SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Index
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: May 19 - June 14, 2023
Prepared on: August 1, 2023

Introduction

The Index Coop builds decentralized structured products that make crypto simple, accessible, and secure

Scope

Repository: IndexCoop/index-coop-smart-contracts

Branch: master

Commit: 317dfb677e9738fc990cf69d198358065e8cb595

Repository: IndexCoop/index-protocol

Branch: master

Commit: 86be7ee76d9a7e4f7e93acfc533216ebef791c89

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
9	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

volodya
0xGoodess
BugBusters
0xStalin
lil.eth
rvierdiiev
Cryptor
Bauer
saidam017
jasonxiale
MohammedRizwan
0x8chars
sashik_eth
Saeedalipoor01988
kutugu
Bauchibred
Ruhum
shogoki
Diana
Phantasmagoria
0x007
Oxsadeeq
Brenzee
Ocean_Sky
kn0t
Madalad
n33k
0x52
ShadowForce
bitsurfer
warRoom
oxchryston
erictee
hildingr
whitehat

Issue H-1: eMode implementation is completely broken

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/251

Found by

0x52, 0xGoodess, 0xStalin, Cryptor, hildingr, volodya

Summary

Enabling eMode allows assets of the same class to be borrowed at much higher a much higher LTV. The issue is that the current implementation makes the incorrect calls to the Aave V3 pool making so that the pool can never take advantage of this higher LTV.

Vulnerability Detail

AaveLeverageStrategyExtension.sol#L1095-L1109

```
function _calculateMaxBorrowCollateral(ActionInfo memory _actionInfo, bool
↪  _isLever) internal view returns(uint256) {

    // Retrieve collateral factor and liquidation threshold for the collateral
    ↪  asset in precise units (1e16 = 1%)
    ( , uint256 maxLtvRaw, uint256 liquidationThresholdRaw, , , , , , ,) =
    ↪  strategy.aaveProtocolDataProvider.getReserveConfigurationData(address(st
    ↪  rategy.collateralAsset));

    // Normalize LTV and liquidation threshold to precise units. LTV is measured
    ↪  in 4 decimals in Aave which is why we must multiply by 1e14
    // for example ETH has an LTV value of 8000 which represents 80%
    if (_isLever) {
        uint256 netBorrowLimit = _actionInfo.collateralValue
            .preciseMul(maxLtvRaw.mul(10 ** 14))
            .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLe
            ↪  veragePercentage));

        return netBorrowLimit
            .sub(_actionInfo.borrowValue)
            .preciseDiv(_actionInfo.collateralPrice);
```

When calculating the max borrow/repay allowed, the contract uses the getReserveConfigurationData subcall to the pool.

AaveProtocolDataProvider.sol#L77-L100

```
function getReserveConfigurationData(
  address asset
)
  external
  view
  override
  returns (
      ...
  )
{
  DataTypes.ReserveConfigurationMap memory configuration =
  ↪  IPool(ADDRESSES_PROVIDER.getPool())
    .getConfiguration(asset);

  (ltv, liquidationThreshold, liquidationBonus, decimals, reserveFactor, ) =
  ↪  configuration
    .getParams();
```

The issue with using getReserveConfigurationData is that it always returns the default settings of the pool. It never returns the adjusted eMode settings. This means that no matter the eMode status of the set token, it will never be able to borrow to that limit due to calling the incorrect function.

It is also worth considering that the set token as well as other integrated modules configurations/settings would assume this higher LTV. Due to this mismatch, the set token would almost guaranteed be misconfigured which would lead to highly dangerous/erratic behavior from both the set and it's integrated modules. Due to this I believe that a high severity is appropriate.

Impact

Usage of eMode, a core function of the contracts, is completely unusable causing erratic/dangerous behavior

Code Snippet

AaveLeverageStrategyExtension.sol#L1095-L1109

Tool used

Manual Review

Recommendation

Pull the adjusted eMode settings rather than the base pool settings

Discussion

ckoopmann

Yep, this is correct, and will be addressed.

Btw: I think I saw a bunch of duplicates of this issue when looking through the unfiltered list.

0xffff11

Agree with sponsor, valid high. Added missing duplicates

ckoopmann

Fixed in below pr by keeping track of the current eMode category id and then getting the data for that specific eMode (if eMode category is not 0):
https://github.com/IndexCoop/index-coop-smart-contracts/pull/142

IAm0x52

Fix looks good. If emode is activated then it will use emode ltv and liquidation threshold

Issue H-2: _calculateMaxBorrowCollateral calculates repay incorrectly and can lead to set token liquidation

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/254

Found by

0x52

Summary

When calculating the amount to repay, `_calculateMaxBorrowCollateral` incorrectly applies `unutilizedLeveragePercentage` when calculating `netRepayLimit`. The result is that if the `borrowValue` ever exceeds `liquidationThreshold * (1 - unutilizedLeveragPercentage)` then all attempts to repay will revert.

Vulnerability Detail

AaveLeverageStrategyExtension.sol#L1110-L1118

```
} else {
    uint256 netRepayLimit = _actionInfo.collateralValue
        .preciseMul(liquidationThresholdRaw.mul(10 ** 14))
        .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLevera
        ↪  gePercentage));

    return _actionInfo.collateralBalance
        .preciseMul(netRepayLimit.sub(_actionInfo.borrowValue))
        .preciseDiv(netRepayLimit);
}
```

When calculating `netRepayLimit`, `_calculateMaxBorrowCollateral` uses the `liquidationThreshold` adjusted by `unutilizedLeveragePercentage`. It then subtracts the borrow value from this limit. This is problematic because if the current `borrowValue` of the set token exceeds `liquidationThreshold * (1 - unutilizedLeveragPercentage)` then this line will revert making it impossible to make any kind of repayment. Once no repayment is possible the set token can't rebalance and will be liquidated.

Impact

Once the leverage exceeds a certain point the set token can no longer rebalance

Code Snippet

AaveLeverageStrategyExtension.sol#L1110-L1118

Tool used

Manual Review

Recommendation

Don't adjust the max value by `unutilizedLeveragPercentage`

Discussion

pblivin0x

The outlined issue and fix LGTM. We need to loosen the performed `netRepayLimit` check to avoid the case where we have high leverage and can't submit repayment (`borrowValue > liquidationThreshold * (1 - unutilizedLeveragPercentage)`)

ckoopmann

Fixed in the below PR by removing the `unutilizedLeveragPercentage` adjustment as suggested: https://github.com/IndexCoop/index-coop-smart-contracts/pull/142

IAm0x52

Fix looks good. unutilizedLeveragePercentage is no longer used when calculating net repay

Issue M-1: setIncentiveSettings would be halt during a rebalance operation that gets stuck due to supply cap is reached at Aave

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/10

Found by

0xGoodess

Summary

setIncentiveSettings would be halt during a rebalance operation that gets stuck due to supply cap is reached at Aave

Vulnerability Detail

rebalance implement a cap of tradeSize and if the need to rebalance require taking more assets than the maxTradeSize, then `twapLeverageRatio` would be set to the targeted leverage. `twapLeverageRatio == 0` is required during rebalance.

