SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Y2K
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: March 13 - March 27, 2023
Prepared on: May 22, 2023

Introduction

Y2K is a crypto-native take on structured products on-chain. The protocol creates liquid markets for hedging, leveraging, speculating and trading.

Scope

Repository: Y2K-Finance/Earthquake

Branch: earthquake-v2-sherlock-audit

Commit: 736b2e1e51bef6daa6a5ecd1decb7d156316d795

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
14	5

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

ast3ros
bin2chen
roguereddwarf
berndartmueller
iglyx
nobody2018
evan
kenzo
cccz
VAD37
Ruhum
Dug

0x52
HonorLt
TrungOre
hickuphh3
immeas
ltyu
Respx
libratus
p0wd3r
minhtrng
warRoom
jprod15
ShadowForce
spyrosonic10
holyhansss
mstpr-brainbot
yixxas
toshii
twicek
0xRobocop
Inspex
Ace-30
Ch_301
sinarette
ElKu
carrot
ck
volodya
csanuragjain
0xmuxyz
joestakey
Bauer
charlesjhongc
0Kage
0xnirlin
J4de
KingNFT
ni8mare
datapunk
b4by_y0d4
Emmanuel
AlexCzm
BPZ
bulej93
climber2002
0xvj
Delvir0
Saeedalipoor01988
ABA
lemonmon
kaysoft
martin
peanuts
zeroknots
shaka
auditor0517
ne0n
pfapostol
jasonxiale
0xMojito
Junnon
0xPkhatri
Aymen0909

Issue H-1: Funds can be stolen because of incorrect update to `ownerToRollOverQueueIndex` for existing rollovers

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/2

Found by

0Kage, 0x52, 0xMojito, 0xPkhatri, 0xRobocop, 0xnirlin, AlexCzm, Aymen0909, Bauer, Ch_301, Dug, ElKu, Emmanuel, HonorLt, Junnon, Respx, TrungOre, VAD37, ast3ros, auditor0517, berndartmueller, bin2chen, cccz, charlesjhongc, ck, climber2002, csanuragjain, datapunk, evan, hickuphh3, holyhansss, iglyx, immeas, jasonxiale, joestakey, kenzo, libratus, ltyu, minhtrng, mstpr-brainbot, ne0n, pfapostol, rogue reddwarf, shaka, sinarette, spyrosonic10, toshii, twicek, volodya, warRoom, yixxas, zeroknots

Summary

In the case where the owner has an existing rollover, the `ownerToRollOverQueueIndex` incorrectly updates to the last queue index. This causes the `notRollingOver` check to be performed on the incorrect `_id`, which then allows the depositor to withdraw funds that should've been locked.

Vulnerability Detail

In `enlistInRollover()`, if the user has an existing rollover, it overwrites the existing data:

```
if (ownerToRollOverQueueIndex[_receiver] != 0) {
  // if so, update the queue
  uint256 index = getRolloverIndex(_receiver);
  rolloverQueue[index].assets = _assets;
  rolloverQueue[index].epochId = _epochId;
```

However, regardless of whether the user has an existing rollover, the `ownerToRolloverQueueIndex` points to the last item in the queue:

```
ownerToRollOverQueueIndex[_receiver] = rolloverQueue.length;
```

Thus, the `notRollingOver` modifier will check the incorrect item for users with existing rollovers:

```
QueueItem memory item = rolloverQueue[getRolloverIndex(_receiver)];
if (
    item.epochId == _epochId &&
```

```
    (balanceOf(_receiver, _epochId) - item.assets) < _assets
) revert AlreadyRollingOver();
```

allowing the user to withdraw assets that should've been locked.

Impact

Users are able to withdraw assets that should've been locked for rollovers.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L252-L257 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L268 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L755-L760

Tool used

Manual Review

Recommendation

The `ownerToRollOverQueueIndex` should be pointing to the last item in the queue in the `else` case only: when the user does not have an existing rollover queue item.

```
} else {
  // if not, add to queue
  rolloverQueue.push(
      QueueItem({
          assets: _assets,
          receiver: _receiver,
          epochId: _epochId
      })
  );
+ ownerToRollOverQueueIndex[_receiver] = rolloverQueue.length;
}
- ownerToRollOverQueueIndex[_receiver] = rolloverQueue.length;
```

Discussion

3xHarry

good catch

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/128

IAm0x52

Fix looks good. Assigning index has been moved inside else block

Issue H-2: Earlier users in rollover queue can grief later users

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/72

Found by

BPZ, Ch_301, Dug, Emmanuel, J4de, Ruhum, TrungOre, ast3ros, berndartmueller, bin2chen, evan, hickuphh3, immeas, jprod15, kenzo, ltyu, minhtrng, mstpr-brainbot, nobody2018, roguereddwarf, sinarette, spyrosonic10, toshii, twicek

Summary

The current implementation enables users who are earlier in the queue to grief those who are later.

Vulnerability Detail

There is a `rolloverAccounting` mapping that, for every epoch, tracks the current index of the queue for which mints have been processed up to thus far.

When a user delists from the queue, the last user enlisted will replace the delisted user's queue index.

It is thus possible for the queue to be processed up to, or past, the delisted user's queue index, but before the last user has been processed, the processed user delists, thus causing the last user to not have his funds rollover.

POC

1. Alice enlists into the queue (index `1`), then Bob (index `2`)
2. Alice (or a relayer) calls `mintRollovers()` with `_operations = 1`, and Alice has her funds rollover.
3. Alice delists from the rollover.

Bob is then unable to have his funds rollover until the next epoch is created, unless he delists and re-enlists into the queue (defeating the purpose of rollover functionality).

Impact

Whether accidental or deliberate, it is possible for users to not have their funds rollover.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L293-L296

Tool used

Manual Review

Recommendation

Instead of specifying the number of operations to execute, consider having start and end indexes, with a boolean mapping to track if a user's rollover has been processed.

Discussion

3xHarry

keeping track of rollovers with a mapping would increase gas cost substantially, however it would be a better solution than blocking delisting during deposit period

3xHarry

setting assets to 0 instead of removing the QueueItem from the array sounds like a more reasonable approach, given that it's very unlikely for the rollover queue array length to reach the max size. Also, there can be more markets with similar strike prices deployed at any time.

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/127

0xRobocop

Escalate for 10 USDC

This is a valid low issue but not a high or med

This is more of an inconvenience for the user and there is no loss:

"User experience and design improvement issues: Issues that cause minor inconvenience to users where there is no material loss of funds are not considered valid. Funds are temporarily stuck and can be recovered by the administrator or owner. Also, if a submission is a design opinion/suggestion without any clear indications of loss of funds is not a valid issue."

There is also a little guideline to identify highs and meds. Pay attention to "should not be easily replaced without loss of funds" which is not the case in this issue.

sherlock-admin

> Escalate for 10 USDC
>
> This is a valid low issue but not a high or med
>
> This is more of an inconvenience for the user and there is no loss:
>
> "User experience and design improvement issues: Issues that cause minor inconvenience to users where there is no material loss of funds are not considered valid. Funds are temporarily stuck and can be recovered by the administrator or owner. Also, if a submission is a design opinion/suggestion without any clear indications of loss of funds is not a valid issue."
>
> There is also a little guideline to identify highs and meds. Pay attention to "should not be easily replaced without loss of funds" which is not the case in this issue.

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.

dmitriia

Not agree with the escalation, that's core logic flaw with a range of material impacts, definitely high.

hrishibhat

Escalation rejected

Based on the issue and its duplicates and their impacts, considering this issue as a valid high since it breaks the core functionality.

sherlock-admin

> Escalation rejected
>
> Based on the issue and its duplicates and their impacts, considering this issue as a valid high since it breaks the core functionality.

