SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Index
Prepared by:	Sherlock
Lead Security Expert:	0x52
Dates Audited:	July 10 - July 17, 2023
Prepared on:	August 15, 2023

Introduction

We're building secure, accessible and simple to use DeFi products that make it easy for everyone from retail investors to institutions to gain exposure to the most important themes in DeFi.

Scope

Repository: IndexCoop/index-protocol

Branch: master

Commit: 663e64efaa95df2247afa8926d4cfb42948f54fe

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
7	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

auditsea	0x007	Arabadzhiev
Brenzee	0x52	ast3ros
Yuki	dany.armstrong90	qandisa

Issue M-1: SetToken can't be unlocked early.

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/38

Found by

0x52, Yuki, auditsea, qandisa

Summary

SetToken can't be unlocked early

Vulnerability Detail

The function unlock() is used to unlock the setToken after rebalancing, as how it is right now there are two ways to unlock the setToken.

- can be unlocked once the rebalance duration has elapsed
- can be unlocked early if all targets are met, there is excess or at-target quote asset, and raiseTargetPercentage is zero

```
function unlock(ISetToken _setToken) external {
    bool isRebalanceDurationElapsed = _isRebalanceDurationElapsed(_setToken);
    bool canUnlockEarly = _canUnlockEarly(_setToken);

    // Ensure that either the rebalance duration has elapsed or the conditions
↪   for early unlock are met
    require(isRebalanceDurationElapsed || canUnlockEarly, "Cannot unlock early
↪   unless all targets are met and raiseTargetPercentage is zero");

    // If unlocking early, update the state
    if (canUnlockEarly) {
        delete rebalanceInfo[_setToken].rebalanceDuration;
        emit LockedRebalanceEndedEarly(_setToken);
    }

    // Unlock the SetToken
    _setToken.unlock();
}
```

```
function _canUnlockEarly(ISetToken _setToken) internal view returns (bool) {
    RebalanceInfo storage rebalance = rebalanceInfo[_setToken];
    return _allTargetsMet(_setToken) && _isQuoteAssetExcessOrAtTarget(_setToken)
↪   && rebalance.raiseTargetPercentage == 0;
}
```

The main problem occurs as the value of raiseTargetPercentage isn't reset after rebalancing. The other thing is that the function setRaiseTargetPercentage can't be used to fix this issue as it doesn't allow giving raiseTargetPercentage a zero value.

A setToken can use the AuctionModule to rebalance multiple times, duo to the fact that raiseTargetPercentage value isn't reset after every rebalancing. Once changed with the help of the function setRaiseTargetPercentage this value will only be non zero for every next rebalancing. A setToken can be unlocked early only if all other requirements are met and the raiseTargetPercentage equals zero.

This problem prevents for a setToken to be unlocked early on the next rebalances, once the value of the variable raiseTargetPercentage is set to non zero.

On every rebalance a manager should be able to keep the value of raiseTargetPercentage to zero (so the setToken can be unlocked early), or increase it at any time with the function setRaiseTargetPercentage.

```
function setRaiseTargetPercentage(
    ISetToken _setToken,
    uint256 _raiseTargetPercentage
)
    external
    onlyManagerAndValidSet(_setToken)
{
    // Ensure the raise target percentage is greater than 0
    require(_raiseTargetPercentage > 0, "Target percentage must be greater than
 ↪  0");

    // Update the raise target percentage in the RebalanceInfo struct
    rebalanceInfo[_setToken].raiseTargetPercentage = _raiseTargetPercentage;

    // Emit an event to log the updated raise target percentage
    emit RaiseTargetPercentageUpdated(_setToken, _raiseTargetPercentage);
}
```

Impact

Once the value of raiseTargetPercentage is set to non zero, every next rebalancing of the setToken won't be eligible for unlocking early. As the value of raiseTargetPercentage isn't reset after every rebalance and neither the manager can set it back to zero with the function setRaiseTargetPercentage().

Code Snippet

https://github.com/sherlock-audit/2023-06-Index/blob/main/index-protocol/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L389

Tool used

Manual Review

Recommendation

Recommend to reset the value raiseTargetPercentage after every rebalancing.

```
    function unlock(ISetToken _setToken) external {
        bool isRebalanceDurationElapsed = _isRebalanceDurationElapsed(_setToken);
        bool canUnlockEarly = _canUnlockEarly(_setToken);

        // Ensure that either the rebalance duration has elapsed or the
 ↪  conditions for early unlock are met
        require(isRebalanceDurationElapsed || canUnlockEarly, "Cannot unlock
 ↪  early unless all targets are met and raiseTargetPercentage is zero");

        // If unlocking early, update the state
        if (canUnlockEarly) {
            delete rebalanceInfo[_setToken].rebalanceDuration;
            emit LockedRebalanceEndedEarly(_setToken);
        }

+       rebalanceInfo[_setToken].raiseTargetPercentage = 0;

        // Unlock the SetToken
        _setToken.unlock();
    }
```

Discussion

pblivin0x

The risk here is a stale `raiseTargetPercentage` can lead to a SetToken that cannot be unlocked early?

Debating if we should include this or not.

FlattestWhite

Should raiseTargetPercentage be part of the check for unlocking early? I'm thinking it doesn't since that's only used when needing to `raiseTargetAssets` because we met all our targets and we have leftover WETH.

pblivin0x

since we can't set the raiseAssetTarget to zero, this is a valid issue. will fix

snake-poison

> since we can't set the raiseAssetTarget to zero, this is a valid issue. will fix

Yea I agree with the assessment and looks like the fix is more or less clear.

pblivin0x

The remediation for this issue is open for review here https://github.com/IndexCoop/index-protocol/pull/25

The changes are to

1) Allow the `raiseTargetPercentage` to be set to 0: https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L420-L439
2) Reset the `raiseTargetPercentage` on every `unlock()` call https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L414

IAm0x52

Fix looks good. Manger can now set `raiseTargetPercentage` to 0 directly. Additionally it will also be set to zero when the auction unlocks.

Issue M-2: price is calculated wrongly in BoundedStepwiseExponentialPriceAdapter

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/39

Found by

0x007, 0x52, Brenzee, auditsea, dany.armstrong90

Summary

The BoundedStepwiseExponentialPriceAdapter contract is trying to implement price change as `scalingFactor * (e^x - 1)` but the code implements `scalingFactor * e^x - 1`. Since there are no brackets, multiplication would be executed before subtraction. And this has been confirmed with one of the team members.