Consider:

lever is needed during rebalance, the strategy require to borrow more ETH and sell to wstETH during the 1st call of rebalance the protocol cache the new twapLeverageRatio However wstETH market in Aave reach supply cap. rebalance/iterateRebalance comes to a halt. `twapLeverageRatio` remains caching the targeted leverage

setIncentiveSettings requires a condition in which no rebalance is in progress. With the above case, setIncentiveSettings can be halted for an extended period of time until the wstETH market falls under supply cap.

Worth-noting, at the time of writing this issue,
the wstETH market at Aave has been at supply cap

In this case, malicious actor who already has a position in wstETH can do the following:

- deposit into the setToken, trigger a rebalance.
- malicious trader withdraw his/her position in Aave wstETH market so there opens up vacancy for supply again.
- protocol owner see supply vacancy, call rebalance in order to lever as required. Now twapLeverageRatio is set to new value since multiple trades are needed

- malicious trader now re-supply the wstETH market at Aave so it reaches supply cap again.
- the protocol gets stuck with a non-zero twapLeverageRatio, `setIncentiveSettings` can not be called.

```
function setIncentiveSettings(IncentiveSettings memory _newIncentiveSettings)
 ↪  external onlyOperator noRebalanceInProgress {
    incentive = _newIncentiveSettings;

    _validateNonExchangeSettings(methodology, execution, incentive);

    emit IncentiveSettingsUpdated(
        incentive.etherReward,
        incentive.incentivizedLeverageRatio,
        incentive.incentivizedSlippageTolerance,
        incentive.incentivizedTwapCooldownPeriod
    );
}
```

Impact

setIncentiveSettings would be halt.

Code Snippet

https://github.com/sherlock-audit/2023-05-Index/blob/main/index-coop-smart-contracts/contracts/adapters/AaveLeverageStrategyExtension.sol#L484-L495

Tool used

Manual Review

Recommendation

Add some checks on whether the supply cap of an Aave market is reached during a rebalance. If so, allows a re-set of twapLeverageRatio

Discussion

ckoopmann

This is another scenario, that we will investigate in more detail.

pblivin0x

In the listed vulnerability, it is proposed that a

`malicious actor who already has a position in wstETH can deposit into the setToken, trigger a rebalance.`

But I don't believe this is the case. Unpermissioned actors can *mint* the SetToken with exact replication via the DebtIssuanceModuleV2. In this case the leverage ratio would remain the same as before the mint and not trigger a rebalance.

I believe the current plan for avoiding any Aave supply cap issues is by imposing a SetToken supply cap.

0xffff11

As discussed with sponsor, valid medium

ckoopmann

Fixed the issue of settings being bricked in the mentioned scenario by adding an override flag that can be set by the operator: https://github.com/IndexCoop/index-coop-smart-contracts/pull/142/commits/edbe0b04a1966ada1e0a4f9c89cbb9e2f475a440

Generally I don't see a way to reliably protect against hitting the supply cap, however it should not endanger users funds as redeeming as well as levering down are not affected. (only minting new set tokens as well as levering up would be blocked, which is a know limitation)

IAm0x52

Fix looks good. Operator can now manually override the `noRebalanceInProgress` modifier

Issue M-2: Protocol doesn't completely protect itself from `LTV = 0` tokens

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/159

Found by

Bauchibred, hildingr, volodya

Summary

The AaveLeverageStrategyExtension does not completely protect against tokens with a Loan-to-Value (LTV) of 0. Tokens with an LTV of 0 in Aave V3 pose significant risks, as they cannot be used as collateral to borrow upon a breaking withdraw. Moreover, LTVs of assets could be set to 0, even though they currently aren't, it could create substantial problems with potential disruption of multiple functionalities. This bug could cause a Denial-of-Service (DoS) situation in some cases, and has a potential to impact the borrowing logic in the protocol, leading to an unintentionally large perceived borrowing limit.

Vulnerability Detail

When an AToken has LTV = 0, Aave restricts the usage of certain operations. Specifically, if a user owns at least one AToken as collateral with an LTV = 0, certain operations could revert:

1. **Withdraw**: If the asset being withdrawn is collateral and the user is borrowing something, the operation will revert if the withdrawn collateral is an AToken with LTV > 0.
2. **Transfer**: If the asset being transferred is an AToken with LTV > 0 and the sender is using the asset as collateral and is borrowing something, the operation will revert.
3. **Set the reserve of an AToken as non-collateral**: If the AToken being set as non-collateral is an AToken with LTV > 0, the operation will revert.