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

IAm0x52

Needs additional changes. Using isEnlistedInRolloverQueue causes duplicate entries that can't be removed

IAm0x52

Fix looks good. isEnlistedInRolloverQueue has been changed making it impossible to have duplicate entries

jacksanford1

Note: 0x52's last message is in reference to this commit:

https://github.com/Y2K-Finance/Earthquake/pull/127/commits/1d1ac0a3411208cc7a3a7d4668ff123bffb2ff21

Issue H-3: `depositFee` can be bypassed via deposit queue

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/75

Found by

0xRobocop, Ace-30, AlexCzm, Ch_301, Dug, ElKu, Inspex, J4de, Respx, Ruhum, ShadowForce, TrungOre, VAD37, ast3ros, bulej93, evan, hickuphh3, iglyx, immeas, kenzo, minhtrng, rogue_reddwarf, toshii, yixxas

Summary

The deposit fee can be circumvented by a queue deposit + `mintDepositInQueue()` call in the same transaction.

Vulnerability Detail

A deposit fee is charged and increases linearly within the deposit window. However, this fee can be avoided if one deposits into the queue instead, then mints his deposit in the queue.

POC

Assume non-zero `depositFee`, valid epoch `_id = 1`. At epoch end, instead of calling `deposit(1, _assets, 0xAlice)`, Alice writes a contract that performs `deposit(0,_assets,0xAlice)` + `mintDepositInQueue(1,1)` to mint her deposit in the same tx (her deposit gets processed first because FILO system) . She pockets the `relayerFee`, essentially paying zero fees instead of incurring the `depositFee`.

Impact

Loss of protocol fee revenue.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L494-L500 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L332-L333 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L354

Tool used

Manual Review

Recommendation

Because of the FILO system, charging the dynamic deposit fee will be unfair to queue deposits as they're reliant on relayers to mint their deposits for them. Consider taking a proportion of the relayer fee.

Discussion

3xHarry

This is a valid issue. We will apply depositFee to all mints (queue and direct). However, given that queue has the potential to affect when users's shares are minted because of FILO, min deposit has to be raised for the queue, to make it substantially harder to DDoS the queue. Minimizing DDoS queue deposits will lead to queue deposits getting the least fees as relayers can mint from the first second the epoch is created.

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/126

3xHarry

@IAm0x52 to elaborate on this issue: relayers are incentivized to mint the depositQueue from the second a new epoch is created to extract the most amount of relayerFees. In fact Y2K will have a build in relayerInfra into the deployment process. The assumption is, that queueDeposit users will pay a minimal Fee. The attack factor of the queue beeing to long leading to prolonged queue deposit executions will be mitigated by adding a significant deposit requirement for queue deposits. These measures will mitigate high deposit Fees for Queue deposits as well as prevent late direct depositors using the queue to evade the depositFee.

jacksanford1

Bringing in this discussion from Discord:

0x52

> As a follow up for PR126. You keep the minRequiredDeposit modifier on enlistInRollover but the way you modified it, it can only apply if epochId == 0 but enlistInRollover doesn't work for epochId == 0 so the modifier is useless on that function. My suggestion would be to either remove it if you no longer need that protection or make a new modifier specifically designed for enlistInRollover

3xHarry

> regarding [issue] 75 / PR 126 fixed in https://github.com/Y2K-Finance/Earthquake/pull/126/commits/9c659161dc952df99201b99d4ea54e9dda642ecb

IAm0x52

Fix looks good. enlistInRollover now applies a minimum deposit requirement

Issue H-4: When rolling over, user will lose his winnings from previous epoch

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/163

Found by

Ace-30, Inspex, TrungOre, VAD37, berndartmueller, bin2chen, carrot, cccz, charlesjhongc, evan, hickuphh3, iglyx, immeas, kenzo, minhtrng, mstpr-brainbot, nobody2018, roguereddwarf, toshii, warRoom

Summary

When `mintRollovers` is called, when the function mints shares for the new epoch for the user, the amount of shares minted will be the same as the original assets he requested to rollover - **not including the amount he won.** After this, **all these asset shares from the previous epoch are burnt.** So the user won't be able to claim his winnings.

Vulnerability Detail

When user requests to enlistInRollover, he supplies the amount of assets to rollover, and this is saved in the queue.

```
rolloverQueue[index].assets = _assets;
```

When `mintRollovers` is called, the function checks if the user won the previous epoch, and proceeds to **burn all the shares** the user requested to roll:

```
if (epochResolved[queue[index].epochId]) {
    uint256 entitledShares = previewWithdraw(
        queue[index].epochId,
        queue[index].assets
    );
    // mint only if user won epoch he is rolling over
    if (entitledShares > queue[index].assets) {
        ...
        // @note we know shares were locked up to this point
        _burn(
            queue[index].receiver,
            queue[index].epochId,
            queue[index].assets
        );
```

Then, and this is the problem, the function **mints to the user his original assets - `assetsToMint` - and not `entitledShares`.**

```
uint256 assetsToMint = queue[index].assets - relayerFee;
_mintShares(queue[index].receiver, _epochId, assetsToMint);
```

So the user has only rolled his original assets, but since all his share of them is burned, he will not be able anymore to claim his winnings from them.

Note that if the user had called `withdraw` instead of rolling over, all his shares would be burned, but he would receive his `entitledShares`, and not just his original assets. We can see in this in `withdraw`. Note that `_assets` is burned (like in minting rollover) but `entitledShares` is sent (unlike minting rollover, which only remints `_assets`.)

```
_burn(_owner, _id, _assets);
_burnEmissions(_owner, _id, _assets);
uint256 entitledShares;
uint256 entitledEmissions = previewEmissionsWithdraw(_id, _assets);
if (epochNull[_id] == false) {
    entitledShares = previewWithdraw(_id, _assets);
} else {
    entitledShares = _assets;
}
if (entitledShares > 0) {
    SemiFungibleVault.asset.safeTransfer(_receiver, entitledShares);
}
if (entitledEmissions > 0) {
    emissionsToken.safeTransfer(_receiver, entitledEmissions);
}
```

Impact

User will lose his rewards when rolling over.

Code Snippet

```
if (epochResolved[queue[index].epochId]) {
    uint256 entitledShares = previewWithdraw(
        queue[index].epochId,
        queue[index].assets
    );
    // mint only if user won epoch he is rolling over
    if (entitledShares > queue[index].assets) {
        ...
        // @note we know shares were locked up to this point
        _burn(
```

```
            queue[index].receiver,
            queue[index].epochId,
            queue[index].assets
        );
```

Tool used

Manual Review

Recommendation

Either remint the user his winnings also, or if you don't want to make him roll over the winnings, change the calculation so he can still withdraw his shares of the winnings.

Discussion

3xHarry

this makes total sense! thx for catching this!

3xHarry

will have to calculate how much his original deposit is worth in entitledShares and rollover the specified amount

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/125

IAm0x52

Needs additional changes. This will revert if diff is too high due to underflow in L412

IAm0x52

Fix looks good. Point of underflow has been removed in a subsequent PR

jacksanford1

Note: Subsequent PR 0x52 is referencing refers to this commit:

https://github.com/Y2K-Finance/Earthquake/pull/125/commits/3732a7075348e87da612166dd060bfd8dd742ecb

Issue H-5: Adversary can break deposit queue and cause loss of funds

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/468

Found by

0x52, 0xRobocop, Bauer, HonorLt, Respx, Ruhum, VAD37, bin2chen, immeas, joestakey, jprod15, libratus, ltyu, mstpr-brainbot, nobody2018, rogueredwarf, warRoom, yixxas

Summary

Vulnerability Detail

Carousel.sol#L531-L538

```
function _mintShares(
    address to,
    uint256 id,
    uint256 amount
) internal {
    _mint(to, id, amount, EMPTY);
    _mintEmissions(to, id, amount);
}
```

When processing deposits for the deposit queue, it _mintShares to the specified receiver which makes a _mint subcall.