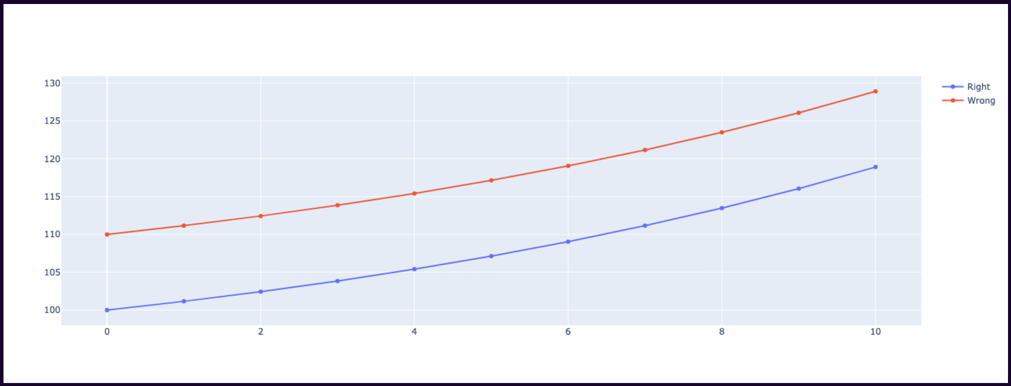
Vulnerability Detail

The getPrice code has been simplified as the following when boundary/edge cases are ignored

```
(
    uint256 initialPrice,
    uint256 scalingFactor,
    uint256 timeCoefficient,
    uint256 bucketSize,
    bool isDecreasing,
    uint256 maxPrice,
    uint256 minPrice
) = getDecodedData(_priceAdapterConfigData);

uint256 timeBucket = _timeElapsed / bucketSize;

int256 expArgument = int256(timeCoefficient * timeBucket);

uint256 expExpression = uint256(FixedPointMathLib.expWad(expArgument));

uint256 priceChange = scalingFactor * expExpression - WAD;
```

When timeBucket is 0, we want priceChange to be 0, so that the returned price would be the initial price. Since `e^0 = 1`, we need to subtract 1 (in WAD) from the `expExpression`.

However, with the incorrect implementation, the returned price would be different than real price by a value equal to `scalingFactor - 1`. The image below shows the difference between the right and wrong formula when initialPrice is 100 and scalingFactor is 11. The right formula starts at 100 while the wrong one starts at 110=100+11-1



Impact

Incorrect price is returned from BoundedStepwiseExponentialPriceAdapter and that will have devastating effects on rebalance.

Code Snippet

https://github.com/sherlock-audit/2023-06-Index/blob/main/index-protocol/contracts/protocol/integration/auction-price/BoundedStepwiseExponentialPriceAdapter.sol#L73

Tool used

Manual Review

Recommendation

Change the following line

```
- uint256 priceChange = scalingFactor * expExpression - WAD;
+ uint256 priceChange = scalingFactor * (expExpression - WAD);
```

Discussion

pblivin0x

Confirmed

IAm0x52

Escalate

This should be medium not high. While it is true that the calculation will be wrong for scaling factors other than 1, it heavily depends on the configuration of auction settings as to whether this sells assets at a bad price and causes a loss to the set token.

sherlock-admin2

> Escalate
>
> This should be medium not high. While it is true that the calculation will be wrong for scaling factors other than 1, it heavily depends on the configuration of auction settings as to whether this sells assets at a bad price and causes a loss to the set token.

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

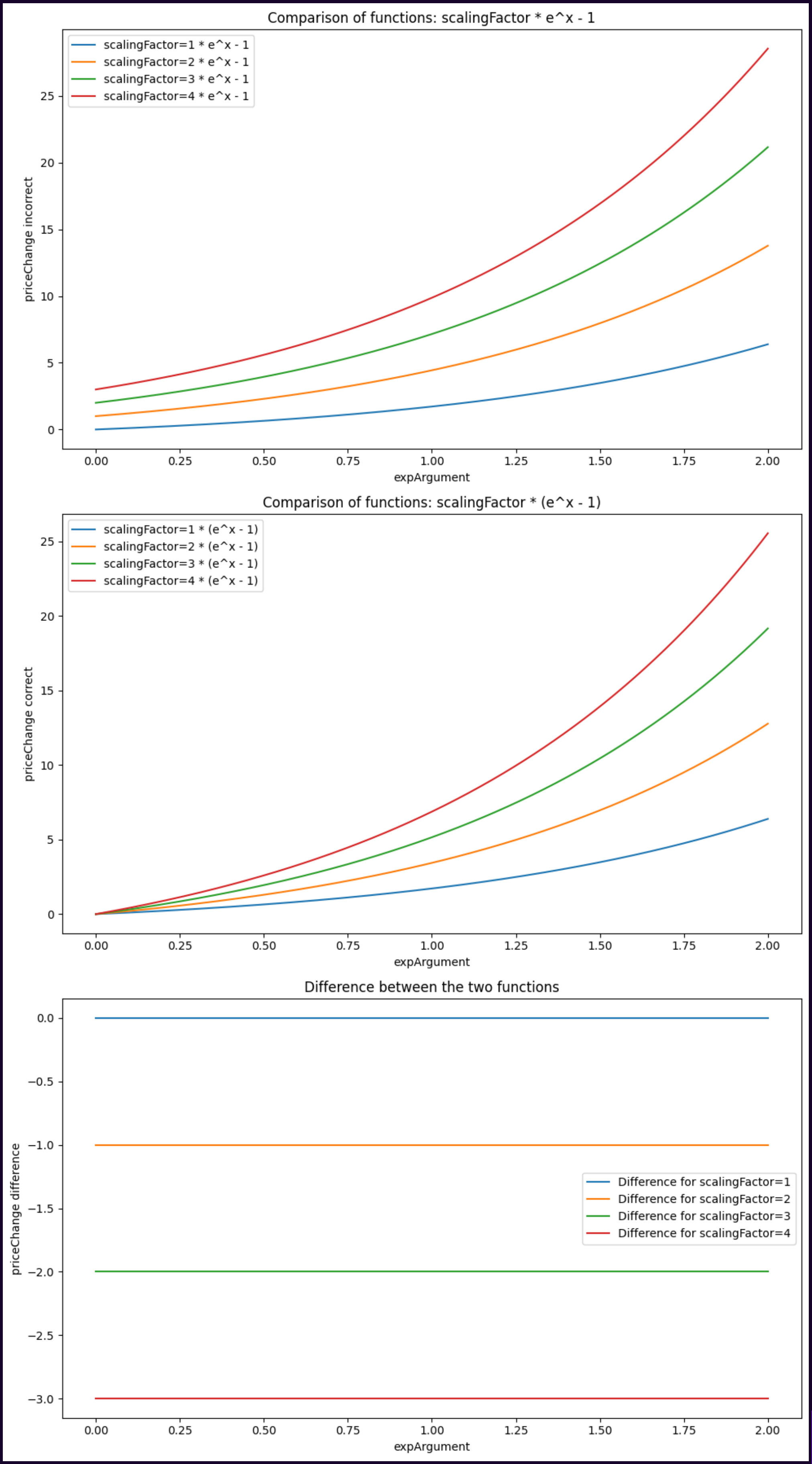
You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

pblivin0x

> Escalate
>
> This should be medium not high. While it is true that the calculation will be wrong for scaling factors other than 1, it heavily depends on the configuration of auction settings as to whether this sells assets at a bad price and causes a loss to the set token.