Take a look at AaveLeverageStrategyExtension.sol#L1050-L1119

```
/**
 * Calculate total notional rebalance quantity and chunked rebalance quantity in
 ↪  collateral units.
 *
 * return uint256          Chunked rebalance notional in collateral units
 * return uint256          Total rebalance notional in collateral units
 */
function _calculateChunkRebalanceNotional(
```

```
    LeverageInfo memory _leverageInfo,
    uint256 _newLeverageRatio,
    bool _isLever
)
    internal
    view
    returns (uint256, uint256)
{
    // Calculate absolute value of difference between new and current leverage
↪   ratio
    uint256 leverageRatioDifference = _isLever ?
↪   _newLeverageRatio.sub(_leverageInfo.currentLeverageRatio) :
↪   _leverageInfo.currentLeverageRatio.sub(_newLeverageRatio);

    uint256 totalRebalanceNotional =
↪   leverageRatioDifference.preciseDiv(_leverageInfo.currentLeverageRatio).preci
↪   seMul(_leverageInfo.action.collateralBalance);

    uint256 maxBorrow = _calculateMaxBorrowCollateral(_leverageInfo.action,
↪   _isLever);

    uint256 chunkRebalanceNotional = Math.min(Math.min(maxBorrow,
↪   totalRebalanceNotional), _leverageInfo.twapMaxTradeSize);

    return (chunkRebalanceNotional, totalRebalanceNotional);
}

/**
 * Calculate the max borrow / repay amount allowed in base units for lever /
↪   delever. This is due to overcollateralization requirements on
 * assets deposited in lending protocols for borrowing.
 *
 * For lever, max borrow is calculated as:
 * (Net borrow limit in USD - existing borrow value in USD) / collateral asset
↪   price adjusted for decimals
 *
 * For delever, max repay is calculated as:
 * Collateral balance in base units * (net borrow limit in USD - existing borrow
↪   value in USD) / net borrow limit in USD
 *
 * Net borrow limit for levering is calculated as:
 * The collateral value in USD * Aave collateral factor * (1 - unutilized
↪   leverage %)
 *
 * Net repay limit for delevering is calculated as:
 * The collateral value in USD * Aave liquiditon threshold * (1 - unutilized
↪   leverage %)
 *
```

```
     * return uint256          Max borrow notional denominated in collateral asset
     */
    function _calculateMaxBorrowCollateral(ActionInfo memory _actionInfo, bool
    ↪  _isLever) internal view returns(uint256) {

        // Retrieve collateral factor and liquidation threshold for the collateral
    ↪  asset in precise units (1e16 = 1%)
        ( , uint256 maxLtvRaw, uint256 liquidationThresholdRaw, , , , , , ,) =
    ↪  strategy.aaveProtocolDataProvider.getReserveConfigurationData(address(strate
    ↪  gy.collateralAsset));

        // Normalize LTV and liquidation threshold to precise units. LTV is measured
    ↪  in 4 decimals in Aave which is why we must multiply by 1e14
        // for example ETH has an LTV value of 8000 which represents 80%
        if (_isLever) {
            uint256 netBorrowLimit = _actionInfo.collateralValue
                .preciseMul(maxLtvRaw.mul(10 ** 14))
                .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLe
    ↪  veragePercentage));

            return netBorrowLimit
                .sub(_actionInfo.borrowValue)
                .preciseDiv(_actionInfo.collateralPrice);
        } else {
            uint256 netRepayLimit = _actionInfo.collateralValue
                .preciseMul(liquidationThresholdRaw.mul(10 ** 14))
                .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLe
    ↪  veragePercentage));

            return _actionInfo.collateralBalance
                .preciseMul(netRepayLimit.sub(_actionInfo.borrowValue))
                .preciseDiv(netRepayLimit);
        }
    }
```

Apart from the aforementioned issue with `LTV = 0` tokens, there's another issue with the `_calculateMaxBorrowCollateral()` function. When LTV = 0, `maxLtvRaw` also equals 0, leading to a `netBorrowLimit` of 0. When the borrowing value is subtracted from this, it results in an underflow, causing the borrowing limit to appear incredibly large. This essentially breaks the borrowing logic of the protocol.

Impact

This bug could potentially disrupt the entire borrowing logic within the protocol by inflating the perceived borrowing limit. This could lead to users borrowing an unlimited amount of assets due to the underflow error. In extreme cases, this could lead to a potential loss of user funds or even a complete protocol shutdown, thus

impacting user trust and the overall functionality of the protocol.

Code Snippet

AaveLeverageStrategyExtension.sol#L1050-L1119

Tool used

Manual Review

Recommendation

The protocol should consider implementing additional protections against tokens with an LTV of 0.

Discussion

ckoopmann

This raises a good point that wasn't fully considered during the design. We will investigate this.

Generally I don't think the "Impact" section is accurate though as users cannot borrow assets on behalf of the token. However if the previous information is accurate it could have an affect on issuance / redemption if the aToken transfers are blocked.

ckoopmann

I set this to confirmed as the issue raises valid questions / scenarios that weren't fully considered during the design. Unfortunately the recommendation is pretty vague and not very actionable.

After digging into this audit report it seems that at least there are protections in place on aave side that no malicious user could produce such a situation by sending us aTokens with ltv=0 or something like that. So the only scenario where this situation could arise would be if aave governance sets the LTV of the collateral token we are using to 0.

One potential change that could allow us to delever the token in such a situation could be to add flashloan based delevering as also suggested in this issue: https://github.com/sherlock-audit/2023-05-Index-judging/issues/255

ckoopmann

Also see this issue on the morpho spearbit audit for reference. (You will have to create an account solodit to access it) - note that the "Vulernability Detail" section seems to be copy / pasted from that report: https://solodit.xyz/issues/16216

ckoopmann

After rereading this more carefully it seems the above listed limitations only affect a second aToken with LTV > 0. So for this scenario to come into effect we would have to have two aTokens as components in the set, one of which would have `LTV = 0` and the other one (which would not be able to be transfered anymore) would have `LTV > 0`.

While the issue seems to be valid, if the above understanding is correct we might keep the logic as is and list this as an explicit limitation. Because the strategy extension is designed to work with only one aToken anyway. (The Set Token could have another aToken as a component that is not managed as part of the leverage strategy but this should be avoided and is certainly not part of the expected use)

0xffff11

`While the issue seems to be valid, if the above understanding is correct we might keep the logic as is and list this as an explicit limitation` Keeping the issue as a med because there is still a small possibility for this to happen even though it is not the intended behavior: `he Set Token could have another aToken as a component that is not managed as part of the leverage strategy but this should be avoided and is certainly not part of the expected use`

Issue M-3: Loss of user funds - unchecked Return of ERC20 Transfer

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/169

Found by

0x007, Bauchibred, bitsurfer, shogoki

Summary

Silently failing transfers can result in a partial or total loss of the users investment.

Vulnerability Detail

Some ERC20 Tokens do not revert on failure of the transfer function, but return a bool value instead. Some do not return any value. Therefore it is required to check if a value was returned, and if true, which value it is. This is not done on some places in these contracts.

The `DebtIssuanceModulev2`, which is required to issue or redeem tokens whenever there is a LeverageModule involved uses the `invokeTransfer` function of the `Invoke` library to transfer ERC20 Tokens from the `SetToken` to the user.

`invokeTransfer` is encoding the Calldata for the regular `transfer` function of ERC20 tokens and passes it together with the target (ERC20 token address) the SetTokens generic `invoke` function, whichin turn uses `functionCallWithValue` from the Openzeppelin Address Library. This method is bubbling up a possible revert of the call and returning the raw data.

The generic `invoke` is returning this raw data, however in `invokeTransfer` the return value of `invoke` is ignored and not used. As some ERC20 Tokens do not revert on a failed transfer, but instead return a `false` bool value, the stated behaviour can lead to silently failing transfers.

This is inside the `DebtIssuanceModulev2` used to:

1. Transfer The "debt" (borrowed) Tokens to the user at Issuance
2. Transfer back the main component Tokens (e.g. aTokens) to the user at Redemption

If such a Transfer silently fails, the funds will remain inside the setToken contract and the user has no chance to recover them.

