

I would maintain medium severity.

jacksanford1

@mstpr Thoughts on the above comments? Seems like it's the ordering of not taking the fee at next settlement?

mstpr

@jacksanford1 I would say it is probably not possible or not a secure way to do that. I would love to hear the protocol teams response on that since taking the fee after settlement or before settlement is their design choice. In my opinion, it looks too risky or undoable to take the fee after settlement so that's why they went for this implementation.

SergeKireev

@jacksanford1 @mstpr additional remarks:

- It is understood that unifying this fees handling is a bit more than a trivial fix. A temporary solution could just to add an invariant which checks if the user becomes liquidatable by doing this call and revert in that case.
- There is another really weird quirk with current implementation: If the user goes through the scenario outlined in the report, and closes his position entirely, the liquidator just takes some of the collateral without providing any utility.

Indeed the positions are already being closed, so this call is a noop: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L362-L372

The liquidator does not help the protocol but takes a potentially hefty fee at the expense of an innocent user.

jacksanford1

@mstpr @SergeKireev I hear your points on the difficulty of the fix, but the difficulty of the fix should not have an effect on the severity of the issue.

Is there an argument for why this issue is invalid? @mstpr

And what is the argument for High vs. Medium @SergeKireev (after seeing Kenzo's latest comment)?

SergeKireev

> @mstpr @SergeKireev I hear your points on the difficulty of the fix, but the difficulty of the fix should not have an effect on the severity of the issue.
>
> Is there an argument for why this issue is invalid? @mstpr
>
> And what is the argument for High vs. Medium @SergeKireev (after seeing Kenzo's latest comment)?

My argument for saying this should be `High` is an assessment of impact and probability:

- Impact: Loss of user funds for a large amount (The whole amount of collateral of the user).
- Probability: If the user is on the brink of liquidation, closing or reducing her position seems like the most sensible action. With regards to latest @KenzoAgada comment, a user does not necessarily need to open a 50x (max leverage) position, he can also get into this situation simply by accumulating negative uPnl

mstpr

I am still seeing this issue as a design choice. Users are aware that there is a maker fee and it's updatable by the protocol team. Traders should always be cautious on their position and maintain it, that's their responsibility and that's a risk they took by taking the position. If user doesn't want to get liquidated, they can add some collateral and deleverage in the current design. It's not that the user has no other ways to dodge liquidation. @jacksanford1 @SergeKireev

jacksanford1

Thanks @SergeKireev @mstpr

I see it pretty clearly as a Medium. This is not a "blackhat can exploit the whole protocol" type of magnitude, it's just one user who is likely already near the brink of liquidation.

But I think the issue is real and worth potentially fixing (or at least knowing about) because it could have a very negative unintended effect for a user who wants to derisk their position.

jacksanford1

Result: Medium Unique Please see above comment for full reasoning.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- SergeKireev: rejected
- mstpr: rejected

Issue M-10: User would liquidate his account to sidestep `takerInvariant` modifier

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/77

Found by

Emmanuel, branch_indigo

Summary

A single user could open a massive maker position, using the maximum leverage possible(and possibly reach the maker limit), and when a lot of takers open take positions, maker would liquidate his position, effectively bypassing the taker invariant and losing nothing apart from position fees. This would cause takers to be charged extremely high funding fees(at the maxRate), and takers that are not actively monitoring their positions will be greatly affected.

Vulnerability Detail

In the closeMakeFor function, there is a modifier called `takerInvariant`.

```
function closeMakeFor(
        address account,
        UFixed18 amount
    )
        public
        nonReentrant
        notPaused
        onlyAccountOrMultiInvoker(account)
        settleForAccount(account)
        takerInvariant
        closeInvariant(account)
        liquidationInvariant(account)
    {
        _closeMake(account, amount);
    }
```

This modifier prevents makers from closing their positions if it would make the global maker open positions to fall below the global taker open positions. A malicious maker can easily sidestep this by liquidating his own account. Liquidating an account pays the liquidator a fee from the account's collateral, and then forcefully closes all open maker and taker positions for that account.

```
function closeAll(address account) external onlyCollateral notClosed
↪  settleForAccount(account) {
        AccountPosition storage accountPosition = _positions[account];
        Position memory p =
↪   accountPosition.position.next(_positions[account].pre);

        // Close all positions
        _closeMake(account, p.maker);
        _closeTake(account, p.taker);

        // Mark liquidation to lock position
        accountPosition.liquidation = true;
    }
```

This would make the open maker positions to drop significantly below the open taker position, and greatly increase the funding fee and utilization ratio.

ATTACK SCENARIO

- A new Product(ETH-Long) is launched on arbitrum with the following configurations:
    - 20x max leverage(5% maintenance)
    - makerFee = 0
    - takerFee = 0.015
    - liquidationFee = 20%
    - minRate = 4%
    - maxRate = 120%
    - targetRate = 12%
    - targetUtilization = 80%
    - makerLimit = 4000 Eth
    - ETH price = 1750 USD
    - Coll Token = USDC
    - max liquidity(USD) = 4000*1750 = $7,000,000
- Whale initially supplies 350k USDC of collateral(~200ETH), and opens a maker position of 3000ETH($5.25mn), at 15x leverage.
- After 2 weeks of activity, global open maker position goes up to 3429ETH($6mn), andbecausefundingFeeislow, peopleareincentivizedtoopentakerpositions, sogloba$ at 80% utilization. Now, rate of fundingFee is 12%