Agree here this should be a medium not a high

1) manager's are expected to preview full auction price curves ahead of a `startRebalance` call by calling the pure function `getPrice` with input `_timeElapsed` in the range `[0, _duration]`
2) the difference is not catastrophic, it is a shift that stays constant throughout the auction https://colab.research.google.com/drive/1ZkXs2MuTJFaWUU611KoINXQ52zh9k3VM?usp=sharing


Comparison of functions: scalingFactor * e^x - 1
scalingFactor=1 * e^x - 1
scalingFactor=2 * e^x - 1
scalingFactor=3 * e^x - 1
scalingFactor=4 * e^x - 1
priceChange incorrect
expArgument
Comparison of functions: scalingFactor * (e^x - 1)
scalingFactor=1 * (e^x - 1)
scalingFactor=2 * (e^x - 1)
scalingFactor=3 * (e^x - 1)
scalingFactor=4 * (e^x - 1)
priceChange correct
expArgument
Difference between the two functions
Difference for scalingFactor=1
Difference for scalingFactor=2
Difference for scalingFactor=3
Difference for scalingFactor=4
priceChange difference
expArgument

SHERLOCK

bizzyvinci

> manager's are expected to preview full auction price curves ahead of a startRebalance call by calling the pure function getPrice with input _timeElapsed in the range [0, _duration]

getPrice only takes in one _timeElapsed (not array), so managers can't view full curve. Also, the curve is directly affected by `timeBucket`, rather than `_timeElapsed`. Different `_timeElapsed` could give the same price depending on bucketSize. There's hardly a simulation or plotting tool for solidity, so it might be done in other languages like we're doing right now.

> the difference is not catastrophic

It could be very catastrophic mainly because it affects priceChange (which would then be added or subtracted from price). WE DON'T KNOW WHAT INITIAL PRICE IS. It could be `10,000`, it could `20`, it could be `0.1`, it could be `1e-12`.

POC

- A manager wants to switch stablecoins. He wants to buy DAI and Sell USDT at a price of minimum and initial price of `1` and increasing to `1.05`.
- Technically, the price is not 1, but rather `1e-12` because of decimals `1e6/1e18` (1e6 in WAD)

With the right formula

- If he uses the scalingFactor of `2`, the priceChange at `t0` would be 0
- Therefore bidder would have to pay 1e18 DAI for 1e6USDT.

With the wrong formula

- If he uses the scalingFactor of `2`, the priceChange at `t0` would `1` (1e18 in WAD) instead of `0`
- Therefore the price would increase by a magnitude 1e12.
- Therefore, bidder would pay approximately 1e18 DAI for 1e6 * 1e12 USDT.
- Attacker could take 1e12 (1 trillion USDT) with one DAI in a flash.
- Or if there's not enough liquidity e.g if there's only 1m USDT, then he'll pay `1e-12` DAI. That's less than a penny.

bizzyvinci

My bad, I agree with Med cause the catastrophe is bounded by min and max value.

pblivin0x

> My bad, I agree with Med cause the catastrophe is bounded by min and max value.

I think we all agree this should be de-escalated to a Medium

hrishibhat

Result: Medium Has duplicates Considering this a valid medium based on the above comments.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- IAm0x52: accepted

pblivin0x

The remediation for this issue is open for review here https://github.com/IndexCoop/index-protocol/pull/25

The fix to the formula is here: https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/integration/auction-price/BoundedStepwiseExponentialPriceAdapter.sol#L73

IAm0x52

Fix looks good. `scalingFactor` is now applied via `wadMul` which fixes this order of operation issue.

Issue M-3: No check for sequencer uptime can lead to dutch auctions executing at bad prices

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/40

Found by

0x52

Summary

When purchasing from dutch auctions on L2s there is no considering of sequencer uptime. When the sequencer is down, all transactions must originate from the L1. The issue with this is that these transactions use an aliased address. Since the set token contracts don't implement any way for these aliased addressed to interact with the protocol, no transactions can be processed during this time even with force L1 inclusion. If the sequencer goes offline during the the auction period then the auction will continue to decrease in price while the sequencer is offline. Once the sequencer comes back online, users will be able to buy tokens from these auctions at prices much lower than market price.

Vulnerability Detail

See summary.

Impact

Auction will sell/buy assets at prices much lower/higher than market price leading to large losses for the set token

Code Snippet

AuctionRebalanceModuleV1.sol#L772-L836

Tool used

Manual Review

Recommendation

Check sequencer uptime and invalidate the auction if the sequencer was ever down during the auction period

Discussion

pblivin0x

What exactly is the remediation here? To check an external uptime feed https://docs.chain.link/data-feeds/l2-sequencer-feeds ?

Not sure if we will fix this issue. This may be on manager parameterizing the auction to select tight upper/lower bounds.

FlattestWhite

Agree won't fix - will look at again if we launch on an L2.

snake-poison

The equivalent effect to this on L1 would be a reorg that would move time forward but not have had any bids on the canonical chain. The protection on this is the manager setting an appropriate floor for the auction as the "loss" outcome is no different than having no participants.

JJtheAndroid

Escalate

This issue should be invalid.

Each auction has a min/max price

Any asset price purchased within min/max bounds set by the manager, is what the manager is willing to accept in terms of asset price volatility. These are not "bad" prices as described in the report. If the set token manager doesn't like the price that his/her asset is being sold for, then they simply set the min price of the auction too low, making this an admin input error which is invalid as per Sherlock's rules.

sherlock-admin2

> Escalate
>
> This issue should be invalid.
>
> Each auction has a min/max price
>
> Any asset price purchased within min/max bounds set by the manager, is what the manager is willing to accept in terms of asset price volatility. These are not "bad" prices as described in the report. If the set token manager doesn't like the price that his/her asset is being sold for, then they simply set the min price of the auction too low, making this an admin input error which is invalid as per Sherlock's rules.

You've created a valid escalation!

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.

IAm0x52

A dutch auction price bounds are set specifically with the max price above the current market price and the min below the current market price. The expectation is that the auction will execute efficiently when the market price is the same as the auction price. I have shown a scenario where the auction is unable to execute as expected due to sequencer downtime. While the admin can lessen the potential loss of this, due to the nature of a dutch auction they cannot prevent it simply with a min/max bound on price.

pblivin0x

I am fine with whatever result for this issue. Medium or Low.