ERC1155.sol#L263-L278

```
function _mint(address to, uint256 id, uint256 amount, bytes memory data)
↪   internal virtual {
    require(to != address(0), "ERC1155: mint to the zero address");

    address operator = _msgSender();
    uint256[] memory ids = _asSingletonArray(id);
    uint256[] memory amounts = _asSingletonArray(amount);

    _beforeTokenTransfer(operator, address(0), to, ids, amounts, data);

    _balances[id][to] += amount;
    emit TransferSingle(operator, address(0), to, id, amount);

    _afterTokenTransfer(operator, address(0), to, ids, amounts, data);
```

```
    _doSafeTransferAcceptanceCheck(operator, address(0), to, id, amount, data);
}
```

The base ERC1155 _mint is used which always behaves the same way that ERC721 safeMint does, that is, it always calls _doSafeTrasnferAcceptanceCheck which makes a call to the receiver. A malicious user can make the receiver always revert. This breaks the deposit queue completely. Since deposits can't be canceled this WILL result in loss of funds to all users whose deposits are blocked. To make matters worse it uses first in last out so the attacker can trap all deposits before them

Impact

Users who deposited before the adversary will lose their entire deposit

Code Snippet

Carousel.sol#L310-L355

Tool used

Manual Review

Recommendation

Override _mint to remove the safeMint behavior so that users can't DOS the deposit queue

Discussion

3xHarry

agree with this issue, there is no easy solution to this, as by definition when depositing into queue, the user gives up the atomicity of his intended mint. Looking at Openzeppelins 1155 implementation guide it is recommended to ensure the receiver of the asset is able to call safeTransferFrom. By removing the acceptance check in the `_mint` function, funds could be stuck in a smart contract.

Another alternative would be to do the 1155 acceptance check in the mint function and confiscate the funds if the receiver is not able to hold 1155s. The funds could be retrieved via a manual process from the treasury afterward.

3xHarry

going with Recommendation is prob the easiest way

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/124

IAm0x52

Fix looks good. _mint no longer calls acceptance check so rollover can longer be DOS'd by it

Issue M-1: ControllerPeggedAssetV2: outdated price may be used which can lead to wrong depeg events

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/70

Found by

0xRobocop, 0xnirlin, ABA, Ch_301, Delvir0, Saeedalipoor01988, ShadowForce, TrungOre, ast3ros, bin2chen, carrot, evan, kaysoft, lemonmon, martin, minhtrng, p0wd3r, peanuts, rogueredd warf

Summary

The `updatedAt` timestamp in the price feed response is not checked. So outdated prices may be used.

Vulnerability Detail

The following checks are performed for the chainlink price feed: https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L299-L315

As you can see the `updatedAt` timestamp is not checked. So the price may be outdated.

Impact

The price that is used by the Controller can be outdated. This means that a depeg event may be caused due to an outdated price which is incorrect. Only current prices must be used to check for a depeg event.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L273-L318

Tool used

Manual Review

Recommendation

Introduce a reasonable limit for how old the price can be and revert if the price is older:

```
iff --git a/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
↪  b/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
index 0587c86..cf2dcf5 100644
--- a/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
+++ b/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
@@ -275,8 +275,8 @@ contract ControllerPeggedAssetV2 {
             ,
             /*uint80 roundId*/
             int256 answer,
-            uint256 startedAt, /*uint256 updatedAt*/ /*uint80 answeredInRound*/
-            ,
+            uint256 startedAt,
+            uint256 updatedAt, /*uint80 answeredInRound*/

         ) = sequencerUptimeFeed.latestRoundData();

@@ -314,6 +314,8 @@ contract ControllerPeggedAssetV2 {

         if (answeredInRound < roundID) revert RoundIDOutdated();

+        if (updatedAt < block.timestamp - LIMIT) revert PriceOutdated();
+
         return price;
     }
```

Discussion

3xHarry

considering this

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/141

IAm0x52

Fix looks good. Controller will now revert if price is stale

Issue M-2: ControllerPeggedAssetV2: `triggerEndEpoch` function can be called even if epoch is null epoch leading to loss of funds

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/108

Found by

0xRobocop, 0xnirlin, 0xvj, KingNFT, berndartmueller, bin2chen, charlesjhongc, climber2002, evan, holyhansss, kenzo, libratus, ltyu, minhtrng, rogue reddwarf, warRoom, yixxas

Summary

An epoch can be resolved in three ways which correspond to the three functions available in the Controller: `triggerDepeg`, `triggerEndEpoch`, `triggerNullEpoch`.

The issue is that `triggerEndEpoch` can be called even though `triggerNullEpoch` should be called. "Null epoch" means that any of the two vaults does not have funds deposited. In this case the epoch should be resolved with `triggerNullEpoch` such that funds are not transferred from the premium vault to the collateral vault.

So in `triggerEndEpoch` is should be checked whether the conditions for a null epoch apply. If that's the case, the `triggerEndEpoch` function should revert.

Vulnerability Detail

The assumption the code makes is that if the null epoch applies, `triggerNullEpoch` will be called before the end timestamp of the epoch which is when `triggerEndEpoch` can be called.

This is not necessarily true.

`triggerNullEpoch` might not be called in time (e.g. because the epoch duration is very short or simply nobody calls it) and then the `triggerEndEpoch` function can be called which sends the funds from the premium vault into the collateral vault: https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L172-L192

If the premium vault is the vault which has funds and the collateral vault does not, then the funds sent to the collateral vault are lost.

Impact

Loss of funds for users that have deposited into the premium vault.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L144-L202

Tool used

Manual Review

Recommendation

`triggerEndEpoch` should only be callable when the conditions for a null epoch don't apply:

```
diff --git a/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
↪  b/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
index 0587c86..7b25cf3 100644
--- a/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
+++ b/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol
@@ -155,6 +155,13 @@ contract ControllerPeggedAssetV2 {
             collateralVault.epochExists(_epochId) == false
         ) revert EpochNotExist();

+        if (
+            premiumVault.totalAssets(_epochId) == 0 ||
+            collateralVault.totalAssets(_epochId) == 0
+        ) {
+            revert VaultZeroTVL();
+        }
+
         (, uint40 epochEnd, ) = premiumVault.getEpochConfig(_epochId);

         if (block.timestamp <= uint256(epochEnd)) revert EpochNotExpired();
```

Discussion

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/140

IAm0x52

Fix looks good. triggerEndEpoch can no longer be called on expired, null epochs

Issue M-3: Controller doesn't send treasury funds to the vault's treasury address

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/110

Found by

Dug, Ruhum, bin2chen, nobody2018, roguereddwarf

Summary

The Controller contract sends treasury funds to its own immutable `treasury` address instead of sending the funds to the one stored in the respective vault contract.

Vulnerability Detail

Each vault has a treasury address that is assigned on deployment which can also be updated through the factory contract:

But, the Controller, responsible for sending the fees to the treasury, uses the immutable treasury address that it was initialized with:

Impact

It's not possible to have different treasury addresses for different vaults. It's also not possible to update the treasury address of a vault although it has a function to do that. Funds will always be sent to the address the Controller was initialized with.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/VaultV2.sol#L79 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/VaultV2.sol#L265-L268

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L186
https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Controllers/ControllerPeggedAssetV2.sol#L40

Tool used

Manual Review

Recommendation

The Controller should query the Vault to get the correct treasury address, e.g.:

Discussion

3xHarry

will use one location for the treasury address which will be on the factory.