- Now, Whale should only be able to close up to 686ETH($1.2mn) of his maker position using the `closeMakeFor` function because of the `takerInvariant` modifier.
- Whale decides to withdraw 87.5k USDC(~50ETH), bringing his total collateral to 262.5k USDC, and his leverage to 20x(which is the max leverage)
- If price of ETH temporarily goes up to 1755 USD, totalMaintenance=3000 * 1755 * 5% = $263250. Because his totalCollateral is 262500 USDC(which is less than totalMaintenance), his account becomes liquidatable.
- Whale liquidates his account, he receives liquidationFee*totalMaintenance = 20% * 263250 = 52650USDC, and his maker position of 3000ETH gets closed. Now, he can withdraw his remaining collateral(262500-52650=209850)USDC because he has no open positions.
- Global taker position is now 2743ETH($4.8mn), andglobalmakerpositionis429ETH$(750k)
- Whale has succeeded in bypassing the takerInvaraiant modifier, which was to prevent him from closing his maker position if it would make global maker position less than global taker position.

Consequently,

- Funding fees would now be very high(120%), so the currently open taker positions will be greatly penalized, and takers who are not actively monitoring their position could lose a lot.
- Whale would want to gain from the high funding fees, so he would open a maker position that would still keep the global maker position less than the global taker position(e.g. collateral of 232750USDC at 15x leverage, open position = ~2000ETH($3.5mn)) so that taker positions will keep getting charged at the funding fee maxRate.

Impact

User will close his maker position when he shouldn't be allowed to, and it would cause open taker positions to be greatly impacted. And those who are not actively monitoring their open taker positions will suffer loss due to high funding fees.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L123-L132 https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L362-L372 https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/

Product.sol#L334 https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L535-L544

Tool used

Manual Review

Recommendation

Consider implementing any of these:

- Protocol should receive a share of liquidation fee: This would disincentivize users from wanting to liquidate their own accounts, and they would want to keep their positions healthy and over-collateralized
- Let there be a maker limit on each account: In addition to the global maker limit, there should be maker limit for each account which may be capped at 5% of global maker limit. This would decentralize liquidity provisioning.

Discussion

arjun-io

This is working as intended. The market dynamics (namely funding rate curve) should incentivize more makers to come into the market in this case, although it is possible for a malicious maker to temporarily increase funding rates in situations where there is not other LPs (or vaults).

mstpr

Escalate for 10 USDC

I think this is an invalid issue.

First of all, it's a great example. However, I would consider this as a great trader than an attacker. There are many assumptions and too many risk factors to pull of such thing and it is definitely a very custom scenario.

Whale deposits tons of maker position and waits for utilization to come to 80%. Since this is very unlikely to happen in a block it will happen over time and meanwhile whale is subject to risk for the entire time since makers are taking the opposite direction of the trade. If everything goes well to plan then whale can pull off this action which again in order to do that whale needs to liquidate itself which is another risk. I think there are too much custom assumptions that are most likely never happens in real world scenarios.

sherlock-admin

> Escalate for 10 USDC

> I think this is an invalid issue.
>
> First of all, it's a great example. However, I would consider this as a great trader than an attacker. There are many assumptions and too many risk factors to pull of such thing and it is definitely a very custom scenario.
>
> Whale deposits tons of maker position and waits for utilization to come to 80%. Since this is very unlikely to happen in a block it will happen over time and meanwhile whale is subject to risk for the entire time since makers are taking the opposite direction of the trade. If everything goes well to plan then whale can pull off this action which again in order to do that whale needs to liquidate itself which is another risk. I think there are too much custom assumptions that are most likely never happens in real world scenarios.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

flowercrimson

I think the issue is valid. It points to opportunities to exploit during liquidation because market liquidity protection is completely bypassed.

The escalation is incorrect in saying that whales are discouraged/unlikely to perform such attacks because of the risks of losing to traders. All makers and traders have risks losing to each other, just because there is inherent risk in trading, doesn't prevent one from becoming a maker. In addition, makers will also be further compensated with position fees.

To me, this issue provides a possible scenario where makers can be malicious and there could be more than one malicious makers.

Funding rates might not be sufficient when there are no other LPs, and also because it is designed under the assumption that `takerOI>makerOI` are edge cases and 'very rare situations' according to docs. When whale makers can be liquidated in a malicious manner, `takerOI>makerOI` is no longer edge case, it would become a regular occurrence for malicious makers to profit, creating cycles of penalization on takers.

KenzoAgada

I think the submission is valid and I agree with flowercrimson's points above.

Additionally, the original escalation said:

> I would consider this as a great trader than an attacker

This reminds me of the Mango Markets exploit, where the exploiter called it *"a very profitable trading strategy"*... Yes, there was no smart contract hack or bug, but the economics allowed manipulation. The situation is not the same here, but there is similarity. The issue throws light on a possible strategy that subverts protocol expectations for fair functioning. I think that the scenario that Emmanuel described is not unlikely enough to be totally dismissed. This will cause unsuspecting users to lose funds (pay fees) in a way the protocol didn't intend to (which is why there is `takerInvariant`). Even if the protocol "accepts the risk", I think this is a good finding that deserves to be pointed out. So I think the first escalation is invalid and a medium severity is appropriate.

jacksanford1

@mstpr Anything else to add after seeing @flowercrimson and @KenzoAgada's responses?

mstpr

@jacksanford1 The reason why I said great trader than an attacker is because this scenario is not a planned attack scenario. This is a very custom scenario overall.