Oot2k

I think this is valid, (in past this issue has been valid and the report shows clear impact) Team mentions that they plan to deploy on layer2 so even if this is a "wont fix" I believe its valid.

0xauditsea

@Oot2k - Where is impact at all? No loss of tokens, benefits for users.

pblivin0x

> @Oot2k - Where is impact at all? No loss of tokens, benefits for users.

Suppose we have a dutch auction which starts at 10% above market price and ends at 10% below market price, if sequencer goes down, the auction never had a chance to fill at 0%, and users are hurt because the SetToken did not perform a valid L2 check on their auction

0xauditsea

No more comments attached, hope you guys make a right decision. Needs fairness.

hrishibhat

@0xauditsea

> hope you guys make a right decision. Needs fairness.

I think this comment explains why this issue is valid and is fair to reward this.

hrishibhat

Result: Medium Unique Considering this a valid medium based on the above comments https://github.com/sherlock-audit/2023-06-Index-judging/issues/40#issuecomment-1664743039

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- JJtheAndroid: rejected

pblivin0x

This issue will not be resolved in remediations, but the following warning was added to the contract documentation https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L53-L55

Issue M-4: Full inventory asset purchases can be DOS'd via frontrunning

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/41

Found by

0x52, Arabadzhiev

Summary

Users who attempt to swap the entire component value can be frontrun with a very small bid making their transaction revert

Vulnerability Detail

AuctionRebalanceModuleV1.sol#L795-L796

```
// Ensure that the component quantity in the bid does not exceed the available
↪  auction quantity.
require(_componentQuantity <= bidInfo.auctionQuantity, "Bid size exceeds auction
↪  quantity");
```

When creating a bid, it enforces the above requirement. This prevents users from buying more than they should but it is also a source of an easy DOS attack. Assume a user is trying to buy the entire balance of a component, a malicious user can frontrun them buying only a tiny amount. Since they requested the entire balance, the call with fail. This is a useful technique if an attacker wants to DOS other buyers to pass the time and get a better price from the dutch auction.

Impact

Malicious user can DOS legitimate users attempting to purchase the entire amount of component

Code Snippet

AuctionRebalanceModuleV1.sol#L772-L836

Tool used

Manual Review

Recommendation

Allow users to specify type(uint256.max) to swap the entire available balance

Discussion

pblivin0x

The recommendation here is that we allow `componentQuantity` to be specified in excess of the auction size? so replace the current check

```
// Ensure that the component quantity in the bid does not exceed the available
↪   auction quantity.
require(_componentQuantity <= bidInfo.auctionQuantity, "Bid size exceeds auction
↪   quantity");
```

with a enforced cap

```
if (_componentQuantity > bidInfo.auctionQuantity) {
    _componentQuantity = bidInfo.auctionQuantity;
}
```

I was originally hesitant because of some unintuitive UX, but if this removes the potential for a DOS attack, i think it is worth implementing.

FlattestWhite

Hmmm we should probably allow user to specify `maxQuantity` rather than the absolute quantity they want to buy

snake-poison

> The recommendation here is that we allow `componentQuantity` to be specified in excess of the auction size? so replace the current check
>
> ```
> // Ensure that the component quantity in the bid does not exceed the
> ↪   available auction quantity.
> require(_componentQuantity <= bidInfo.auctionQuantity, "Bid size exceeds
> ↪   auction quantity");
> ```
>
> with a enforced cap
>
> ```
> if (_componentQuantity > bidInfo.auctionQuantity) {
>     _componentQuantity = bidInfo.auctionQuantity;
> }
> ```
>
> I was originally hesitant because of some unintuitive UX, but if this removes the potential for a DOS attack, i think it is worth implementing.

I believe what the submitter was recommending was something more akin to keeping the :

```
 if (_componentQuantity == type(uint256).max) {
     _componentQuantity = bidInfo.auctionQuantity;
} else {
require(_componentQuantity <= bidInfo.auctionQuantity, "Bid size exceeds auction
↪  quantity");
}
```

note: the difference in gas is trivial, but it isn't equivalent to your suggestion so I wanted to point it out. My examples assumes pragma > 0.7 to use the type().max syntax but the older idiomatic `uint256(-1)` would.

sherlock-admin2

> Escalate
>
> Severity should not be Medium, has to be low or invalid, because there is no incentive at all for front-runners and also based on Sherlock's documentation, DOS < 1yr is not a valid one.

The escalation could not be created because you are not exceeding the escalation threshold.

You can view the required number of additional valid issues/judging contest payouts in your Profile page, in the Sherlock webapp.

JJtheAndroid

Escalate

This issue should be low/invalid as per Sherlock's rules https://docs.sherlock.xyz/audits/judging/judging. In addition, this is an edge case with no user funds at risk.

sherlock-admin2

> Escalate
>
> This issue should be low/invalid as per Sherlock's rules https://docs.sherlock.xyz/audits/judging/judging. In addition, this is an edge case with no user funds at risk.

You've created a valid escalation!

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.

Arabadzhiew

> Escalate

> This issue should be low/invalid as per Sherlock's rules https://docs.sherlock.xyz/audits/judging/judging. In addition, this is an edge case with no user funds at risk.

Can't agree on this. This is definitely not an edge case. There is an incentive to perform such kind of a DoS - If the price of the auction decreases over time, users will probably want to prolong it as much as they can. Additionally, the DoS can very easily happen unintentionally, when a lot of bid transactions get executed in the same block.

JJtheAndroid

> > Escalate This issue should be low/invalid as per Sherlock's rules https://docs.sherlock.xyz/audits/judging/judging. In addition, this is an edge case with no user funds at risk.
>
> Can't agree on this. This is definitely not an edge case. There is an incentive to perform such kind of a DoS - If the price of the auction decreases over time, users will probably want to prolong it as much as they can. Additionally, the DoS can very easily happen unintentionally, when a lot of bid transactions get executed in the same block.

In your scenario, users can only prolong it as long the current price is higher than min price. Malicious front running is useless beyond that point, because the price cannot go any lower. This would be invalid as per Sherlock rules on DOS attacks.

The unintentional scenario is not a DOS, it is just multiple people bidding at the same time. This is by design.

Again, no user funds are at risk. This should not be a med

IAm0x52

> Can't agree on this. This is definitely not an edge case. There is an incentive to perform such kind of a DoS - If the price of the auction decreases over time, users will probably want to prolong it as much as they can. Additionally, the DoS can very easily happen unintentionally, when a lot of bid transactions get executed in the same block.