3xHarry

fixed in https://github.com/Y2K-Finance/Earthquake/pull/137

IAm0x52

Needs additional changes. Controller still sends to it's immutable address and not treasury address on factory

IAm0x52

Fix looks good. Controller has been updated to use treasury address from factory

jacksanford1

Note: 0x52 is referring to this specific commit in the last message:

https://github.com/Y2K-Finance/Earthquake/pull/137/commits/272199687465252d1da8cb1af624e90c12315953

Issue M-4: Stuck emissions for nullified epochs

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/122

Found by

0x52, Ch_301, bin2chen, carrot, cccz, hickuphh3, immeas, kenzo, libratus, ltyu, rogue reddwarf, sinarette

Summary

If either the premium and / or collateral vault has 0 TVL for an epoch with emissions, those emissions will not be withdrawable by anyone.

Vulnerability Detail

The `finalTVL` set for a vault with 0 TVL (epoch will be nullified) will be 0. As a result, emissions that were allocated to that vault are not withdrawable by anyone.

It's admittedly unlikely to happen since the `emissionsToken` is expected to be Y2K which has value and is tradeable.

Impact

Emissions cannot be recovered.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L157 https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L630-L636

Tool used

Manual Review

Recommendation

Create a function to send emissions back to the treasury if an epoch is marked as nullified.

A related issue is that if both the premium and collateral vaults have 0 TVL, only the collateral vault gets marked as nullified. Consider handling this edge case.

Discussion

3xHarry

great catch

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/139

IAm0x52

Fix looks good. setEpochNull is overridden in Carousel to transfer emissions back to treasury

Issue M-5: Malicious user can make rolloverQueue never get processed

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/172

Found by

Ace-30, ElKu, Respx, ShadowForce, TrungOre, bin2chen, ck, evan, hickuphh3, immeas, minhtrng, nobody2018, twicek

Summary

`rolloverQueue` is **shared by all epochs**. For each round of epoch, `mintRollovers` will process `rolloverQueue` from the beginning. A normal user calls `enlistInRollover` to enter the rolloverQueue, and in the next round of epoch, he will call `delistInRollover` to exit the rolloverQueue. In this case, `rolloverQueue.length` is acceptable. However, malicious user can make the `rolloverQueue.length` huge, causing the relayer to **consume a huge amount of gas for every round of epoch**. Carousel will send relayerFee to relayer in order to encourage external relayer to call `mintRollovers`. Malicious user can make external relayer unwilling to call `mintRollovers`. **Ultimately, rolloverQueue will never be processed.**

Vulnerability Detail

Let's assume the following scenario:

relayerFee is 1e18. The current epochId is E1, and the next epochId is E2. At present, `rolloverQueue` has 10 normal user `QueueItem`. Bob has deposited 1000e18 assets before the start of E1, so `balanceOf(bob, E1) = 1000e18`.

1. Bob creates 1000 addresses, each address has `setApprovalForAll` to bob. He calls two functions for each address:

   `Carousel.safeTransferFrom(bob, eachAddress, E1, 1e18)`

   `Carousel.enlistInRollover(E1, 1e18, eachAddress)`, **1e18 equal to minRequiredDeposit**.

2. `rolloverQueue.length` equals to 1010(1000+10).

These 1000 addresses will **never** call `delistInRollover` to exit the rolloverQueue, so no matter whether these addresses win or lose, **their QueueItem will always be in the rolloverQueue**. In each round of epoch, the relayer has to process at least 1000 QueueItems, and these QueueItems are useless. **Malicious users only need to do it once to cause permanent affects**.

When a normal user loses in a certain round of epoch, he may not call delistInRollover to exit the rolloverQueue. For example, he left the platform and

stopped playing. **In this case, rolloverQueue.length will become larger and larger as time goes by**.

Carousel contract will not send any relayerFee to the relayer, because these useless QueueItem will not increase the value of [executions](https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L447). Obviously, calling `mintRollovers` has no benefit for the relayer. Therefore, no relayer is willing to do this.

Impact

The relayer consumes a huge amount of gas for calling `mintRollovers` for each round of epoch. **In other words, as long as the rolloverQueue is unacceptably long, it is a permanent DOS**.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L361-L459

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L238-L271

Tool used

Manual Review

Recommendation

We should change the single queue to **queue mapping**. In this way, relayer only needs to process the queue corresponding to the epochId.

```
--- a/Earthquake/src/v2/Carousel/Carousel.sol
+++ b/Earthquake/src/v2/Carousel/Carousel.sol
@@ -23,7 +23,7 @@ contract Carousel is VaultV2 {
     IERC20 public immutable emissionsToken;

     mapping(address => uint256) public ownerToRollOverQueueIndex;
-    QueueItem[] public rolloverQueue;
+    mapping(uint256 => QueueItem[]) public rolloverQueues;
     QueueItem[] public depositQueue;
     mapping(uint256 => uint256) public rolloverAccounting;
     mapping(uint256 => mapping(address => uint256)) public _emissionsBalances;
```

Discussion

3xHarry

I would disagree with the feasibility of this attack.

1. there is a non neglectable minDeposit which makes this attack much more expensive
2. the queue can be processed in multiple transactoins and the relayerFee is supposed to be configured so much so that each processed item gas consumption is reimbursed with a profit

IAm0x52

Issue has been acknowledged by sponsor

Issue M-6: User deposit may never be entertained from deposit queue

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/174

Found by

0Kage, 0xmuxyz, Ruhum, TrungOre, ck, csanuragjain, hickuphh3, jprod15, twicek

Summary

Due to FILO (first in last out) stack structure, while dequeuing, the first few entries may never be retrieved. These means User deposit may never be entertained from deposit queue if there are too many deposits

Vulnerability Detail

1. Assume User A made a deposit which becomes 1st entry in `depositQueue`
2. Post this X more deposits were made, so `depositQueue.length=X+1`
3. Relayer calls `mintDepositInQueue` and process `X-9` deposits

```
while ((length - _operations) <= i) {
            // this loop impelements FILO (first in last out) stack to reduce gas
 ↪  cost and improve code readability
            // changing it to FIFO (first in first out) would require more code
 ↪  changes and would be more expensive
            _mintShares(
                queue[i].receiver,
                _epochId,
                queue[i].assets - relayerFee
            );
            emit Deposit(
                msg.sender,
                queue[i].receiver,
                _epochId,
                queue[i].assets - relayerFee
            );
            depositQueue.pop();
            if (i == 0) break;
            unchecked {
                i--;
            }
        }
```

4. This reduces deposit queue to only 10

5. Before relayer could process these, Y more deposits were made which increases deposit queue to `y+10`
6. This means Relayer might not be able to again process User A deposit as this deposit is lying after processing `Y+9` deposits

Impact

User deposit may remain stuck in deposit queue if a large number of deposit are present in queue and relayer is interested in dequeuing all entries

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L310

Tool used

Manual Review

Recommendation

Allow User to dequeue deposit queue based on index, so that if such condition arises, user would be able to dequeue his deposit (independent of relayer)

Discussion

3xHarry

depositing into queue should count as committing to an epoch. By giving the user the ability to delist his queue he could take advantage of market movements. However, we will raise min deposit for the queue to make DDoS very expensive.

twicek

Escalate for 10 USDC

My issues #62 and #63 are both marked as duplicate of this issue when only #63 is actually a duplicate. #63 is a duplicate of #174 who both relate to how queued deposits can get stuck in the deposit queue for various reasons. #62 however, does not describe anything related to both the deposit queue and the separate fact that a there is DoS attack vector. Instead, it relates to how relayers can get griefed because unrollable rollover items in the rollover queue can aggregate and lead them to not get paid for their work. In the duplicates, that I will cite below, this issue is achieved in various different ways but they all lead to the same impact.