I think the main problem described here is that, in some cases where it is not possible to exit the market users may put theirselves to liquidation and hope to liquidate themselves to exit the market. Which is correct. However, can you block this from happening?

User can't decrease their collateral to a level where they are liquidatable. They can decrease the collateral up to liquidation threshold + 1 at minimum. Then, since the market is reporting in settlements, user needs and hopes that the price declines tiny bit more such that the next settlement user is liquidatable. Meanwhile, user needs to make sure that he will be able to liquidate hisself. If someone else liquidates the position, there is no profit or benefit of doing this action. If there is a robust liquidator community then user takes a great risk.

I would say this is valid if only user could have done this actions in 1 tx. User notices he can't exit the market. Reduce some collateral and liquidates hisself in 1 tx.

I think everything works as intended. User doesn't close the position here, user decrease its collateral and hopes to liquidate itself before others.

Another take to the issue: Whale above scenario might say "I don't care about who liquidates me and takes the liquidation fee I need to exit this position because being a maker in current market will be worse than me to just give up the liquidation fee right now" . Well, obviously this is a trading choice of the whale then. Pay the liquidation fee and exit your position our wait for the utilization to go down and exit your position.

jacksanford1

@flowercrimson @KenzoAgada Any response to @mstpr's latest comment?

Emedudu

The issue here is that `takerInvariant` modifier will not serve its intended purpose.

`takerInvariant` modifier was put in place to ensure that open maker positions does not go lower than open taker positions.

Of course, it is the right of users to close their open positions, but according to Perennial's rules(`takerInvariant` modifier), they shouldn't be able to close their positions such that the global open positions will be less than global taker positions.

I don't think protocol would have added `takerInvariant` to `closeMakeFor` function if they are comfortable with global maker positions dropping below taker positions.

> The reason why I said great trader than an attacker is because this scenario is not a planned attack scenario. This is a very custom scenario overall.

The protocol put `takerInvariant` in place to prevent this scenario from happening, whether it was a planned attack scenario or not.

But now, instead of using the normal door way, which is `closeMakeFor` function (which would enforce the rules and restrict a user from closing his position), User goes through "liquidation" door way(which would have about same effect as `closeMakeFor` and bypass the rules).

This would also lead to HIGH funding fees which would be unfair to all open taker positions.

KenzoAgada

@mstpr in your reply you kind of reframed the discussion to about "user may want to liquidate themselves to exit the market". But what you haven't touched upon in your explanation is the problem of the forced high funding fee. Looking at the original issue's example, the whale can "force" takers to pay the maximum high maxRate. He can then take another maker position himself and make himself the beneficiary of the payment. Isn't this valid as a possible attack path (/a very profitable trading strategy)?

I understand there are some requirements, such as no other large maker positions and the attacker/trader liquidating himself (though depending on params that might be less necessary). So it's not a high severity issue. And I understand that the sponsor says this is working as intended and the market *should* take care of it. But *should* does not guarantee it... It seems to me that the issue describes a valid possible oversight/exploit of the mechanism.

jacksanford1

@arjun-io Do you still feel the same way about this issue?

arjun-io

Our view is this is working as intended as per my initial comment (reposted below). Part of a healthy market is numerous LPs (directly or indirectly through the vault).

> This is working as intended. The market dynamics (namely funding rate curve) should incentivize more makers to come into the market in this case, although it is possible for a malicious maker to temporarily increase funding rates in situations where there is not other LPs (or vaults).

One note: The protocol+market can receive a portion of the close fees which would disincentivize this behavior (similar to the liquidation fee share as described in the recommendation). Capping LPs as a % of the maker limit doesn't really achieve anything, as it creates a bad UX for good actors and bad actors can simply split positions across multiple accounts

jacksanford1

This is a really tricky issue. I think the Mango Markets example actually doesn't apply here, and if this were an issue similar to Mango Markets then it would be Low. The Mango Markets exploit was poorly chosen collateral (MNGO token) and/or a poorly chosen LTV for that collateral. Those are not smart contract issues, and the Mango Markets protocol actually functioned exactly as it was intended to function from a smart contract perspective.

But coming back to the original Impact of this issue:

> User will close his maker position when he shouldn't be allowed to, and it would cause open taker positions to be greatly impacted. And those who are not actively monitoring their open taker positions will suffer loss due to high funding fees.

If it's true that the user shouldn't be allowed to close his position (i.e. not an intended functionality of the smart contracts) AND if it's true that it could cause potentially large losses for other users, then it should be a Medium.

Unless there is further discussion around either of those aspects (intended functionality or loss potential to users) then this should be a Medium imo.

jacksanford1

Result: Medium Has duplicates See previous message for reasoning.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- mstpr: rejected

Issue M-11: Leveraged trader with small collateral can create a riskless position until settlement

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/104

Found by

cergyk

Summary

A highly leveraged trader (around max leverage) with small collateral can autoliquidate position to create a riskless position

Vulnerability Detail

As we can see in closeTake function: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L246-L256

- maintenanceInvariant is not enforced, meaning that extraction of `takerFee` can make the position go unhealthy

And in `liquidate`: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L128-L129

- minimum liquidation incentive equals `liquidationFee*minCollateral`. Which is constant and equal to 0.5 DSU on arbitrum products such as: https://arbiscan.io/address/0x5E660B7B8357059241EAEc143e1e68A5A108D035

We can deduce that a highly leveraged trader, with very small collateral (0.5 DSU) can create a riskless position lasting until next settlement by auto liquidating.