Agreed with this. Given the nature of a dutch auction, the temporary DOS will prevent the auction from executing as expected and lead to assets being sold under market value.

pblivin0x

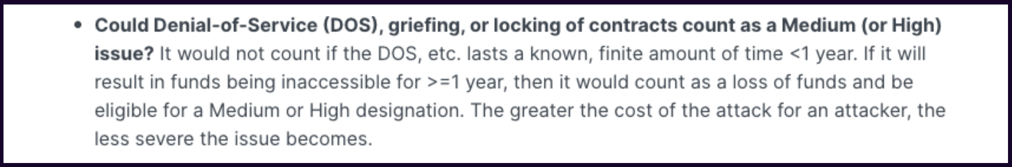
I agree this is a valid Medium

0xauditsea

valid Medium?

0xauditsea

Here's the docs about the severity of DOS:
https://docs.sherlock.xyz/audits/judging/judging



- **Could Denial-of-Service (DOS), griefing, or locking of contracts count as a Medium (or High) issue?** It would not count if the DOS, etc. lasts a known, finite amount of time <1 year. If it will result in funds being inaccessible for >=1 year, then it would count as a loss of funds and be eligible for a Medium or High designation. The greater the cost of the attack for an attacker, the less severe the issue becomes.

Hope this helps in escalation! Thanks y'all!

hrishibhat

@0xauditsea @JJtheAndroid There seems to be some confusion about the rule for DOS. Funds not being accessible temporarily does not apply here. Here the DOS results in loss of funds which is considered to be a medium:

> Agreed with this. Given the nature of a dutch auction, the temporary DOS will prevent the auction from executing as expected and lead to assets being sold under market value.

0xauditsea

@hrishibhat - Why is there loss of funds at all? The auction has minimum price defined which can be lower than the market value, that's totally fine for users to buy tokens with lower price, it's benefit for users, acceptable for auction manager.

hrishibhat

@pblivin0x @0xauditsea Correct me if I'm wrong, assuming here it is a Dutch auction where price reduces over time. If i can DOS someone who is ready to bid for a higher price and I bid for a lesser price after some time. Isn't this an issue?

hrishibhat

To add to my comment: the Max min values are values that are acceptable by the manager if the users decide to bid for the value within that range. That does not mean the code should unfairly allow someone to stop a higher bid and bid at a lower price. This seems like valid issue that does not allow normal functioning of dutch auction.

0xauditsea

@hrishibhat - In selling auction (users buy component), component price goes lower as time goes. So when it's front-run, the user will try again with lower price, which is fine for the user, also no problem with the auction manager.

0xauditsea

This is basically not loss of funds.

JJtheAndroid

> To add to my comment: the Max min values are values that are acceptable by the manager if the users decide to bid for the value within that range. That does not mean the code should unfairly allow someone to stop a higher bid and bid at a lower price. This seems like valid issue that does not allow normal functioning of dutch auction.

I don't want to go back and forth on this. I just want to say that your description is inaccurate. The report does not describe DOS on higher bids, it is a DOS on **full inventory** bids which is a rarer occurrence. Big difference. Also such an "attack" does not benefit the attacker nor does it hurt the "victim" because they both want to buy assets at a lower price.

Finally, there is a min price bound set by the manager. So any full inventory DOS at that point is completely useless and a waste of gas.

All of this is assuming that there are only 2 actors (the attacker and the victim) bidding. In reality, there will likely be many more and not all of them will submit a full inventory bid.

I will not comment on this further

pblivin0x

> @pblivin0x @0xauditsea Correct me if I'm wrong, assuming here it is a Dutch auction where price reduces over time. If i can DOS someone who is ready to bid for a higher price and I bid for a lesser price after some time. Isn't this an issue?

We have a 1000 ETH auction, a legitimate bidder wants to settle the full 1000 ETH auction at `PRICE_HIGH`

The malicious bidder is willing to settle the 1000 ETH auction at `PRICE_LOW`, so they frontrun the legitimate bidder with `SMALL_SIZE` bid.

If some transactions are successfully DOS'd, the legitimate bidder could submit a 1000 ETH - `SMALL_SIZE` bid, or something like a 500 ETH bid, such that the malicious bidder is not willing to front run at that size

Oot2k

> > Escalate This issue should be low/invalid as per Sherlock's rules https://docs.sherlock.xyz/audits/judging/judging. In addition, this is an edge case with no user funds at risk.
>
> Can't agree on this. This is definitely not an edge case. There is an incentive to perform such kind of a DoS - If the price of the auction decreases over time, users will probably want to prolong it as much as they can. Additionally, the DoS can very easily happen unintentionally, when a lot of bid transactions get executed in the same block.

I think this is valid, this comment explains it well.

0xauditsea

> > @pblivin0x @0xauditsea Correct me if I'm wrong, assuming here it is a Dutch auction where price reduces over time. If i can DOS someone who is ready to bid for a higher price and I bid for a lesser price after some time. Isn't this an issue?
>
> We have a 1000 ETH auction, a legitimate bidder wants to settle the full 1000 ETH auction at `PRICE_HIGH`
>
> The malicious bidder is willing to settle the 1000 ETH auction at `PRICE_LOW`, so they frontrun the legitimate bidder with `SMALL_SIZE` bid.
>
> If some transactions are successfully DOS'd, the legitimate bidder could submit a 1000 ETH - `SMALL_SIZE` bid, or something like a 500 ETH bid, such that the malicious bidder is not willing to front run at that size

Totally agree with this, no way front-runners would try it.

Arabadzhiew

> > @pblivin0x @0xauditsea Correct me if I'm wrong, assuming here it is a Dutch auction where price reduces over time. If i can DOS someone who is ready to bid for a higher price and I bid for a lesser price after some time. Isn't this an issue?
>
> We have a 1000 ETH auction, a legitimate bidder wants to settle the full 1000 ETH auction at `PRICE_HIGH`
>
> The malicious bidder is willing to settle the 1000 ETH auction at `PRICE_LOW`, so they frontrun the legitimate bidder with `SMALL_SIZE` bid.
>
> If some transactions are successfully DOS'd, the legitimate bidder could submit a 1000 ETH - `SMALL_SIZE` bid, or something like a 500 ETH bid, such that the malicious bidder is not willing to front run at that size

Correct me if I'm wrong, but even in your example, the legitimate bidder is most likely still going to end up buying the ETH at a lower price, leading to the protocol receiving less assets that it could have received.

The main issue here is that due to the current implementation, full inventory purchases are going to end up being reverted most of the time, be it due to intentional DoS attacks, or simply because there were a lot of bid transactions executed at the given time (for example, if there were 5 bid transactions sitting in the mempool with the same gas price and one of them was for a full inventory purchase, if that one does not get executed first, it will simply be reverted), leading to the component assets being sold at lower prices.