In the issuance event the user receives the SetTokens but not the borrowed Tokens, which he has to repay when he wants to redeem the tokens. (Results in Loss of the "Debt")

In the redemption event the user repays his debt & bruns his Set Tokens, but never receives his original Tokens back. (Total Loss of investment)

Impact

Possible Loss of all/part of the Investment for the User

Code Snippet

Usage of invokeTransfer in`DebtIssuanceModulev2`:

https://github.com/sherlock-audit/2023-05-Index/blob/main/index-protocol/contracts/protocol/modules/v1/DebtIssuanceModuleV2.sol#L283

https://github.com/sherlock-audit/2023-05-Index/blob/main/index-protocol/contracts/protocol/modules/v1/DebtIssuanceModuleV2.sol#L315

`invokeTransfer` function ignores return value of `invoke`:

https://github.com/sherlock-audit/2023-05-Index/blob/main/index-protocol/contracts/protocol/lib/Invoke.sol#L66-L78

`invoke` uses `functionCallwithValue` and returns the raw return Data (which is ignored in this case):

https://github.com/sherlock-audit/2023-05-Index/blob/main/index-protocol/contracts/protocol/SetToken.sol#L197-L212

Tool used

Manual Review

Recommendation

Check for the existence and value of the returned data of the Transfer call. If there is a return value, it has to be true. This could be achieved by using Openzeppelins SafeERC20 library´s `safeTransfer`.

Discussion

Shogoki

Escalate for 10USDC This should be considered a valid finding. it backs up the existing escalation on #236

There exists a risk of loosing funds due to missing checks on the transfer calls, as described in the report.

sherlock-admin

> Escalate for 10USDC This should be considered a valid finding. it backs up the existing escalation on #236
>
> There exists a risk of loosing funds due to missing checks on the transfer calls, as described in the report.

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.

0xffff11

This should definitely be a duplicate of the issue that states that the protocol does not work with tokens like USDT. This is just a superset of that issue that has the same fix.

Shogoki

> This should definitely be a duplicate of the issue that states that the protocol does not work with tokens like USDT. This is just a superset of that issue that has the same fix.

No it should not. It should be a valid issue together with duplicates like: #155, #236 and #280

This issue is not about USDT. It is about Tokens that do not revert on failure, which can lead to a silently failed transfer.

hrishibhat

Result: Medium Has duplicates Considering this and its duplicates a separate issue as its core issue is different from #314

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- Shogoki: accepted

pblivin0x

Fix opened in https://github.com/IndexCoop/index-protocol/pull/28

IAm0x52

Fix looks good. Non-empty return data is now checked to accommodate tokens that return false instead of reverting

Issue M-4: no validation to ensure the arbitrum sequencer is down

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/262

Found by

0x007, Bauer, BugBusters, MohammedRizwan, Phantasmagoria, Saeedalipoor01988, ShadowForce, hildingr, jasonxiale, kutugu, rvierdiiev, sashik_eth

Summary

There is no validation to ensure sequencer is down

Vulnerability Detail

```
int256 rawCollateralPrice = strategy.collateralPriceOracle.latestAnswer();
       rebalanceInfo.collateralPrice = rawCollateralPrice.toUint256().mul(10 **
↪  strategy.collateralDecimalAdjustment);
       int256 rawBorrowPrice = strategy.borrowPriceOracle.latestAnswer();
       rebalanceInfo.borrowPrice = rawBorrowPrice.toUint256().mul(10 **
↪  strategy.borrowDecimalAdjustment);
```

Using Chainlink in L2 chains such as Arbitrum requires to check if the sequencer is down to avoid prices from looking like they are fresh although they are not.

The bug could be leveraged by malicious actors to take advantage of the sequencer downtime.

Impact

when sequencer is down, stale price is used for oracle and the borrow value and collateral value is calculated and the protocol can be forced to rebalance in a loss position

Code Snippet

https://github.com/IndexCoop/index-coop-smart-contracts/blob/317dfb677e9738fc990cf69d198358065e8cb595/contracts/adapters/AaveLeverageStrategyExtension.sol#L889-L907

Tool used

Manual Review

Recommendation

recommend to add checks to ensure the sequencer is not down.

Discussion

ckoopmann

Seems to be correct however I'm not sure regarding validity / severity since this is specific to L2 and not relevant for our current deployment strategy on Ethereum.

Issue M-5: Relying solely on oracle base slippage parameters can cause significant loss due to sandwich attacks

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/285

Found by

0x52

Summary

AaveLeverageStrategyExtension relies solely on oracle price data when determining the slippage parameter during a rebalance. This is problematic as chainlink oracles, especially mainnet, have upwards of 2% threshold before triggering a price update. If swapping between volatile assets, the errors will compound causing even bigger variation. These variations can be exploited via sandwich attacks.

Vulnerability Detail

AaveLeverageStrategyExtension.sol#L1147-L1152

```
function _calculateMinRepayUnits(uint256 _collateralRebalanceUnits, uint256
↪  _slippageTolerance, ActionInfo memory _actionInfo) internal pure returns
↪  (uint256) {
    return _collateralRebalanceUnits
        .preciseMul(_actionInfo.collateralPrice)
        .preciseDiv(_actionInfo.borrowPrice)
        .preciseMul(PreciseUnitMath.preciseUnit().sub(_slippageTolerance));
}
```

When determining the minimum return from the swap, _calculateMinRepayUnits directly uses oracle data to determine the final output. The differences between the true value and the oracle value can be systematically exploited via sandwich attacks. Given the leverage nature of the module, these losses can cause significant loss to the pool.

Impact

Purely oracle derived slippage parameters will lead to significant and unnecessary losses

Code Snippet

AaveLeverageStrategyExtension.sol#L1147-L1152

Tool used

Manual Review

Recommendation

The solution to this is straight forward. Allow keepers to specify their own slippage value. Instead of using an oracle slippage parameter, validate that the specified slippage value is within a margin of the oracle. This gives the best of both world. It allows for tighter and more reactive slippage controls while still preventing outright abuse in the event that the trusted keeper is compromised.

Discussion

ckoopmann

While raising a valid point regarding Chainlink price change trigger, this still seems the best way to set slippage.

Letting keepers set their own min/max amount is definitely not safe as they are not necessarily trusted. (Especially in the case of the "ripcord" function which can be called by anyone)

pblivin0x

The proposed solution to `allow keepers to specify their own slippage value` and `validate that the specified slippage value is within a margin of the oracle` looks good to me

ckoopmann

Ah yes, I missed the `validate that the specified slippage value is within a margin of the oracle` part of the suggestion, which does make sense and would probably slightly decrease slippage risk. However I'm not sure if it is worth the resulting changes in our keeper infrastructure and having a different keeper interface from other leverage modules.

Overall changing this to "confirmed" but unsure yet of wether we will actually fix it.

0xffff11

Great catch and fix seems reasonable. Valid medium

ckoopmann

After extensive deliberation we decided to not fix this, as the suggested changes don't seem to justify the effort and might in fact open new attack vectors / issues.