Example: Maintenance factor is 2% Liquidation fee is 5% Taker fee is 0.1% Funding fee is 0%

Alice has a 50x leverage `taker` position opened as authorized per maintenance invariant:

Alice collateral: 0.5 DSU Alice taker position: notional of 25 DSU

To create the riskless position Alice does in the same tx:

- closes the taker position (and by paying the `takerFee` makes her position instantly liquidatable)
- liquidates her position withdrawing the whole 0.5 DSU collateral

For the rest of the oracle version Alice has a 25 DSU position with 0 collateral. Which means that in the end of the period:

- if positive Pnl: Alice gets it added to her collateral, and can withdraw some gains
- if negative Pnl: It gets induced upon the protocol as shortfall: https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/types/OptimisticLedger.sol#L62-L73

If Alice has multiple such positions on multiple accounts, considerable amounts can be stolen this way from the protocol.

Impact

Alice obtains a riskless position, inducing potential shortfall on the protocol

Code Snippet

Tool used

Manual Review

Recommendation

Do not add positive Pnl to the collateral of an account if it does not have any collateral (to enforce the invariant: A position can not be without risk)

Discussion

SergeKireev

Escalate for 10 USDC

It seems this issue has been deemed non-valid because of the minCollateral requirement.

However in current setup on arbitrum (funding fees == 0), it is actually easy to create two opposite positions (one long and one short) with exactly `MIN_COLLATERAL`, such that the sum of the values stays constant. One of these is very likely to end up close to 10% of MIN_COLLATERAL (since when one position increases collateral, the other decreases collateral).

When that happens, the attacker can use the technique described in this report to create a riskless position until next settlement and effectively steal from other users of the market

sherlock-admin

> Escalate for 10 USDC
>
> It seems this issue has been deemed non-valid because of the minCollateral requirement.

> However in current setup on arbitrum (funding fees == 0), it is actually easy to create two opposite positions (one long and one short) with exactly `MIN_COLLATERAL`, such that the sum of the values stays constant. One of these is very likely to end up close to 10% of MIN_COLLATERAL (since when one position increases collateral, the other decreases collateral).
>
> When that happens, the attacker can use the technique described in this report to create a riskless position until next settlement and effectively steal from other users of the market

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

@arjun-io can you please read and share your perspective?

arjun-io

This does appear to be possible, although the work required here to get the position at minCollateral is quite a bit. The attacker will be paying 4x open/close fees in order to get their position into this state - and the most efficient way to get to this sub-minCollateral position is by opening opening highly levered positions which will increase the fee

I would consider this a valid medium

jacksanford1

Since protocol team validates it as possible, seems like this should be a valid Medium. Funds are risk seems potentially too small, but this type of attack could cause a chain reaction once everyone else realizes they are losing small amounts of funds quickly.

jacksanford1

Result: Medium Unique Valid Medium based on Arjun's last comment. This issue is very similar to #103 but I could not make quite enough of a case to duplicate them.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- SergeKireev: accepted

Issue M-12: Accounts will not be liquidated when they are meant to.

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/132

Found by

0xGoodess, Emmanuel, mstpr-brainbot, simon135

Summary

In the case that the totalMaintenance*liquidationFee is higher than the account's totalCollateral, liquidators are paid the totalCollateral. I think one of the reasons for this is to avoid the case where liquidating an account would attempt to debit fees that is greater than the collateral balance The problem is that, the value of totalCollateral used as fee is slightly higher value than the current collateral balance, which means that in such cases, attempts to liquidate the account would revert due to underflow errors.

Vulnerability Detail

Here is the `liquidate` function:

```
function liquidate(
        address account,
        IProduct product
    ) external nonReentrant notPaused isProduct(product)
↪   settleForAccount(account, product) {
        if (product.isLiquidating(account)) revert
↪   CollateralAccountLiquidatingError(account);

        UFixed18 totalMaintenance = product.maintenance(account); maintenance?
        UFixed18 totalCollateral = collateral(account, product);

        if (!totalMaintenance.gt(totalCollateral))
            revert CollateralCantLiquidate(totalMaintenance, totalCollateral);

        product.closeAll(account);

        // claim fee
        UFixed18 liquidationFee = controller().liquidationFee();

        UFixed18 collateralForFee = UFixed18Lib.max(totalMaintenance,
↪   controller().minCollateral());
        UFixed18 fee = UFixed18Lib.min(totalCollateral,
↪   collateralForFee.mul(liquidationFee));
```

```
        _products[product].debitAccount(account, fee);
        token.push(msg.sender, fee);

        emit Liquidation(account, product, msg.sender, fee);
    }
```

`fee=min(totalCollateral,collateralForFee*liquidationFee)` But the PROBLEM is, the value of totalCollateral is fetched before calling `product.closeAll`, and `product.closeAll` debits the closePosition fee from the collateral balance. So there is an attempt to debit `totalCollateral`, when the current collateral balance of the account is `totalCollateral`-closePositionFees This allows the following:

- There is an ETH-long market with following configs:
  - maintenance=5%
  - minCollateral=100USDC
  - liquidationFee=20%
  - ETH price=$1000
- User uses 500USDC to open $10000(10ETH) position
- Price of ETH spikes up to $6000
- Required maintenance= 60000*5%=$3000 which is higher than account's collateral balance(500USDC), therefore account should be liquidated
- A watcher attempts to liquidate the account which does the following:
  - totalCollateral=500USDC
  - `product.closeAll` closes the position and debits a makerFee of 10USDC
  - current collateral balance=490USDC
  - collateralForFee=totalMaintenance=$3000
  - fee=min(500,3000*20%)=500
  - `_products[product].debitAccount(account,fee)` attempts to subtract 500 from 490 which would revert due to underflow
  - account does not get liquidated
- Now, User is not liquidated even when he is using 500USD to control a $60000 position at 120x leverage(whereas, maxLeverage=20x)

NOTE: This would happen when the market token's price increases by (1/liquidationFee)x. In the above example, price of ETH increased by 6x (from 1000USD to 6000USD) which is greater than 5(1/20%)

Impact

A User's position will not be liquidated even when his collateral balance falls WELL below the required maintenance. I believe this is of HIGH impact because this scenario is very likely to happen, and when it does, the protocol will be greatly affected because a lot of users will be trading abnormally high leveraged positions without getting liquidated.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L118
https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L123
https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L129-L132

Tool used

Manual Review

Recommendation

`totalCollateral` that would be paid to liquidator should be refetched after `product.closeAll` is called to get the current collateral balance after closePositionFees have been debited.

Discussion

KenzoAgada

See the `Recommendation` section above for the root of the issue.

arjun-io

Fixed: https://github.com/equilibria-xyz/perennial-mono/pull/204 - this updates `totalCollateral` to reflect the amount after fees have been paid

Issue M-13: Unintended Vault Operation Due to Product Settling and Oracle Version Skips

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/152

Found by

mstpr-brainbot

Summary

Vault operation can behave unexpectedly when products settle and skip versions in their oracles. This problem arises when the vault assumes a non-existent version of a product, leading to miscalculations in the _assetsAtEpoch() function. This inaccuracy can affect the distribution of shares among depositors, redeemers, and claimers, disrupting the balance of the market.

Vulnerability Detail

When a product progresses to a new version of its oracle, it first settles at the current version plus one, then jumps to the latest version. For instance, if a product's latest version is 20 and the current version is 23, the product will first settle at version 21 and then jump from 21 to 23, bypassing version 22. This process can lead to unintended behavior in the vault, particularly if the vault's deposited epoch points to the bypassed version, which can result in a potential underflow and subsequent rounding to zero.

Consider this scenario: A vault is operating with two markets, each having two products. Let's assume the vault has equal positions and collateral in both markets, with all assets priced at $1 for simplicity. Also, assume that Market0 contains product A and product B, while Market1 contains product C and product D.

The latest version of product A is 20, while the vault's latest epoch is 2, which corresponds to version 20 for product A. Similarly, the latest version of product C is 10, with the vault's latest epoch at 2, corresponding to version 10 for product A.

Assume there are no positions created in product C, D, and also no deposits made to the vault, meaning the vault has not opened any positions on these products either. The oracle version for product C, D gets updated twice consecutively. Since there are no open positions in product C, D, their products will progress to version 12 directly, bypassing version 11.

This circumstance renders the epoch stale, which will lead to miscalculations in deposits, claims, and redemptions. During the _assetsAtEpoch() function, the vault incorrectly assumes that there is a version 11 of product C, D. However, these products have skipped version 11 and advanced to 12. Since the latest version of product C, D is greater than the version the vault holds, the vault presumes that version 11 exists. However, as version 11 doesn't exist, the _accumulatedAtEpoch will be the negative of whatever was deposited. This leads the vault to incorrectly assume a large loss in that market. Consequently, depositors can gain more shares, while redeemers and claimers may receive fewer shares.

Impact

If the above scenario happens, vault users can incur extreme losses on claim and redeem operations and depositors can earn uneven shares.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/product/Product.sol#L84-L130 settle

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/interfaces/types/PrePosition.sol#L133-L136 if no open positions in preposition settle version is current version

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L376

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L823-L843 checks for +1 version, if latestVersion > currentVersion, it assumes vault lost all the funds in that market.

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L797-L805

Tool used

Manual Review

Recommendation

Don't check +1 version, check the current version recorded for that epoch in vault and the latest version of the product, cover all the versions

Discussion

kbrizzle

It is guaranteed that all applicable versions have a corresponding checkpoint in the underlying markets at version `n` because product.settle() is called on every market (here from here) when an epoch snapshot takes place.

Further, when there is a deposit or withdrawal `_updateMakerPosition` will create a new position in the underlying product here, which will create a checkpoint for `n + 1`.

KenzoAgada

Suspected as much but wanted to make sure... Closing as invalid.

KenzoAgada

After internal discussion: the issue is valid, but the watson didn't fully identify the exact complex edge case where it will materialize. Therefore downgrading to medium severity.

mstpr

Escalate for 10 USDC

I am not sure what's the edge case, it is straightforward.

A vault is synced with its underlying product as long as it is interacting with its products. A product can advance more oracle versions while vault still outdates with another one if there isn't any interactions been made in due time. At the end of the _settle function the correct and the most updated oracle version is synced. However, just in the settlement process vault only checks the latest synced version of its and the one after that (which there might not be a +1 version due to skipping).