0xauditsea

@Arabadzhiew You guys keep saying components are being sold at lower prices, if auction manager doesn't want them to be sold at lower prices, they should increase

MIN price. When auction manager defines MIN price, it surely means that purchase at MIN price is pretty acceptable, this is pretty logical thing. Components being sold at lower price, good for buyers, acceptable for the auction manager, what is the problem here at all?

Regarding the example you mentioned above, you are right that full purchase bid will be reverted when there is another bid tx is executed before it, I think that's fine, that's what the auction is for - first buyer gets what they want. If you don't agree with this and let full purchase tx buy all remaining amount, it will cause an issue like, users wanted to buy whole 10WETH from the auction but they end up only buying 5WETH, do you think users will like this?

Ofc this issue needs to be fixed, but the severity can not be Med at all.

Arabadzhiew

> @Arabadzhiew You guys keep saying components are being sold at lower prices, if auction manager doesn't want them to be sold at lower prices, they should increase MIN price. When auction manager defines MIN price, it surely means that purchase at MIN price is pretty acceptable, this is pretty logical thing. Components being sold at lower price, good for buyers, acceptable for the auction manager, what is the problem here at all?
>
> Regarding the example you mentioned above, you are right that full purchase bid will be reverted when there is another bid tx is executed before it, I think that's fine, that's what the auction is for - first buyer gets what they want. If you don't agree with this and let full purchase tx buy all remaining amount, it will cause an issue like, users wanted to buy whole 10WETH from the auction but they end up only buying 5WETH, do you think users will like this?
>
> Ofc this issue needs to be fixed, but the severity can not be Med at all.

Sure, the MIN value is defined by the auction manager, but the fact that the component asset is going to be sold at a lower price is still a loss of funds for the protocol. The MIN value is there to make sure that the auction targets get reached, but I don't think the protocol team should be ok with receiving less assets, when they can receive more.

Also, regarding the mitigation recommended in this report, I think it is fine due to the fact that it is optional - users can only use the entire available balance purchase functionality if they explicitly say so, otherwise the bidding functionality should work as it currently does.

I won't comment on this issue any further. Let's let the Sherlock team decide whether it is a valid medium or not.

hrishibhat

Result: Medium Has duplicates After consideration of the above comments this issue is a valid medium, DOS of a valid bid at a certain price in a Dutch auction is considered damage to how the auction functions.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- JJtheAndroid: rejected

hrishibhat

Additionally, I see that the Sherlock rules are being interpreted incorrectly, will make sure to make the necessary changes to the docs and see how best any possible confusion can be avoided.

pblivin0x

The remediation for this issue is open for review here https://github.com/IndexCoop/index-protocol/pull/25

The changes are to allow users to specify `type(uint256.max)` to settle the remaining auction https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L811-L817

IAm0x52

Fix looks good. User can now set `_componentQuantity` to `type(uint256.max)` to buy the entire amount remaining

Issue M-5: Exponential and logarithmic price adapters will return incorrect pricing when moving from higher dp token to lower dp token

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/42

Found by

0x52

Summary

The exponential and logarithmic price adapters do not work correctly when used with token pricing of different decimal places. This is because the resolution of the underlying expWad and lnWad functions is not fit for tokens that aren't 18 dp.

Vulnerability Detail

AuctionRebalanceModuleV1.sol#L856-L858

```
function _calculateQuoteAssetQuantity(bool isSellAuction, uint256
 ↪   _componentQuantity, uint256 _componentPrice) private pure returns (uint256) {
    return isSellAuction ? _componentQuantity.preciseMulCeil(_componentPrice) :
     ↪   _componentQuantity.preciseMul(_componentPrice);
}
```

The price returned by the adapter is used directly to call _calculateQuoteAssetQuantity which uses preciseMul/preciseMulCeil to convert from component amount to quote amount. Assume we wish to sell 1 WETH for 2,000 USDT. WETH is 18dp while USDT is 6dp giving us the following price:

```
1e18 * price / 1e18 = 2000e6
```

Solving for price gives:

```
price = 2000e6
```

This establishes that the price must be scaled to:

```
price dp = 18 - component dp + quote dp
```

Plugging in our values we see that our scaling of 6 dp makes sense.

BoundedStepwiseExponentialPriceAdapter.sol#L67-L80

```
uint256 expExpression = uint256(FixedPointMathLib.expWad(expArgument));

// Protect against priceChange overflow
if (scalingFactor > type(uint256).max / expExpression) {
    return _getBoundaryPrice(isDecreasing, maxPrice, minPrice);
}
uint256 priceChange = scalingFactor * expExpression - WAD;

if (isDecreasing) {
    // Protect against price underflow
    if (priceChange > initialPrice) {
        return minPrice;
    }
    return FixedPointMathLib.max(initialPrice - priceChange , minPrice);
```

Given the pricing code and notably the simple scalingFactor it also means that priceChange must be in the same order of magnitude as the price which in this case is 6 dp. The issue is that on such small scales, both lnWad and expWad do not behave as expected and instead yield a linear behavior. This is problematic as the curve will produce unexpected behaviors under these circumstances selling the tokens at the wrong price. Since both functions are written in assembly it is very difficult to determine exactly what is going on or why this occurs but testing in remix gives the following values:

```
expWad(1e6) - WAD = 1e6
expWad(5e6) - WAD = 5e6
expWad(10e6) - WAD = 10e6
expWad(1000e6) - WAD = 1000e6
```

As seen above these value create a perfect linear scaling and don't exhibit any exponential qualities. Given the range of this linearity it means that these adapters can never work when selling from higher to lower dp tokens.

Impact

Exponential and logarithmic pricing is wrong when tokens have mismatched dp

Code Snippet

BoundedStepwiseExponentialPriceAdapter.sol#L28-L88

BoundedStepwiseLogarithmicPriceAdapter.sol#L28-L88

Tool used

Manual Review

Recommendation

scalingFactor should be scaled to 18 dp then applied via preciseMul instead of simple multiplication. This allows lnWad and expWad to execute in 18 dp then be scaled down to the correct dp.

Discussion

pblivin0x

Agree that scalingFactor should be 18 decimals and applied with preciseMul, will fix.

0ot2k

Not a duplicate

bizzyvinci

Escalate

This is invalid

`FixedPointMathLib.expWad` and `FixedPointMathLib.lnWad` uses WAD as input and WAD as output. This is mentioned in docs and you can test it out on remix. Therefore, `exp(1) = FixedPointMathLib.expWad(WAD) / WAD` and `exp(5) = FixedPointMathLib.expWad(5*WAD) / WAD`.

`expWad(1e6) - WAD = 1e6` is equal to `exp(1e-12) - 1 = 1e-12` which is absolutely correct.