Therefore, to reiterate, the #62 and #63 are different because they don't involve the same states, attack vector and users. #63 involve the deposit queue and a DoS

attack that lead to economic damage for regular users of the protocol. #62 involve the rollover queue and a griefing attack that lead to economic damage for relayers.

In my judging repos I market as duplicate of #63: #79 #82 #114 #174 #220 #274 #295 #317 #342 #431 #447 and as duplicate of #62: #80 #218 #235 #275 #284 #309 #393 #411 #475

From what I have seen all or almost all this issue are present here as duplicate of #174. A lot of people submitted them as different issues because they are indeed completely different. I might have made some mistakes in my judging, but it's mostly consistent with what I say above. Specifically, I missed #176 for which I partially agree with the escalation of securitygrid that it should not be a duplicate of #174 but it also should not be a solo finding because it is a duplicate of #62 and its duplicates.

sherlock-admin

> Escalate for 10 USDC
>
> My issues #62 and #63 are both marked as duplicate of this issue when only #63 is actually a duplicate. #63 is a duplicate of #174 who both relate to how queued deposits can get stuck in the deposit queue for various reasons. #62 however, does not describe anything related to both the deposit queue and the separate fact that a there is DoS attack vector. Instead, it relates to how relayers can get griefed because unrollable rollover items in the rollover queue can aggregate and lead them to not get paid for their work. In the duplicates, that I will cite below, this issue is achieved in various different ways but they all lead to the same impact.
>
> Therefore, to reiterate, the #62 and #63 are different because they don't involve the same states, attack vector and users. #63 involve the deposit queue and a DoS attack that lead to economic damage for regular users of the protocol. #62 involve the rollover queue and a griefing attack that lead to economic damage for relayers.
>
> In my judging repos I market as duplicate of #63: #79 #82 #114 #174 #220 #274 #295 #317 #342 #431 #447 and as duplicate of #62: #80 #218 #235 #275 #284 #309 #393 #411 #475
>
> From what I have seen all or almost all this issue are present here as duplicate of #174. A lot of people submitted them as different issues because they are indeed completely different. I might have made some mistakes in my judging, but it's mostly consistent with what I say above. Specifically, I missed #176 for which I partially agree with the escalation of securitygrid that it should not be a duplicate of #174 but it also should not be a solo finding because it is a duplicate of #62 and its duplicates.

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.

dmitriia

Agree re 62, here the issues were grouped per 'pop' allowing for various manipulations.

hrishibhat

Escalation accepted

After reviewing the issues and all its duplicates, given the complexity as well as the similarities between these issues, the fair move would be to split up these issues into two categories based on the functions of the root cause: `mintDepositInQueue` & `mintRollovers`. These two categories of duplicates primarily contain issues related to large queue lengths resulting in dos or insufficient relayer incentives from the same root cause.

sherlock-admin

> Escalation accepted
>
> After reviewing the issues and all its duplicates, given the complexity as well as the similarities between these issues, the fair move would be to split up these issues into two categories based on the functions of the root cause: `mintDepositInQueue` & `mintRollovers`. These two categories of duplicates primarily contain issues related to large queue lengths resulting in dos or insufficient relayer incentives from the same root cause.

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

IAm0x52

Issue has been acknowledged by sponsor

Issue M-7: changeTreasury() Lack of check and remove old

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/208

Found by

HonorLt, VAD37, bin2chen, nobody2018

Summary

changeTreasury() Lack of check and remove old

Vulnerability Detail

changeTreasury() used to set new treasury The code is as follows

```
function changeTreasury(uint256 _marketId, address _treasury)
    public
    onlyTimeLocker
{
    if (_treasury == address(0)) revert AddressZero();

    address[2] memory vaults = marketIdToVaults[_marketId];

    if (vaults[0] == address(0) || vaults[1] == address(0)) {
        revert MarketDoesNotExist(_marketId);
    }
    IVaultV2(vaults[0]).whiteListAddress(_treasury);
    IVaultV2(vaults[1]).whiteListAddress(_treasury);
    IVaultV2(vaults[0]).setTreasury(treasury);
    IVaultV2(vaults[1]).setTreasury(treasury);

    emit AddressWhitelisted(_treasury, _marketId);
}
```

The above code has the following problem:

1. no check whether the new treasury same as the old. If it is the same, the whitelist will be canceled.
2. Use setTreasury(VaultFactoryV2.treasury), it should be setTreasury(_treasury)
3. not cancel old treasury from the whitelist

Impact

whiteListAddress abnormal

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/VaultFactoryV2.sol#L228

Tool used

Manual Review

Recommendation

```
    function changeTreasury(uint256 _marketId, address _treasury)
        public
        onlyTimeLocker
    {
        if (_treasury == address(0)) revert AddressZero();

        address[2] memory vaults = marketIdToVaults[_marketId];

        if (vaults[0] == address(0) || vaults[1] == address(0)) {
            revert MarketDoesNotExist(_marketId);
        }

+       require(vaults[0].treasury() !=_treasury,"same"); //check same
+       IVaultV2(vaults[0]).whiteListAddress(vaults[0].treasury()); //cancel old
 ↪  whitelist
+       IVaultV2(vaults[1]).whiteListAddress(vaults[1].treasury()); //cancel old
 ↪  whitelist

        IVaultV2(vaults[0]).whiteListAddress(_treasury);
        IVaultV2(vaults[1]).whiteListAddress(_treasury);
+       IVaultV2(vaults[0]).setTreasury(_treasury);
+       IVaultV2(vaults[1]).setTreasury(_treasury);
-       IVaultV2(vaults[0]).setTreasury(treasury);
-       IVaultV2(vaults[1]).setTreasury(treasury);

        emit AddressWhitelisted(_treasury, _marketId);
    }
```

Discussion

dmitriia

Keeping it separate from 435 because of whitelist observation (1)

pauliax

Escalate for 10 USDC.

I believe it is unfair to leave it as a solo medium.

#410 also mentions the problem with whitelisting: "Also, probably the old treasury should be removed from the whitelist to prevent accidental abuse of privileges." but was grouped together with other issues from #435.

sherlock-admin

> Escalate for 10 USDC.
>
> I believe it is unfair to leave it as a solo medium.
>
> #410 also mentions the problem with whitelisting: "Also, probably the old treasury should be removed from the whitelist to prevent accidental abuse of privileges." but was grouped together with other issues from #435.

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.

3xHarry

fix Pr: https://github.com/Y2K-Finance/Earthquake/pull/137

hrishibhat

Escalation accepted

Added relevant duplicates based on whitelist observation

sherlock-admin

> Escalation accepted
>
> Added relevant duplicates based on whitelist observation

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

IAm0x52

Fixes look good. Carousel now directly uses the treasury address sent on factory

Issue M-8: mintRollovers should require entitledShares >= relayerFee

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/293

Found by

cccz, iglyx, roguereddwarf

Summary

mintRollovers should require entitledShares >= relayerFee

Vulnerability Detail

In mintRollovers, the rollover is only not skipped if queue[index].assets >= relayerFee,

```
if (entitledShares > queue[index].assets) {
    // skip the rollover for the user if the assets cannot cover the relayer fee
↪   instead of revert.
    if (queue[index].assets < relayerFee) {
        index++;
        continue;
    }
```

In fact, since the user is already profitable, entitledShares is the number of assets of the user, which is greater than queue[index].assets, so it should check that entitledShares >= relayerFee, and use entitledShares instead of queue[index].assets to subtract relayerFee when calculating assetsToMint later.