If a product has in version 12 and the previous version of it was 10 (means that product skipped version 11 and went directly to version 12) and the vaults latest synced version is 10 (no interactions made with vault so vault still thinks the latest version of the product is 10), the next time someone calls deposit/redeem/claim or in general anything that would trigger the settlement process, the accumulated assets at that epoch will be calculated as follows:

products latest version of the underlying product is 10. So _accumulatedAtEpoch will calculate the balance respect to oracle version 10 and 11. However, there is no version 11, and default it will be 0 and since the latest version of the product (12) > 10 (latest synced oracle version of vault) the accounting will be mess. Since there is no oracle version 11, the accumulated balance will be calculated as the negative of version 10. If the epoch is stale, there can be deposits/redeems/claims in the latestEpoch and when the sync() happens the accounting will go off.

Also, this is directly related with user funds and huge losses can be reported, so I think this is a valid high.

sherlock-admin

> Escalate for 10 USDC
>
> I am not sure what's the edge case, it is straightforward.

> A vault is synced with its underlying product as long as it is interacting with its products. A product can advance more oracle versions while vault still outdates with another one if there isn't any interactions been made in due time. At the end of the _settle function the correct and the most updated oracle version is synced. However, just in the settlement process vault only checks the latest synced version of its and the one after that (which there might not be a +1 version due to skipping).
>
> If a product has in version 12 and the previous version of it was 10 (means that product skipped version 11 and went directly to version 12) and the vaults latest synced version is 10 (no interactions made with vault so vault still thinks the latest version of the product is 10), the next time someone calls deposit/redeem/claim or in general anything that would trigger the settlement process, the accumulated assets at that epoch will be calculated as follows:
>
> products latest version of the underlying product is 10. So _accumulatedAtEpoch will calculate the balance respect to oracle version 10 and 11. However, there is no version 11, and default it will be 0 and since the latest version of the product (12) > 10 (latest synced oracle version of vault) the accounting will be mess. Since there is no oracle version 11, the accumulated balance will be calculated as the negative of version 10. If the epoch is stale, there can be deposits/redeems/claims in the latestEpoch and when the sync() happens the accounting will go off.
>
> Also, this is directly related with user funds and huge losses can be reported, so I think this is a valid high.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

securitygrid

Comment from a waston:

```
/**
     * @notice Rebalances the collateral and position of the vault without a
     ↪  deposit or withdraw
     * @dev Should be called by a keeper when a new epoch is available, and
     ↪  there are pending deposits / redemptions
     */
    function sync() external {
        syncAccount(address(0));
    }
```

`sync()` Should be called by a keeper when a new epoch is available. If the keeper is running normally, there will be no such assumptions in the report: `if a product's latest version is 20 and the current version is 23`. Because sync() will call product._settle internally. so the above scenario will not happen.

mstpr

> ```
> /**
>      * @notice Rebalances the collateral and position of the vault without
>       ↪  a deposit or withdraw
>      * @dev Should be called by a keeper when a new epoch is available, and
>       ↪  there are pending deposits / redemptions
>      */
>     function sync() external {
>         syncAccount(address(0));
>     }
> ```
>
> sync() Should be called by a keeper when a new epoch is available. so the above scenario will not happen.

Product advances an oracle version and vault syncs.

When a product advances to version 12 from 10 by skipping version 11 (which is possible if there are no open positions in version 10)

and if the epoch is stale there can be deposits made by users.

Now, even if keeper would call the sync() after 1 second the above scenario is consistent.

KenzoAgada

I don't have a lot to add on this escalation. On one hand, it seems the watson identified a valid critical issue. On the other hand, if I understand the sponsor correctly, they said that report was not very useful or detailed nor enough to really understand and fix the problem. Up to Sherlock to decide severity.

jacksanford1

@securitygrid Are you satisfied with @mstpr's last response?

I don't believe the likelihood and severity potential for this issue combine to warrant a High.

However it seems to be a legitimate issue (or very close to describing one) and so it seems like a Medium.

securitygrid

If the keeper is working, why is there a delay of 2 versions?

jacksanford1

I guess @mstpr can answer that best

mstpr

I don't think you get the issue here @securitygrid. As stated above, vaults and products separate things. Product advance a version first then vault syncs.

Vault only advances a version if the underlying products advances to the next version. It is completely not about keepers. If marketA (2 products, long and short) and marketB (2 products, long and short) are in oracle version 3 and vault is in epoch 2, when marketA advances to version 3 but marketB is still in version 2, the vault is still in epoch2, vault can't advance to epoch3 before marketB is advanced to version 3 aswell.

jacksanford1

@securitygrid Any thoughts on mstpr's latest comment?

jacksanford1

Result: Medium Unique

> On one hand, it seems the watson identified a valid critical issue. On the other hand, if I understand the sponsor correctly, they said that report was not very useful or detailed nor enough to really understand and fix the problem.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- mstpr: rejected

Issue M-14: Users can be forced to claim assets at bad rate in some cases

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/174

Found by

mstpr-brainbot, rvierdiiev, tvdung94

Summary

In balanced vault contract, anyone can call claim assets for other accounts; malicious users could abuse this function to force other users to receive less assets than they expect.

Vulnerability Detail

When claiming asset, pro rate, in which users will receive less assets than expect, will be applied when total collateral is less than total unclaimed amount. So after redeeming shares and converting it to assets, users might not want to claim assets right away in this scenario, for they will receive less token amount. However, other users can force them to claim via claim() function because there is no restrict on this function to claim for other accounts.

Impact

Users will be forced to receive less assets in some scenarios.

Code Snippet

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L211-L228

Tool used

Manual Review

Recommendation

Only account owner (msg.sender == account) can claim asset for their account.