For the formula to work, timeCoefficient has to be in WAD. @pblivin0x should look at our DM around this time.

His comment: `scalingFactor should be 18 decimals and applied with preciseMul` is on a different matter. It's a plan for the future to allow decimal scalingFactor e.g 0.5, 2.5 rather than just integers like 1, 2, 3 etc.

To recap: `block.timestamp` is in seconds, therefore `timeBucket`, `_timeElapsed` and `bucketSize` are in seconds. `_componentPrice`, `initialPrice`, `minPrice`, `maxPrice`, `priceChange` and `FixedPointMathLib` are in WAD. Therefore, `expExpression`, `expArgument` and `timeCoefficient` has to also be in WAD. `scalingFactor` is just a scaler unit which the team plan to turn into WAD in the future for more precision with scaling.

sherlock-admin2

> Escalate

> This is invalid
>
> `FixedPointMathLib.expWad` and `FixedPointMathLib.lnWad` uses WAD as input and WAD as output. This is mentioned in docs and you can test it out on remix. Therefore, `exp(1) = FixedPointMathLib.expWad(WAD) / WAD` and `exp(5) = FixedPointMathLib.expWad(5*WAD) / WAD`.
>
> `expWad(1e6) - WAD = 1e6` is equal to `exp(1e-12) - 1 = 1e-12` which is absolutely correct.
>
> For the formula to work, timeCoefficient has to be in WAD. @pblivin0x should look at our DM around this time.
>
> His comment: `scalingFactor should be 18 decimals and applied with preciseMul` is on a different matter. It's a plan for the future to allow decimal scalingFactor e.g 0.5, 2.5 rather than just integers like 1, 2, 3 etc.
>
> To recap: `block.timestamp` is in seconds, therefore `timeBucket`, `_timeElapsed` and `bucketSize` are in seconds. `_componentPrice`, `initialPrice`, `minPrice`, `maxPrice`, `priceChange` and `FixedPointMathLib` are in WAD. Therefore, `expExpression`, `expArgument` and `timeCoefficient` has to also be in WAD. `scalingFactor` is just a scaler unit which the team plan to turn into WAD in the future for more precision with scaling.

You've created a valid escalation!

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.

IAm0x52

Since scaling factor is not applied via precise mul in the current implementation, in order to work as the code is written it has to be have the same number of decimals as price and therefore can't be WAD, regardless of what sponsor has said in the discord comments. As I've shown in my issue these smaller values are not compatible with expWAD and lnWAD and will produce incorrect values, negatively affecting auction pricing.

bizzyvinci

The getPrice function can be broken down to the following after removing `boundary price` and type conversions.

1. timeBucket = _timeElapsed / bucketSize
2. expArgument = timeCoefficient * timeBucket
3. expExpression = FixedPointMathLib.expWad(expArgument)
4. priceChange = scalingFactor * expExpression - WAD

5. price = initialPrice + priceChange (or minus)

To know what unit should be WAD or not, we need to look elsewhere.

1. `_timeElapsed = block.timestamp - rebalanceInfo[_setToken].rebalanceStartTime` here and `rebalanceInfo[_setToken].rebalanceStartTime` is set to `block.timestamp` when startRebalance is called here. Therefore `_timeElapsed`, `bucketSize` and `timeBucket` has to be seconds.
2. `_componentPrice` is in precise unit or WAD based on calculation here. Therefore `price`, `initialPrice` and `priceChange` have to also be in WAD.
3. Formula 4 is wrong as pointed out in #39 therefore let's just focus on the multiplication part and assume `priceChange = scalingFactor * expExpression`. If `priceChange` is `WAD`, then `scalingFactor * expExpression` has to be WAD. Either `scalingFactor is WAD` or `expExpression` is WAD.
4. FixedPointMathLib.expWad returns WAD, so `expExpression` is indeed WAD. So `scalingFactor` is basic unit.
5. Furthermore, `FixedPointMathLib.expWad` takes WAD as input, and that input is `timeCoefficient * timeBucket`. We've established that `timeBucket` is seconds in 1, so therefore `timeCoefficient` has to be WAD.

The sponsor's message was referenced because

- I was the one who decided that his initial statement `timeCoefficient and bucketSize are not in WAD` is wrong. So he might want to cross-check things again.
- We had some discussions about scalingFactor and converting it to WAD around that time.

If scalingFactor is changed to WAD, then priceChange would be `WAD^2`. Therefore, we must use preciseMul to keep things balanced again. P.S: -WAD is ignored again.

```diff
+ priceChange = scalingFactorWAD.preciseMul(expExpression)
- priceChange = scalingFactorBasic * expExpression
```

The 2 formula are the same thing because `preciseMul(a, b) = a * b / WAD` code

```
function preciseMul(uint256 a, uint256 b) internal pure returns (uint256) {
    return a.mul(b).div(PRECISE_UNIT);
}
```

IAm0x52

> _componentPrice is in precise unit or WAD based on calculation here. Therefore price, initialPrice and priceChange have to also be in WAD.

This is incorrect. I've proven in my submission above that when going from an 18 dp token to 6 dp that price has to be 6 dp. Since scalingFactor is applied as a scaler and not via preciseMul then expArgument and expExpression have to also be in 6 dp as well. If you used a WAD expression for them the pricing would be completely wrong as it would return an 18 dp price. As I've shown expWAD returns incorrectly when inputting a 6 dp number.

IAm0x52

The point of this issue is to prove that scaling factor must be applied via preciseMul or else the price cannot work as expect. To just say "scaling factor should be applied via preciseMul" is not a valid issue unless you can show why it's incorrect that it's not applied that way and the damages that it causes.

bizzyvinci

precise unit is used for precision in calculation because the numbers could be very small and solidity does automatic rounding. When multiplying or dividing, preciseMul or preciseDiv is used to finally get rid of that precision. You can view the PreciseUnitMath library and the key take away are

1. `PRECISE_UNIT = PRECISE_UNIT_INT = WAD = 1e18`
2. `preciseMul(a, b) = a * b / WAD` and that only makes sense if a or b is WAD
3. `preciseDiv(a, b) = a * WAD / b` and that only makes sense if b is WAD

Now, why is `_componentPrice` in WAD? Because `_componentQuantity` and the returned `quoteAssetQuantity` are not in WAD and preciseMul needs `b` to be WAD. `_componentQuantity` and `quoteAssetQuantity` are the raw quantity amount that would be transferred with `token.transfer`.

https://github.com/sherlock-audit/2023-06-Index/blob/main/index-protocol/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L856-858

```
function _calculateQuoteAssetQuantity(bool isSellAuction, uint256
↪   _componentQuantity, uint256 _componentPrice) private pure returns (uint256) {
      return isSellAuction ? _componentQuantity.preciseMulCeil(_componentPrice)
↪   : _componentQuantity.preciseMul(_componentPrice);
  }
```

pblivin0x

I believe this is a valid medium

Oot2k

I agree that this is valid medium

bizzyvinci

I still stand by my escalation and I think my proof is sufficient cause it proves the following

- _componentPrice in WAD
- Right now, scalingFactor is not WAD cause that would be mathematically wrong. It must be a normal integer.
- The unit of each parameters (e.g which one is seconds, WAD or int) to show that price does work correctly.
- The team plans to make scalingFactor WAD and use preciseMul. They **must** use preciseMul to make sure priceChange remains a WAD (rather than WAD^2)
- The migration of scalingFactor from integer scalar to WAD scalar **would not** change price nor priceChange because `preciseMul(a,b) = a * b / WAD`
- The only effect the migration has is precision. scalingFactor could then be represented as decimals instead of just integers.