Impact

This will prevent rollover even if the user has more assets than relayerFee

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/Carousel/Carousel.sol#L401-L406

Tool used

Manual Review

Recommendation

Change to

```
                if (entitledShares > queue[index].assets) {
                    // skip the rollover for the user if the assets cannot cover
 ↪  the relayer fee instead of revert.
-                   if (queue[index].assets < relayerFee) {
+                   if (entitledShares < relayerFee) {
                        index++;
                        continue;
                    }
...
-                   uint256 assetsToMint = queue[index].assets - relayerFee;
+                   uint256 assetsToMint = entitledShares - relayerFee;
```

Discussion

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/136

IAm0x52

Needs additional changes. L423 doesn't make sense to me. queue[index].assets is in shares and entitledAmount isn't but they are subtracted directly.

3xHarry

@IAm0x52 thx for your comment, basically since `QueueItem.asset` (later renamed to shares) represents share of the epoch, i converted `relayerFee` which is denominated in underlying asset to `shares` for that epoch. Later i realized that Users only want to rollover their original deposit value, therefore i burn the original deposit Value and mint this value - relayerFeeInShares into the next epoch, the winnings sahres amount are left in the epoch to be withdrawn

jacksanford1

Bringing in some Discord discussion:

0x52

> For 293 your comment then conflicts with how you take the fee because you're charging the user the relayer fee for epoch n+1 but you're using epoch n to estimate. You should just take the fee directly not convert it into shares

3xHarry

> @0x52 thx for raising this concern: uint256 relayerFeeInShares = previewAmountInShares(queue[index].epochId, relayerFee); converts

> relayerFee into amount of shares of prev epoch (epoch users wants to rollover collateral from)
>
> uint256 assetsToMint = queue[index].assets - relayerFeeInShares;
> assets represent shares in prev epoch and arrimetic operation is done in same denominator (shares in queue[index].epochId)

0x52

> Shares and assets are 1:1 for the open epoch correct?
>
> So imagine this scenario. You deposit 100 asset into epoch 1 to get 100 shares in epoch 1. Now you queue them into the rollover.
>
> Epoch 1 ends with a profit of 25% which means your 100 shares are now worth 125. 80 shares are burned (worth 100 assets) leaving the user with 20 shares for epoch 1.
>
> If the relayer fee is 10 then it will be converted to 8 shares of epoch 2. But epoch 2 is still 1:1 with assets so it's only taking 8 assets from the user but sending them 10 asset as the relayer fee So you either need to reduce epoch 1 shares by 8 (i.e. leave the user with 12 shares) or you need to reduce assetsToMint by relayer fee directly (i.e. only mint 90 to epoch 2)

jacksanford1

Bringing in some discussion from Y2K's repo:

3xHarry

> @IAm0x52 thx for noticing the relayerFeeInShares Bug I will close this PR, but fix can be observed in 9165674

https://github.com/Y2K-Finance/Earthquake/pull/136#issuecomment-1541996529

IAm0x52

Fix looks good. Fee is no longer converted since epoch in which fee is removed is always 1:1

Issue M-9: Vault Factory ownership can be changed immediately and bypass timelock delay

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/337

Found by

ast3ros

Summary

The VaultFactoryV2 contract is supposed to use a timelock contract with a delay period when changing its owner. However, there is a loophole that allows the owner to change the owner address instantly, without waiting for the delay period to expire. This defeats the purpose of the timelock contract and exposes the VaultFactoryV2 contract to potential abuse.

Vulnerability Detail

In project description, timelock is required when making critical changes. Admin can only configure new markets and epochs on those markets.

```
2) Admin can configure new markets and epochs on those markets, Timelock can
↪  make cirital changes like changing the oracle or whitelisitng controllers.
```

The VaultFactoryV2 contract has a `changeOwner` function that is supposed to be called only by the timelock contract with a delay period.

```
function changeOwner(address _owner) public onlyTimeLocker {
        if (_owner == address(0)) revert AddressZero();
        _transferOwnership(_owner);
    }
```

The VaultFactoryV2 contract inherits from the Openzeppelin Ownable contract, which has a transferOwnership function that allows the owner to change the owner address immediately. However, the `transferOwnership` function is not overridden by the `changeOwner` function, which creates a conflict and a vulnerability. The owner can bypass the timelock delay and use the transferOwnership function to change the owner address instantly.

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}
```

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/VaultFactoryV2.sol#L325-L328

Impact

The transferOwnership is not worked as design (using timelock), the timelock delay become useless. This means that if the owner address is hacked or corrupted, the attacker can take over the contract immediately, leaving no time for the protocol and the users to respond or intervene.

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/main/Earthquake/src/v2/VaultFactoryV2.sol#L325-L328

Tool used

Manual Review

Recommendation

Override the `transferOwnership` function and add modifier `onlyTimeLocker`.

Discussion

thangtranth

Escalate for 10 USDC.

This issue is different from #501 and cannot be ignored. It is not related to using two steps to change ownership. The problem here is that the transferOwnership function in the Ownable contract is not overridden as it should be. This allows the owner to change the ownership without going through the timelock. This creates a severe security risk and undermines the trust and transparency of the protocol as stated in spec.

sherlock-admin

> Escalate for 10 USDC.
>
> This issue is different from #501 and cannot be ignored. It is not related to using two steps to change ownership. The problem here is that the transferOwnership function in the Ownable contract is not overridden as it should be. This allows the owner to change the ownership without going through the timelock. This creates a severe security risk and undermines the trust and transparency of the protocol as stated in spec.

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.

hrishibhat

Escalation accepted

Not a duplicate of #501 and can be considered a valid medium since this identifies the issue that `transferOwnership` is not overridden and needs to have ‘onlyTimeLocker' modifier,

sherlock-admin

> Escalation accepted
>
> Not a duplicate of #501 and can be considered a valid medium since this identifies the issue that `transferOwnership` is not overridden and needs to have ‘onlyTimeLocker' modifier,

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

hrishibhat

Lead Judge comment:

> looks valid, maybe med, if they intend to do it without a delay is one thing and to be documented, but if a function just left not overriden it's a bug

Sponsor comment:

> Actually thats valid issue, ... fixing this will make this action more complicated. My thinking is to add a direct function on timelocker which lets timelocker execute the owner (deployer) change without 7day queue.

3xHarry

FIX RP: https://github.com/Y2K-Finance/Earthquake/pull/147 - last two commits

IAm0x52

Fix looks good. changeOwner has been removed and transferOwnership has been overridden to allow only timelocker

Issue M-10: `Carousel.mintRollovers` potentially mints `0` shares and can grief rollover queue

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/418

Found by

berndartmueller, evan, kenzo

Summary

If the deposited assets for a queued rollover item are equal to the relayer fee, the rollover will be minted with `0` shares, potentially leading to zero TVL and hence `finalTVL[_id] = 0`. This will cause the `previewWithdraw` call to revert due to division by zero and the rollover queue will be stuck forever.

Vulnerability Detail

Minting rollovers in the carousel vault iterates over all items in the `rolloverQueue` queue. Each item is processed, and the entitled shares (`entitledShares`) are calculated using `previewWithdraw`. If the
entitled shares are greater than the deposited assets), the rollover is minted.

However, if the deposited assets for the queued item are equal to the relayer fee, the assets to mint (`assetsToMint`) calculated in line 436 will be `0`.

If this happens to be the only deposit (mint) for the epoch and the vaults TVL remains zero, the `previewWithdraw` call in line 396 will revert due to division by zero.

Impact

Once there is a rollover minted with `0` shares for an epoch and the vaults TVL (i.e., `finalTVL`) remains zero, the rollover queue will be stuck forever unless the owner of this queue item delists it.

Code Snippet

src/v2/Carousel/Carousel.mintRollovers

```
361: function mintRollovers(uint256 _epochId, uint256 _operations)
362:     external
363:     epochIdExists(_epochId)
364:     epochHasNotStarted(_epochId)
365:     nonReentrant
366: {
...      // [...]
```