Discussion

KenzoAgada

While the general claim pro-rata mechanism has already been reported in the Veriside audit (issue 002) and acknowledged by the protocol, this submission correctly adds that a user can pro-rata claim *other user's* assets, even if they preferred to avoid realizing losses and stay in the vault until collateral recovers. This will make users lose funds, and increase the malicious user's own amount of rewards, for when the collateral recovers.

arjun-io

This is a great find. We will think about whether or not we want to fix this as the unpermissioned claim is currently used by the protocol for better UX with the MultiInvoker

arjun-io

Fixed: https://github.com/equilibria-xyz/perennial-mono/pull/206 - this change only allows "self" claims if the vault is pro-rated

Issue M-15: Liquidators can prevent users from making their positions healthy during an unpause

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/190

Found by

AkshaySrivastav, SolidityATL, rvierdiiev

Summary

The Perennial protocol has a paused state in which all operations are paused. The protocol can be unpaused by privileged accounts. But when this unpause happens the liquidators can frontrun and liquidate user positions before those users get a chance to make their positions healthy.

https://github.com/sherlock-audit/2023-05-perennial/blob/main/perennial-mono/packages/perennial/contracts/collateral/Collateral.sol#L108-L135

Vulnerability Detail

The `pauser` address can pause the perennial protocol using the `Controller.updatePaused` function. Once the protocol is paused all these operations cannot be done by users:

- Open/close/modify current make or take positions on a product.
- Deposit or withdraw collateral.
- Liquidate under-collateralized positions.

Though, real prices from oracles will surely move up or down during this paused period. If the oracle prices go down, the users won't be allowed to add more collateral to their positions or close their positions. Hence their positions will get under-collateralized (based upon real prices).

Once the protocol is unpaused the liquidators can front-run most users and liquidate their positions. Most users will not get a chance to make their position healthy.

This results in loss of funds for the users.

Perennial also has the concept of settlement delay. Any position opened/closed at oracle version `n` is settled at oracle version `n + 1`. This also alleviates the frontrunning liquidation issue. While validating an account's health before liquidation, the protocol only considers the current maintenance requirement for the account (not the next). This makes users more prone to getting front-runned and liquidated.

Ref: audit finding

Impact

By front-running any collateral deposit or position closure of a legitimate user which became under-collateralized during the paused state, the liquidator can unfairly liquidate user positions and collect liquidation profit as soon as the protocol is unpaused. This causes loss of funds to the user.

Code Snippet

Collateral

```
function depositTo(address account, IProduct product, UFixed18 amount)
external
nonReentrant
notPaused
notZeroAddress(account)
isProduct(product)
collateralInvariant(account, product)
{
    _products[product].creditAccount(account, amount);
    token.pull(msg.sender, amount);

    emit Deposit(account, product, amount);
}
```

Product

```
function closeTakeFor(address account, UFixed18 amount)
    public
    nonReentrant
    notPaused
    onlyAccountOrMultiInvoker(account)
    settleForAccount(account)
    closeInvariant(account)
    liquidationInvariant(account)
{
    _closeTake(account, amount);
}

function closeMakeFor(address account, UFixed18 amount)
    public
    nonReentrant
    notPaused
    onlyAccountOrMultiInvoker(account)
    settleForAccount(account)
    takerInvariant
    closeInvariant(account)
    liquidationInvariant(account)
{
    _closeMake(account, amount);
}
```

Tool used

Manual Review

Recommendation

Consider adding a grace period after unpausing during which liquidation remains blocked to allow users to avoid unfair liquidation by closing their positions or supplying additional collateral.

Discussion

akshaysrivastav

Escalate for 10 USDC

I think this issue should be considered as valid. As per the comment here, the protocol team can disagree with the mitigation suggested in #168 but the validity of issue should be accepted. The report clearly shows how a protocol owner actions (pause) will result in unfair liquidations causing loss of funds to users.

Also it is unlikely that on unpause human users will be able to secure their positions before MEV/liquidation bots capture the available profit. Hence the loss is certain.

For reference, a similar issue was consider valid in the recent Notional V3 contest. Maintaining a consistent valid/invalid classification standard will be ideal here.

sherlock-admin

> Escalate for 10 USDC
>
> I think this issue should be considered as valid. As per the comment here, the protocol team can disagree with the mitigation suggested in #168 but the validity of issue should be accepted. The report clearly shows how a protocol owner actions (pause) will result in unfair liquidations causing loss of funds to users.
>
> Also it is unlikely that on unpause human users will be able to secure their positions before MEV/liquidation bots capture the available profit. Hence the loss is certain.

> For reference, a similar issue was consider valid in the recent Notional V3 contest. Maintaining a consistent valid/invalid classification standard will be ideal here.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

Sherlock:

- At the main issue, #168, I wrote that "This can be considered as an un-mandatory "idea for improvement", but I think that the risk is there so this is a reasonable submission, and if I recall correctly, was accepted by Sherlock judges in the past."
- The sponsor disputed the issue and replied that "By introducing a time delay to allow traders to optionally increase their collateral to meet maintenance requirements the Product runs the risk of going into further shortfall if the traders opt to not increase their margin. The fairest option for both liquidators and other traders in the protocol is to require that traders are paying attention and self-liquidate (to retain the fee) if they so chose.".
- I accepted the sponsor's POV and closed the issue as a design choice.
- However here the escalator says that even if the sponsor would not like to implement the mitigation, the *validity* of the issue should be accepted.
- This issue has been accepted recently in Notional and Blueberry. But unlike here, there the sponsor confirmed the issues.