Given that we have had multiple years of experience with this setup in other leveraged tokens, without encountering issues the more conservative approach seems to keep it as is.

Issue M-6: Chainlink price feed is `deprecated`, not sufficiently validated and can return `stale` prices.

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/296

Found by

0x007, 0x8chars, 0xGoodess, 0xStalin, Bauchibred, Bauer, Brenzee, BugBusters, Cryptor, Diana, Madalad, MohammedRizwan, Ocean_Sky, Oxsadeeq, Phantasmagoria, Saeedalipoor01988, ShadowForce, erictee, jasonxiale, kn0t, kutugu, lil.eth, oxchryston, rvierdiiev, saidam017, sashik_eth, shogoki, volodya, warRoom, whitehat

Summary

The function `_createActionInfo()` uses Chainlink's deprecated latestAnswer function, this function also does not guarantee that the price returned by the Chainlink price feed is not stale and there is no additional checks to ensure that the return values are valid.

Vulnerability Detail

The internal function `_createActionInfo()` uses calls `strategy.collateralPriceOracle.latestAnswer()` and `strategy.borrowPriceOracle.latestAnswer()` that uses Chainlink's deprecated latestAnswer() to get the latest price. However, there is no check for if the return value is a stale data.

```
function _createActionInfo() internal view returns(ActionInfo memory) {
        ActionInfo memory rebalanceInfo;

        // Calculate prices from chainlink. Chainlink returns prices with 8
↪  decimal places, but we need 36 - underlyingDecimals decimal places.
        // This is so that when the underlying amount is multiplied by the
↪  received price, the collateral valuation is normalized to 36 decimals.
        // To perform this adjustment, we multiply by 10^(36 - 8 -
↪  underlyingDecimals)
        int256 rawCollateralPrice =
↪  strategy.collateralPriceOracle.latestAnswer();
        rebalanceInfo.collateralPrice = rawCollateralPrice.toUint256().mul(10 **
↪  strategy.collateralDecimalAdjustment);
        int256 rawBorrowPrice = strategy.borrowPriceOracle.latestAnswer();
        rebalanceInfo.borrowPrice = rawBorrowPrice.toUint256().mul(10 **
↪  strategy.borrowDecimalAdjustment);
// More Code....
```

```
}
```

Impact

The function `_createActionInfo()` is used to return important values used throughout the contract, the staleness of the chainlinklink return values will lead to wrong calculation of the collateral and borrow prices and other unexpected behavior.

Code Snippet

https://github.com/IndexCoop/index-coop-smart-contracts/blob/317dfb677e9738fc990cf69d198358065e8cb595/contracts/adapters/AaveLeverageStrategyExtension.sol#L889

Tool used

Manual Review

Recommendation

The `latestRoundData` function should be used instead of the deprecated `latestAnswer` function and add sufficient checks to ensure that the pricefeed is not stale.

```
(uint80 roundId, int256 assetChainlinkPriceInt, , uint256 updatedAt, uint80
↪  answeredInRound) = IPrice(_chainlinkFeed).latestRoundData();
            require(answeredInRound >= roundId, "price is stale");
            require(updatedAt > 0, "round is incomplete");
```

Discussion

0xffff11

Sponsor comments:

```
Good point to switch away from using the deprecated method, which we will look
↪  into.
However from this issue it is not clear how / if there is any actual
↪  vulnerability resulting from the use of this method.
--
Agree with @ckoopmann , the proposed fix of using latestRoundData() looks
↪  reasonable to me
```

```
--
I switched to confirmed / disagree with severity as this issue is factually
↪  correct and will result in us changing the code, but does not seem to have
↪  any real adverse consequences.
```

0xffff11

I do believe that this should remain as a medium. Not just for the impact stated by the watson, but also because Chainlink might simply not support it anymore in the future.

ckoopmann

Switched to using `latestRoundData` and adding a configurable maxPriceAge that is compared against the `updatedAt` value. Fixed in: https://github.com/IndexCoop/index-coop-smart-contracts/pull/142

IAm0x52

Oracle was changed to AAVEOracle, which also fixed this issue

Issue M-7: The protocol does not compatible with token such as USDT because of the Approval Face Protection

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/314

Found by

0x007, 0xStalin, Ruhum, ShadowForce, jasonxiale, kutugu, n33k, rvierdiiev, shogoki, volodya

Summary

The protocol does not compatible with token such as USDT because of the Approval Face Protection

Vulnerability Detail

the protocol is intended to interact with any ERC20 token and USDT is a common one

> Q: Which ERC20 tokens do you expect will interact with the smart contracts? The protocol expects to interact with any ERC20.
>
> Individual SetToken's should only interact with ERC20 chosen by the SetToken manager.

when doing the deleverage

https://github.com/IndexCoop/index-protocol/blob/86be7ee76d9a7e4f7e93acfc533216ebef791c89/contracts/protocol/modules/v1/AaveV3LeverageModule.sol#L313

first, we construct the deleverInfo

```
ActionInfo memory deleverInfo = _createAndValidateActionInfo(
        _setToken,
        _collateralAsset,
        _repayAsset,
        _redeemQuantityUnits,
        _minRepayQuantityUnits,
        _tradeAdapterName,
        false
    );
```

then we withdraw from the lending pool, execute trade and repay the borrow token

```
_withdraw(deleverInfo.setToken, deleverInfo.lendingPool, _collateralAsset,
↪  deleverInfo.notionalSendQuantity);

        uint256 postTradeReceiveQuantity = _executeTrade(deleverInfo,
↪  _collateralAsset, _repayAsset, _tradeData);

        uint256 protocolFee = _accrueProtocolFee(_setToken, _repayAsset,
↪  postTradeReceiveQuantity);

        uint256 repayQuantity = postTradeReceiveQuantity.sub(protocolFee);

        _repayBorrow(deleverInfo.setToken, deleverInfo.lendingPool, _repayAsset,
↪  repayQuantity);
```

this is calling _repayBorrow

```
/**
 * @dev Invoke repay from SetToken using AaveV2 library. Burns DebtTokens for
↪  SetToken.
 */
function _repayBorrow(ISetToken _setToken, ILendingPool _lendingPool, IERC20
↪  _asset, uint256 _notionalQuantity) internal {
    _setToken.invokeApprove(address(_asset), address(_lendingPool),
↪  _notionalQuantity);
    _setToken.invokeRepay(_lendingPool, address(_asset), _notionalQuantity,
↪  BORROW_RATE_MODE);
}
```

the trade received (quantity - the protocol fee) is used to repay the debt

but the required debt to be required is the (borrowed amount + the interest rate)