```
392:
393:     while ((index - prevIndex) < (_operations)) {
394:         // only roll over if last epoch is resolved
395:         if (epochResolved[queue[index].epochId]) {
396: @>          uint256 entitledShares = previewWithdraw( // @audit-info reverts if epoch's `finalTVL` == 0
397:                 queue[index].epochId,
398:                 queue[index].assets
399:             );
400:             // mint only if user won epoch he is rolling over
401:             if (entitledShares > queue[index].assets) {
402:                 // skip the rollover for the user if the assets cannot cover the relayer fee instead of revert.
403:                 if (queue[index].assets < relayerFee) {
404:                     index++;
405:                     continue;
406:                 }
407:                 // @note we know shares were locked up to this point
408:                 _burn(
409:                     queue[index].receiver,
410:                     queue[index].epochId,
411:                     queue[index].assets
412:                 );
413:                 // transfer emission tokens out of contract otherwise user could not access them as vault shares are burned
414:                 _burnEmissions(
415:                     queue[index].receiver,
416:                     queue[index].epochId,
417:                     queue[index].assets
418:                 );
419:                 // @note emission token is a known token which has no before transfer hooks which makes transfer safer
420:                 emissionsToken.safeTransfer(
421:                     queue[index].receiver,
422:                     previewEmissionsWithdraw(
423:                         queue[index].epochId,
424:                         queue[index].assets
425:                     )
426:                 );
427:
428:                 emit Withdraw(
429:                     msg.sender,
430:                     queue[index].receiver,
431:                     queue[index].receiver,
432:                     _epochId,
433:                     queue[index].assets,
434:                     entitledShares
435:                 );
```

```
436: @>              uint256 assetsToMint = queue[index].assets - relayerFee; //
↪  @audit-info `assetsToMint` can potentially become 0
437:                 _mintShares(queue[index].receiver, _epochId, assetsToMint);
438:                 emit Deposit(
439:                     msg.sender,
440:                     queue[index].receiver,
441:                     _epochId,
442:                     assetsToMint
443:                 );
444:                 rolloverQueue[index].assets = assetsToMint;
445:                 rolloverQueue[index].epochId = _epochId;
446:                 // only pay relayer for successful mints
447:                 executions++;
448:             }
449:         }
450:         index++;
451:     }
452:
...      // [...]
459: }
```

src/v2/VaultV2.previewWithdraw

```
357: function previewWithdraw(uint256 _id, uint256 _assets)
358:     public
359:     view
360:     override(SemiFungibleVault)
361:     returns (uint256 entitledAmount)
362: {
363:     // entitledAmount amount is derived from the claimTVL and the finalTVL
364:     // if user deposited 1000 assets and the claimTVL is 50% lower than
↪  finalTVL, the user is entitled to 500 assets
365:     // if user deposited 1000 assets and the claimTVL is 50% higher than
↪  finalTVL, the user is entitled to 1500 assets
366:     entitledAmount = _assets.mulDivDown(claimTVL[_id], finalTVL[_id]);
367: }
```

Tool used

Manual Review

Recommendation

Consider checking the total assets of the epoch `queue[index].epochId` to be greater than 0 before calling `previewWithdraw` in line 396.

Discussion

3xHarry

will move check from line `403` up before `previewWithdraw`, also considering implementing rollover delisting if `assetsToMint` is less than relayerFee

3xHarry

in general delisting of stale rollovers (not enough to pay for relayer, or not won prev epoch) should be delisted by smart contract.

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/133

IAm0x52

Needs additional changes. This still doesn't address the issue of minting 0 because if assets == relayerFee then it will still mint 0. Should instead be:

```
if (queue[index].assets <= relayerFee) {
```

IAm0x52

Fix looks good. Suggested change above has been added

jacksanford1

Note: 0x52 referenced this commit in their second message from PR #133:

https://github.com/Y2K-Finance/Earthquake/pull/133/commits/9edaa8a5da96edf7c61bef4a8847f7c107f2b630

Issue M-11: Arbitrum sequencer downtime lasting before and beyond epoch expiry prevents triggering depeg

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/422

Found by

Dug, Respx, ShadowForce, berndartmueller, holyhansss, libratus, ltyu, spyrosonic10

Summary

A depeg event can not be triggered if the Arbitrum sequencer went down before the epoch ends and remains down beyond the epoch expiry. Instead, the collateral vault users can unfairly end the epoch without a depeg and claim the premium payments.

Vulnerability Detail

A depeg event can be triggered during an ongoing epoch by calling the `ControllerPeggedAssetV2.triggerDepeg` function. This function retrieves the latest price of the pegged asset via the `getLatestPrice` function.

If the Arbitrum sequencer is down or the grace period has not passed after the sequencer is back up, the `getLatestPrice` function reverts and the depeg event can not be triggered.

In case the sequencer went down before the epoch expired and remained down well after the epoch expired, a depeg can not be triggered, and instead, the epoch can be incorrectly ended without a depeg by calling the `ControllerPeggedAssetV2.triggerEndEpoch` function. Incorrectly, because at the time of the epoch expiry, it was not possible to trigger a depeg and hence it would be unfair to end the epoch without a depeg.

Impact

A depeg event can not be triggered, and premium vault users lose out on their insurance payout, while collateral vault users can wrongfully end the epoch and claim the premium.

Code Snippet

v2/Controllers/ControllerPeggedAssetV2.sol - triggerDepeg()

```
051: function triggerDepeg(uint256 _marketId, uint256 _epochId) public {
052:     address[2] memory vaults = vaultFactory.getVaults(_marketId);
053:
```

```
054:     if (vaults[0] == address(0) || vaults[1] == address(0))
055:         revert MarketDoesNotExist(_marketId);
056:
057:     IVaultV2 premiumVault = IVaultV2(vaults[0]);
058:     IVaultV2 collateralVault = IVaultV2(vaults[1]);
059:
060:     if (premiumVault.epochExists(_epochId) == false) revert EpochNotExist();
061:
062:     int256 price = getLatestPrice(premiumVault.token());
063:
064:     if (int256(premiumVault.strike()) <= price)
065:         revert PriceNotAtStrikePrice(price);
066:
...      // [...]
138: }
```

v2/Controllers/ControllerPeggedAssetV2.sol - getLatestPrice()