I think the watson raises a valid point. Even if the sponsor chose not to allow this as a design choice, the issue itself is there, was accepted by Sherlock in the past, and therefore should probably be accepted here as well. I think restoring it to medium is fair.

jacksanford1

Seems like a valid issue if true. The recommended fix's acceptance/rejection is not relevant to the issue itself.

This is a temporary freezing (pause) induced by the protocol team, but it can cause the protocol to enter a state where on unpause many users will lose their funds to MEV (due to being liquidatable) before the user can salvage the position potentially.

Borderline Low issue but we can make it a Medium since it can result in a significant loss of funds for many users at the same time.

jacksanford1

Result: Medium Has Duplicates Reasoning can be found in the previous message.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- akshaysrivastav: accepted

Issue M-16: `BalancedVault` doesn't consider potential break in one of the markets

Source: https://github.com/sherlock-audit/2023-05-perennial-judging/issues/232

Found by

BLACK-PANDA-REACH, Bauchibred, mstpr-brainbot

Summary

In case of critical failure of any of the underlying markets, making it permanently impossible to close position and withdraw collateral all funds deposited to balanced Vault will be lost, including funds deposited to other markets.

Vulnerability Detail

As Markets and Vaults on Perennial are intented to be created in a permissionless manner and integrate with external price feeds, it cannot be ruled out that any Market will enter a state of catastrophic failure at a point in the future (i.e. oracle used stops functioning and Market admin keys are compromised, so it cannot be changed), resulting in permanent inability to process closing positions and withdrawing collateral.

`BalancedVault` does not consider this case, exposing all funds deposited to a multi-market Vault to an increased risk, as it is not implementing a possibility for users to withdraw deposited funds through a partial emergency withdrawal from other markets, even at a price of losing the claim to locked funds in case it becomes available in the future. This risk is not mentioned in the documentation.

Proof of Concept

Consider a Vault with 2 markets: ETH/USD and ARB/USD.

1. Alice deposits to Vault, her funds are split between 2 markets
2. ARB/USD market undergoes a fatal failure resulting in `maxAmount` returned from `_maxRedeemAtEpoch` to be 0
3. Alice cannot start withdrawal process as this line in `redeem` reverts:

```
if (shares.gt(_maxRedeemAtEpoch(context, accountContext, account))) revert
↪  BalancedVaultRedemptionLimitExceeded();
```

Impact

Users funds are exposed to increased risk compared to depositing to each market individually and in case of failure of any of the markets all funds are lost. User has no possibility to consciously cut losses and withdraw funds from Markets other than the failed one.

Code Snippet

`_maxRedeemAtEpoch` https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L649-L677

check in `redeem` https://github.com/sherlock-audit/2023-05-perennial/blob/0f73469508a4cd3d90b382eac2112f012a5a9852/perennial-mono/packages/perennial-vaults/contracts/balanced/BalancedVault.sol#L188

Tool used

Manual Review

Recommendation

Implement a partial/emergency withdrawal or acknowledge the risk clearly in Vault's documentation.

Discussion

KenzoAgada

Please note that the duplicate issues referenced above mention paused markets and oracle fails as additional scenarios where markets can break and therefore break the vault. Issue #20 contains recommendations on how to handle stuck epochs and oracles.

securitygrid

Escalate for 10 USDC This is not valid M. The protocol is designed that way. Any protocol that requires external oracle price feeds will experience oracle misbehavior. This is inevitable. But this is only temporary and occasional. This is acceptable.

sherlock-admin

> Escalate for 10 USDC This is not valid M. The protocol is designed that way. Any protocol that requires external oracle price feeds will experience oracle misbehavior. This is inevitable. But this is only temporary and occasional. This is acceptable.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

KenzoAgada

- Note that I grouped together with this issue all submissions which speak about increased danger of vaults due to breaking markets - either by oracle failures or paused products.
- The issues say that a vault with `n` markets is exposed to `2n` possible problem in products.
- If even *only one* of the products is paused, or experiences oracle issues, the vault would jam.
- There are no contingency measures in such a scenario.
- I agree with the escalation that "the protocol is designed that way". But I'm not sure that users are aware that eg. a pause in one of the products would render the whole vault inoperable.
- Therefore I think it is debatable, and understand if the sponsor won't fix, but I would maintain medium severity due to the increased risk and lack of contingency measures.

edit: additionally, a watson mentioned in the chat that in the contest readme there is the following section:

> **Q: In case of external protocol integrations, are the risks of external contracts pausing or executing an emergency withdrawal acceptable? If not, Watsons will submit issues related to these situations that can harm your protocol's functionality.** A: We want to be aware of issues that might arise from Chainlink or DSU integrations

I think that this can be legitimately said to be such an issue, and AFAIK it was not mentioned specifically anywhere that a vault would break if one of `2n` products breaks.

jacksanford1

@securitygrid I think what the submitter is saying is that a market could break for an unknown reason (could be oracle-related or not) and if the vault deposits into 10 markets and 1 market breaks in a irrecoverable way, then the funds in the other 9 markets cannot be retrieved by the depositor.

If that's the case, I'd argue this is a Medium vulnerability.

jacksanford1

Result: Medium Has duplicates Keeping as Medium due to reasoning in previous comment.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- securitygrid: rejected