suppose the only debt that needs to be repayed is 1000 USDT

trade received (quantity - the protocol) fee is 20000 USDT

only 1000 USDT is used to repay the debt

because when repaying, the paybackAmount is only the debt amount

https://github.com/aave/aave-v3-core/blob/29ff9b9f89af7cd8255231bc5faf26c3ce0fb7ce/contracts/protocol/libraries/logic/BorrowLogic.sol#L204

```
uint256 paybackAmount = params.interestRateMode ==
↪  DataTypes.InterestRateMode.STABLE
  ? stableDebt
  : variableDebt;
```

then when burning the variable debt token

https://github.com/aave/aave-v3-core/blob/29ff9b9f89af7cd8255231bc5faf26c3ce0fb7ce/contracts/protocol/libraries/logic/BorrowLogic.sol#L224

```
reserveCache.nextScaledVariableDebt = IVariableDebtToken(
    reserveCache.variableDebtTokenAddress
  ).burn(params.onBehalfOf, paybackAmount, reserveCache.nextVariableBorrowIndex);
```

only the "payback amount", which is 1000 USDT is transferred to pay the debt,

the excessive leftover amount is (20000 USDT - 1000 USDT) = 19000 USDT

but if we lookback into the repayBack function

```
/**
 * @dev Invoke repay from SetToken using AaveV2 library. Burns DebtTokens for
 ↪  SetToken.
 */
function _repayBorrow(ISetToken _setToken, ILendingPool _lendingPool, IERC20
 ↪  _asset, uint256 _notionalQuantity) internal {
    _setToken.invokeApprove(address(_asset), address(_lendingPool),
 ↪  _notionalQuantity);
    _setToken.invokeRepay(_lendingPool, address(_asset), _notionalQuantity,
 ↪  BORROW_RATE_MODE);
}
```

the approved amount is 20000 USDT, but only 1000 USDT approval limit is used, we have 19000 USDT approval limit left

according to

https://github.com/d-xo/weird-erc20#approval-race-protections

> Some tokens (e.g. OpenZeppelin) will revert if trying to approve the zero address to spend tokens (i.e. a call to approve(address(0), amt)).
>
> Integrators may need to add special cases to handle this logic if working with such a token.

USDT is such token that subject to approval race condition, without approving 0 first, the second approve after first repay will revert

Impact

second and following repay borrow will revert if the ERC20 token is subject to approval race condition

Code Snippet

https://github.com/IndexCoop/index-protocol/blob/86be7ee76d9a7e4f7e93acfc533216ebef791c89/contracts/protocol/modules/v1/AaveV3LeverageModule.sol#L313

Tool used

Manual Review

Recommendation

Approval 0 first

Discussion

pblivin0x

Looks reasonable to add a preceding zero approval here
`_setToken.invokeApprove(address(_asset), address(_lendingPool), 0);`

0xffff11

Valid medium

ckoopmann

Still having some final discussion internally over wether to fix this or explicitly list USDT as an incompatible token.

ckoopmann

Fixed in: https://github.com/IndexCoop/index-protocol/pull/22

Specifically in the following commit by preceding any approve transaction with a 0 approval: https://github.com/IndexCoop/index-protocol/pull/22/commits/dae4d65ae8a95f9044b4c2edccaa394dbff451f3

bizzyvinci

@ckoopmann some of the duplicates such as #39 and #129 are about the AmmModule

IAm0x52

As mentioned by @bizzyvinci this same change should be made to the AMM module as well

IAm0x52

Fix for AMM module here

IAm0x52

Fixes look good and now always approve to 0 first

Issue M-8: Operator is blocked when sequencer is down on Arbitrum

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/321

Found by

hildingr

Summary

When the sequencer is down on Arbitrum state changes can still happen on L2 by passing them from L1 through the Delayed Inbox.

Users can still interact with the Index protocol but due to how Arbitrum address aliasing functions the operator will be blocked from calling onlyOperator().

Vulnerability Detail

The `msg.sender` of a transaction from the Delayed Inbox is aliased:

```
L2_Alias = L1_Contract_Address + 0x1111000000000000000000000000000000001111
```

All functions with the `onlyOperator()` modifier are therefore blocked when the sequencer is down.

The issue exists for all modifiers that are only callable by specific EOAs. But the operator of the Aave3LeverageStrategyExtension is the main security risk.

Impact

The operator has roles that are vital for the safety of the protocol. Re-balancing and issuing/redeeming can still be done when the sequencer is down it is therefore important that the operator call the necessary functions to operate the protocol when the sequencer is down.

`disengage()` is an important safety function that the operator should always have access especially when the protocol is still in accessible to other users. Changing methodology and adding/removing exchanges are also important for the safety of the protocol.

Code Snippet

https://github.com/sherlock-audit/2023-05-Index/blob/3190057afd3085143a31746d65045a0d1bacc78c/index-coop-smart-contracts/contracts/manager/BaseManagerV2.sol#L113-L116

Tool used

Manual Review

Recommendation

Change the `onlyOperator()` to check if the address is the aliased address of the operator.

Discussion

hildingr

Escalate for 10 USDC

This is not a duplicate off #262 and should be a separate issue.

This is not about a check if a sequencer is down but rather a peculiarity on how Arbitrum aliases addresses. On the other L2s the manager is still able to reach all functions when the sequencer is down since the aliasing is not done on EOA's initiating a L1->L2 call.

On Arbitrum this is not the case and the manager/operators are completely blocked from controlling the protocol when the sequencer is down.

As it stands on the other L2's the manager/operators can still govern over the protocol and use all the available safety features if the sequencer is down. As it stands this is not possible on Arbitrum.

Low probability event where it would be crucial for operator/manager to have access:

Sequencer is down for a prolonged time this could be some kind of attack or a technical issue, couple this with volatility in the market either due to the issues with the sequencer or unrelated. The governance should be able to change the safe parameters of position during such an event.

The recommended changes in the duplicates actually makes this worse in some cases since repaying debt is completely blocked when the sequencer is down. This is not the case for normal AAVE users which always have the ability to repay loans, this is an important safety feature guaranteed by AAVE.

This can be taken further if the Index Team wishes to have the same safety level as a native AAVE user.

New functionality can be added to allow the operator to repay debt and de-leverage when the sequencer is down. This is a safety feature available to all AAVE users, AAVE users are never blocked from repaying debt but only from taking out additional loans when the sequencer is down.