```
273: function getLatestPrice(address _token) public view returns (int256) {
274:     (
275:         ,
276:         /*uint80 roundId*/
277:         int256 answer,
278:         uint256 startedAt, /*uint256 updatedAt*/ /*uint80 answeredInRound*/
279:         ,
280:
281:     ) = sequencerUptimeFeed.latestRoundData();
282:
283:     // Answer == 0: Sequencer is up
284:     // Answer == 1: Sequencer is down
285:     bool isSequencerUp = answer == 0;
286:     if (!isSequencerUp) {
287:         revert SequencerDown();
288:     }
289:
290:     // Make sure the grace period has passed after the sequencer is back up.
291:     uint256 timeSinceUp = block.timestamp - startedAt;
292:     if (timeSinceUp <= GRACE_PERIOD_TIME) {
293:         revert GracePeriodNotOver();
294:     }
295:
...      // [...]
318: }
```

Tool used

Manual Review

Recommendation

Consider adding an additional "challenge" period (with reasonable length of time) after the epoch has expired and before the epoch end can be triggered without a depeg.

Within this challenge period, anyone can claim a depeg has happened during the epoch's expiry and trigger the epoch end. By providing the Chainlink round id's for both feeds (sequencer and price) at the time of the epoch expiry (`epochEnd`), the claim can be verified to assert that the sequencer was down and the strike price was reached.

Discussion

3xHarry

We are aware of this mechanic, however, users prefer to have the atomicity of instant settlement, this is so that users can utilize farming y2k tokens most effectively by rotating from one epoch to the next. Users are made aware of the risks when using chainlink oracles as well as the execution environment being on Arbitrum.

pauliax

Escalate for 10 USDC.

I believe this should be low severity because it falls under the misbehaving of infrastructure and integrations:

Q: In case of external protocol integrations, are the risks of an external protocol 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: [NOT ACCEPTABLE]*

sherlock-admin

> Escalate for 10 USDC.
>
> I believe this should be low severity because it falls under the misbehaving of infrastructure and integrations:
>
> ***Q: In case of external protocol integrations, are the risks of an external protocol 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: [NOT 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.

hrishibhat

Escalation rejected

Valid medium This is a valid issue as the readme indicates that risks associated with external integrations are not acceptable. That means issues are acceptable.

However, Sherlock acknowledges the escalator's concern about some of these issues and will consider addressing them in the next update of the judging guidelines.

sherlock-admin

> Escalation rejected
>
> Valid medium This is a valid issue as the readme indicates that risks associated with external integrations are not acceptable. That means issues are acceptable.
>
> However, Sherlock acknowledges the escalator's concern about some of these issues and will consider addressing them in the next update of the judging guidelines.

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

IAm0x52

Issue has been acknowledged by sponsor

Issue M-12: VaultFactoryV2#changeTreasury misconfigures the vault

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/435

Found by

0x52, 0xnirlin, Dug, ElKu, TrungOre, ast3ros, holyhansss, ni8mare, roguereddwarf, spyrosonic10, volodya, warRoom

Summary

VaultFactoryV2#changeTreasury misconfigures the vault because the setTreasury subcall uses the wrong variable

Vulnerability Detail

VaultFactoryV2.sol#L228-L246

```
function changeTreasury(uint256 _marketId, address _treasury)
    public
    onlyTimeLocker
{
    if (_treasury == address(0)) revert AddressZero();

    address[2] memory vaults = marketIdToVaults[_marketId];

    if (vaults[0] == address(0) || vaults[1] == address(0)) {
        revert MarketDoesNotExist(_marketId);
    }

    IVaultV2(vaults[0]).whiteListAddress(_treasury);
    IVaultV2(vaults[1]).whiteListAddress(_treasury);
    IVaultV2(vaults[0]).setTreasury(treasury);
    IVaultV2(vaults[1]).setTreasury(treasury);

    emit AddressWhitelisted(_treasury, _marketId);
}
```

When setting the treasury for the underlying vault pair it accidentally use the treasury variable instead of _treasury. This means it uses the local VaultFactoryV2 treasury rather than the function input.

ControllerPeggedAssetV2.sol#L111-L123

```
premiumVault.sendTokens(_epochId, premiumFee, treasury);
premiumVault.sendTokens(
    _epochId,
    premiumTVL - premiumFee,
    address(collateralVault)
);
// strike price is reached so collateral is still entitled to premiumTVL -
 ↪  premiumFee but looses collateralTVL
collateralVault.sendTokens(_epochId, collateralFee, treasury);
collateralVault.sendTokens(
    _epochId,
    collateralTVL - collateralFee,
    address(premiumVault)
);
```

This misconfiguration can be damaging as it may cause the triggerDepeg call in the controller to fail due to the sendToken subcall. Additionally the time lock is the one required to call it which has a minimum of 3 days wait period. The result is that valid depegs may not get paid out since they are time sensitive.

Impact

Valid depegs may be missed due to misconfiguration

Code Snippet

ControllerPeggedAssetV2.sol#L51-L138

Tool used

Manual Review

Recommendation

Set to _treasury rather than treasury.

Discussion

3xHarry

good catch!

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/132

IAm0x52

Fix looks good. setTreasury now correctly uses `_treasury` rather than `treasury`

Issue M-13: Null epochs will freeze rollovers

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/442

Found by

0x52, berndartmueller, bin2chen, iglyx, p0wd3r

Summary

When rolling a position it is required that the user didn't payout on the last epoch. The issue with the check is that if a null epoch is triggered then rollovers will break even though the vault didn't make a payout

Vulnerability Detail

Carousel.sol#L401-L406

```
uint256 entitledShares = previewWithdraw(
    queue[index].epochId,
    queue[index].assets
);
// mint only if user won epoch he is rolling over
if (entitledShares > queue[index].assets) {
```

When minting rollovers the following check is made so that the user won't automatically roll over if they made a payout last epoch. This check however will fail if there is ever a null epoch. Since no payout is made for a null epoch it should continue to rollover but doesn't.

Impact

Rollover will halt after null epoch

Code Snippet

Carousel.sol#L361-L459

Tool used

Manual Review

Recommendation

Change to less than or equal to:

```
-           if (entitledShares > queue[index].assets) {
+           if (entitledShares >= queue[index].assets) {
```

Discussion

3xHarry

makes sense

3xHarry

Won't be able to fix this edge case. Changes in the rollover queue make it now that positions are not deleted anymore but rather marked to 0 to prevent rollover queue manipulation. In this case, users would have to resolve their stuck rollover position manually. https://github.com/Y2K-Finance/Earthquake/pull/127

IAm0x52

Issue has been acknowledged by sponsor

Issue M-14: Inconsistent use of epochBegin could lock user funds

Source: https://github.com/sherlock-audit/2023-03-Y2K-judging/issues/480

Found by

Inspex, KingNFT, TrungOre, b4by_y0d4, berndartmueller, datapunk, evan, minhtrng, rogueredddwarf, sinarette, toshii, volodya, yixxas

Summary

The epochBegin timestamp is used inconsistently and could lead to user funds being locked.

Vulnerability Detail

The function `ControllerPeggedAssetV2.triggerNullEpoch` checks for timestamp like this:

```
if (block.timestamp < uint256(epochStart)) revert EpochNotStarted();
```

The modifier `epochHasNotStarted` (used by `Carousel.deposit`) checks it like this:

```
if (block.timestamp > epochConfig[_id].epochBegin)
    revert EpochAlreadyStarted();
```

Both functions can be called when `block.timestamp == epochBegin`. This could lead to a scenario where a deposit happens after `triggerNullEpoch` is called (both in the same block). Because `triggerNullEpoch` sets the value for `finalTVL`, the TVL that comes from the deposit is not accounted for. If emissions have been distributed this epoch, this will lead to the incorrect distribution of emissions and once all emissions have been claimed the remaining assets will not be claimable, due to reversion in `withdraw` when trying to send emissions:

```
function previewEmissionsWithdraw(uint256 _id, uint256 _assets)
    public
    view
    returns (uint256 entitledAmount)
{
    entitledAmount = _assets.mulDivDown(emissions[_id], finalTVL[_id]);
}
...
//in withdraw:
uint256 entitledEmissions = previewEmissionsWithdraw(_id, _assets);
```

```
if (epochNull[_id] == false) {
    entitledShares = previewWithdraw(_id, _assets);
} else {
    entitledShares = _assets;
}
if (entitledShares > 0) {
    SemiFungibleVault.asset.safeTransfer(_receiver, entitledShares);
}
if (entitledEmissions > 0) {
    emissionsToken.safeTransfer(_receiver, entitledEmissions);
}
```

The above could also lead to revert through division by 0 if `finalTVL` is set to 0, even though the deposit after was successful.

Impact

incorrect distribution, Loss of deposited funds

Code Snippet

https://github.com/sherlock-audit/2023-03-Y2K/blob/ae7f210d8fbf21b9abf09ef30edfa548f7ae1aef/Earthquake/src/v2/VaultV2.sol#L433

Tool used

Manual Review

Recommendation

The modifier `epochHasNotStarted` should use >= as comparator

Discussion

3xHarry

fix PR: https://github.com/Y2K-Finance/Earthquake/pull/130

IAm0x52

Fix looks good to me. Small inequality change for consistency