This can be done by allowing a new operator L1Operator to access a new rebalancing feature, this L1Operator is a L1 smart-contract that uses L1 oracle data to initiate a L2 repayment of debt and rebalance when the sequencer is down.

sherlock-admin

> Escalate for 10 USDC
>
> This is not a duplicate off #262 and should be a separate issue.
>
> This is not about a check if a sequencer is down but rather a peculiarity on how Arbitrum aliases addresses. On the other L2s the manager is still able to reach all functions when the sequencer is down since the aliasing is not done on EOA's initiating a L1->L2 call.
>
> On Arbitrum this is not the case and the manager/operators are completely blocked from controlling the protocol when the sequencer is down.
>
> As it stands on the other L2's the manager/operators can still govern over the protocol and use all the available safety features if the sequencer is down. As it stands this is not possible on Arbitrum.
>
> Low probability event where it would be crucial for operator/manager to have access:
>
> Sequencer is down for a prolonged time this could be some kind of attack or a technical issue, couple this with volatility in the market either due to the issues with the sequencer or unrelated. The governance should be able to change the safe parameters of position during such an event.
>
> The recommended changes in the duplicates actually makes this worse in some cases since repaying debt is completely blocked when the sequencer is down. This is not the case for normal AAVE users which always have the ability to repay loans, this is an important safety feature guaranteed by AAVE.
>
> This can be taken further if the Index Team wishes to have the same safety level as a native AAVE user.
>
> New functionality can be added to allow the operator to repay debt and de-leverage when the sequencer is down. This is a safety feature available to all AAVE users, AAVE users are never blocked from repaying debt but only from taking out additional loans when the sequencer is down.
>
> This can be done by allowing a new operator L1Operator to access a new rebalancing feature, this L1Operator is a L1 smart-contract that uses L1 oracle data to initiate a L2 repayment of debt and rebalance when the sequencer is down.

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.

0xffff11

Thanks. I think the changes of this happening are slim to non. Sequencer has to be down and at the same time the operator rebalance. Also, I the operator is a multisig, not an EOA, unsure if that makes any difference, but still, I do not see the medium here. @IAm0x52 thoughts?

IAm0x52

Due to the ripcord function always being available to prevent set token liquidation, low seems more appropriate to me.

0xffff11

Thanks for the second opinion. As said above, I agree with low

hrishibhat

Agree with the Lead Judge and Watson on this being low @hildingr

hildingr

I will have to disagree here. The ripcord can not be trusted when the sequencer is down. The price could move in either direction while the Oracle price is stale.

Imagine if the price is moving in a direction that would allow a ripcord pull, the ripcord can not be pulled since the oracle has the stale price.

On optimism and polygon the operator can call disengage() and stop the protocol from going above the max liquidation ratio. On Arbitrum this is not possible and the position can go way beyond the max LTV without being able to delever. The position could even get liquidated before ripcord or disengage can be called. If 0.95 < HF < 1 the position can be instantly liquidated when the sequencer comes back up since no grace period is given to heavily undercollateralized positions.

ckoopmann

I don't have a strong opinion on wether this is low or medium so will leave it up to the lead watson / judge to decide.

Since we are not planning to deploy on arbitrum for now we will not act on this issue for now, but will review should we deploy this there in the future.

hrishibhat

@hildingr Additional comment from the Lead watson.

> The key consideration here is that the main intention here is to keep the set token from becoming liquidated. So it doesn't really matter what the oracle is at whether it's stale or completely wrong. As long as the oracle being used by the set token matches AAVE then that's all that matters. So as I've stated, ripcord will always protect the set token so this is low

hildingr

> @hildingr Additional comment from the Lead watson.
>
> > The key consideration here is that the main intention here is to keep the set token from becoming liquidated. So it doesn't really matter what the oracle is at whether it's stale or completely wrong. As long as the oracle being used by the set token matches AAVE then that's all that matters. So as I've stated, ripcord will always protect the set token so this is low

I disagree, the ripcord will not always protect the position. I will give a concrete example:

Stage 1, the sequencer is up: HF = 1.5

Stage 2, the sequencer is down: HF is -> 1. ripcord can not be pulled due to incorrect internal LTV.

Stage 3, sequencer still down: HF < 0.95.

Stage 4, the moment the sequencer comes up: Race condition between instant liquidation and pulling the ripcord. This is because HF < 0.95 and no grace period is given by the AAVE sentinel.

The true LTV always matters even if AAVE and Index use the same stale oracle. We can look at it as a discontinuity in the LTV, when the seq is down it is "unknown" to both protocols but in the instant the sequencer comes back the LTV jumps to the true value. The true LTV can be in very dangerous territory, possibly high enough for instant liquidation.

IAm0x52

> Stage 4, the moment the sequencer comes up: Race condition between instant liquidation and pulling the ripcord. This is because HF < 0.95 and no grace period is given by the AAVE sentinel.

This is correct. it would create race conditions under these circumstances

hrishibhat

Result: Medium Unique Considering this a valid medium based on the above comments

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- hildingr: accepted

Issue M-9: Oracle Price miss matched when E-mode uses single oracle

Source: https://github.com/sherlock-audit/2023-05-Index-judging/issues/323

Found by

0x52, hildingr

Summary

AAVE3 can turn on single oracle use on any E-mode category. When that is done collateral and the borrowed assets will be valued based on a single oracle price. When this is done the prices used in AaveLeverageStrategyExtension can differ from those used internally in AAVE3.

This can lead to an increased risk of liquidation and failures to re-balance properly.

Vulnerability Detail

There is currently no accounting for single oracle use in the AaveLeverageStragyExtension, if AAVE3 turns it on the extension will simply continue using its current oracles without accounting for the different prices.

When re-balancing the following code calculate the `netBorrowLimit`/`netRepayLimit`:

```
if (_isLever) {
    uint256 netBorrowLimit = _actionInfo.collateralValue
        .preciseMul(maxLtvRaw.mul(10 ** 14))
        .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLevera
↪   gePercentage));

    return netBorrowLimit
        .sub(_actionInfo.borrowValue)
        .preciseDiv(_actionInfo.collateralPrice);
} else {
    uint256 netRepayLimit = _actionInfo.collateralValue
        .preciseMul(liquidationThresholdRaw.mul(10 ** 14))
        .preciseMul(PreciseUnitMath.preciseUnit().sub(execution.unutilizedLevera
↪   gePercentage));

    return _actionInfo.collateralBalance
        .preciseMul(netRepayLimit.sub(_actionInfo.borrowValue))
        .preciseDiv(netRepayLimit);
```

The `_actionInfo.collateralValue` and `_adminInfo.borrowValue` are `_getAndValidateLeverageInfo()` where they are both retrieved based on the current set chainlink oracle.

When E-mode uses a single oracle price a de-pegging of one of the assets will lead to incorrect values of `netBorrowLimit` and `netRepayLimit` depending on which asset is de-pegging.

`collateralValue` or `borrowValue` can be either larger or smaller than how they are valued internally in AAVE3.

Impact

When Levering

If `collateralValue` is to valued higher than internally in AAVE3 OR If `borrowValue` is to valued lower than internally in AAVE3:

The `netBorrowLimit` is larger than it should be we are essentially going to overriding `execute.unutilizedLeveragePercentage` and attempting to borrow more than we should.

If `collateralValue` is valued lower than internally in AAVE3 OR If `borrowValue` is to valued higher than internally in AAVE3:

The `netBorrowLimit` is smaller than it should be, we are not borrowing as much as we should. Levering up takes longer.

When Delevering

If `collateralValue` is to valued higher than internally in AAVE3 OR If `borrowValue` is to valued lower than internally in AAVE3:

We will withdraw more collateral and repay more than specified by `execution.unutilizedLeveragePercentage`.

If `collateralValue` is valued lower than internally in AAVE3 OR If `borrowValue` is to valued higher than internally in AAVE3:

We withdraw less and repay less debt than we should. This means that both `ripcord()` and `disengage()` are not functioning as they, they will not delever as fast they should. We can look at it as `execution.unutilizedLeveragePercentage` not being throttled.

The above consequences show that important functionality is not working as expected. "overriding" `execution.unutilizedLeveragePercentage` is a serious safety concern.

Code Snippet

https://github.com/sherlock-audit/2023-05-Index/blob/3190057afd3085143a3174

6d65045a0d1bacc78c/index-coop-smart-contracts/contracts/adapters/AaveLeverageStrategyExtension.sol#L1095-L1119

Tool used

Manual Review

Recommendation

Aave3LeverageStrategyExtension should take single oracle usage into account. `_calcualteMaxBorrowCollateral` should check if there is a discrepancy and adjust such that the `execute.unutilizedLeveragePercentage` safety parameter is honored.

Discussion

sherlock-admin

> Escalate for 10 USDC
>
> This issue is similar to #284 but is much more likely to happen in practice. If AAVE turns on single oracle for a category the oracles will be miss-configured this can lead to liquidation and other issues that I have outlined.
>
> This is more likely to happen than #284 since single oracle is a feature of E-mode that can be turned on at any time for any E-mode category.
>
> I want to escalate this to a valid Medium. I recommend that the index team implements functionality that take into account single oracle use which can be turned on at any point.

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

hildingr

Escalate for 10 USDC

This issue is similar to #284 but is much more likely to happen in practice. If AAVE turns on single oracle for a category the oracles will be miss-configured this can lead to liquidation and other issues that I have outlined.

This is more likely to happen than #284 since single oracle is a feature of E-mode that can be turned on at any time for any E-mode category.

I want to escalate this to a valid Medium. I recommend that the index team implements functionality that take into account single oracle use which can be turned on at any point.

sherlock-admin

> Escalate for 10 USDC
>
> This issue is similar to #284 but is much more likely to happen in practice. If AAVE turns on single oracle for a category the oracles will be miss-configured this can lead to liquidation and other issues that I have outlined.
>
> This is more likely to happen than #284 since single oracle is a feature of E-mode that can be turned on at any time for any E-mode category.
>
> I want to escalate this to a valid Medium. I recommend that the index team implements functionality that take into account single oracle use which can be turned on at any point.

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.

0xffff11

Even using a single oracle in E-mode. Both implementations would still use chainlink right?

hildingr

Yes, but the issue is that when single oracle is turned on one of the assets will be guaranteed to use a different oracle feed in the Index Protocol compared to AAVE. AAVE and Index will value the asset differently which means that the calculations in _calculateMaxBorrowCollateral will be incorrect.

Example: Initial state: Both index and AAVE use feed 1 for X and feed 2 for Y. Assume E-mode is activated.

Single Oracle turned on: AAVE now uses feed 1 as the single oracle feed for the E-mode category, both X and Y will use feed 1.

Now Y follows feed 1 on AAVE and feed 2 on Index. The calculations in _calculateChunkRebalanceNotional will as a result be incorrect. The magnitude of the error depends on how much and which direction the oracle feeds diverge in.

0xffff11

Thanks! Seems reasonable to me and I could see the medium here. Thoughts? @IAm0x52

IAm0x52

Same underlying divergence issue as #284 though this one definitely has a better example of how that would happen. My take is that these should be duped then presented to Index team as well as Sherlock for a final severity discussion.

In the unlikely case that something like this does occur (oracles are different AND diverge) the consequences are tremendous. Ultimately the severity would come down to the likelihood.

hildingr

I disagree that this and #284 are duplicates. The issue I identify can happen during the expected behavior of AAVE V3 due to new functionality added in V3 that has not been accounted for in Index.

The likelihood is much larger since an Oracle feed mismatch is guaranteed to happen if Single Oracle is turned on. Single Oracle is a feature of AAVE V3, I therefore believe that it should not be considered an unlikely scenario to enter a state with mismatched feeds.

ckoopmann

Even though it could be seen as an "external" / "admin" feature, this issue raises a valid concern. Using the AaveOracle instead might acutally be the better choice and we will review internally if we want to make that fix / change.

Therefore I will mark this issue as "Sponsor confirmed".

While this issue seems to go into more detail of how oracle mismatch can happen, the core issue seems to be the same as 284. Therefore this response also applies to both issues.

hildingr

> Even though it could be seen as an "external" / "admin" feature, this issue raises a valid concern. Using the AaveOracle instead might acutally be the better choice and we will review internally if we want to make that fix / change.
>
> Therefore I will mark this issue as "Sponsor confirmed".
>
> While this issue seems to go into more detail of how oracle mismatch can happen, the core issue seems to be the same as 284. Therefore this response also applies to both issues.

I agree that #284 is an admin/external since it is a potential issue that arise due to admin changing a parameter that is "trusted".

What I point to is an issue that arise due to index protocol not handling a feature added to AAVE V3. AAVE V3 is built to use E-modes where single oracle is turned on and off, any such event would put index at risk.

I think it is incorrect to label an issue an "admin" issue if it is a consequence of incompatibility with a feature of a protocol being used as expected.

hrishibhat

Result: Medium Has duplicates After further discussions internally and the protocol. Considering this issue a valid medium as changing parameters in the external protocol affects index protocol as shown in the issue. Also considering issue #284 as a duplicate the underlying issue originates from the external protocol due to a change in parameters/functionality by the external admin, affecting oracle price. More context on the external admin trust assumptions is updated in the judging guide.
https://docs.sherlock.xyz/audits/judging/judging#some-standards-observed

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- hildingr: accepted

ckoopmann

Fixed in: https://github.com/IndexCoop/index-coop-smart-contracts/pull/142/commits/bed0e348e05e17e3c0e11be4cf6c22bf900abd36

IAm0x52

Fix looks good. Now using AAVE oracle so that oracles are always aligned