If anyone disagrees it would be nice if they state why. Or if there's a point that wasn't clear, I'm here to clarify.

bizzyvinci

I understand the proof might be daunting to comprehend so I'll recommend using pen and paper (and maybe remix with calculator) to make things easier.

bizzyvinci

I do agree that my escalation be rejected

hrishibhat

Result: Medium Has duplicates Considering this a valid medium based on the above comments.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- bizzyvinci: rejected

pblivin0x

The remediation for this issue is open for review here https://github.com/IndexCoop/index-protocol/pull/25

The changes are to update to 18 decimal `scalingFactor` in both the exponential and logarithmic adapter

- https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/integration/auction-price/BoundedStepwiseExponentialPriceAdapter.sol#L73
- https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/integration/auction-price/BoundedStepwiseLogarithmicPriceAdapter.sol#L73

IAm0x52

Fix looks good. `scalingFactor` is now applied via mulWad

Issue M-6: Target raises can be highly damaging for dutch auctions with multiple components

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/45

Found by

0x52

Summary

Multi-component dutch auctions are fundamentally incompatible with target raises and will lead to inefficient pricing causing loss to set token.

Vulnerability Detail

The AuctionRebalanceModuleV1 allows targets to be increased when all component targets have been met and there is still excess quote token. When combined with multiple components, it his highly likely that these target raises will lead to inefficient pricing which will cause loss to the set token.

Consider the following a set token has the following composition that has target raises enabled:

40% USDC 30% WBTC 30% WETH

The manager wishes to rebalance the set to the following using USDC as the quote token:

20% USDC 40% WBTC 40% WETH

Assume the WETH portion of the execute within the first hour of the auction. The WBTC on the other hand doesn't execute until 12 hours in. Assume there is excess quote so the target is increased. The issue is that now because of the change in time, the WETH auction is now well above the market price. This buys the WETH for a large loss compared to the market price of WETH.

Impact

Pricing after target raises will likely be heavily skewed from market prices for some components lead to set token losses

Code Snippet

AuctionRebalanceModuleV1.sol#L359-L380

Tool used

Manual Review

Recommendation

Target raises should reset `rebalanceStartTime` allowing the dutch auction to restart and properly price the assets

Discussion

pblivin0x

Agree, especially since the raising of targets is onlyAllowedBidder, we should reset the pricings.

In the fix, I think we will make it such that the rebalance still ends at the same time.

bizzyvinci

Escalate

This is invalid. Let's take a look at why would we would raise target based on the docs

```
* @dev ACCESS LIMITED: Increases asset targets uniformly when all target units
↪  have been met but there is remaining quote asset.
```

raiseAssetTarget is meant for uniformly raising targets **when all target units have been met**. Everything else (such as price and ratio) can be resolved by calling `startRebalance`.

sherlock-admin2

> Escalate
>
> This is invalid. Let's take a look at why would we would raise target based on the docs
>
> ```
> * @dev ACCESS LIMITED: Increases asset targets uniformly when all target
> ↪  units have been met but there is remaining quote asset.
> ```
>
> raiseAssetTarget is meant for uniformly raising targets **when all target units have been met**. Everything else (such as price and ratio) can be resolved by calling `startRebalance`.

You've created a valid escalation!

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

@bizzyvinci Could you please elaborate a bit further as to what exactly is invalid about the points raised in the issue?

bizzyvinci

First of all, I believe the auction buy and sell is to be used instead of trade module and this was mentioned on the discord channel. And the way it works is that it provides discount over market price to motivate bidder. And this discount increases linearly, exponential or logarithmically.

And an assumption is that the discount will start low e.g 0% till a cap the manager is comfortable with e.g 5%. Whether you're buying or selling, the focus is on `discount` and that's why price would decrease during sell and price would increase during buy of quoteAsset. The price change is the discount and that's what grows or fall with the mathematical equation.

Assuming manager/operator wants to move from 40% USDC, 30% WBTC, 30% WETH

To 20% USDC, 40% WBTC, 40% WETH using USDC

If after an hour, the ratio are 35% USDC, 30% WBTC, 35% WETH. It's safe to assume that price on WETH was too good but price on WBTC wasn't. The operator can call `startRebalance` to **update this prices**.

If the price is indeed good and the market/whale is just crazy for WBTC and it reaches 30% USDC, 30% WBTC, 40% WETH. Then no one can bid on WETH again, because of the following requirements

```
require(currentUnit != targetUnit, "Target already met");

// Determine whether the component is being sold (sendToken) or bought
isSellAuction = targetNotional < currentNotional;

// Calculate the max quantity of the component to be exchanged. If buying,
↪   account for the protocol fees.
maxComponentQty = isSellAuction
    ? currentNotional.sub(targetNotional)
    : targetNotional.sub(currentNotional).preciseDiv(PreciseUnitMath.preciseUnit
↪   ().sub(protocolFee));
```

Therefore, the best thing to do is wait till the discount on WBTC is suitable and every WBTC is munched on. Thereby reaching the goal of `20% USDC, 40% WBTC, 40% WETH`.

Excess USDC means USDC is greater than 20%. And that also means either WBTC or WETH is less than 40% or both. And it is possible to be unable to bid cause targetUnit of WBTC and WETH has been reached. However, all target unit has not been reached because of USDC. The solution is to call startRebalance again and **update targetUnits** (and maybe price too is depending on what is considered fair price).

bizzyvinci

An example of when to use `raiseAssetTargets`.

Assuming USDC is quoteAsset but not a component. And we want WETH and WBTC to be 50 and 50 respectively.

If targetUnit and hence the 50:50 is reached and there's still USDC. We could use raiseAssetTarget to uniformly raise the target unit by 10% to 55:55. And that's still a 50% ratio for each component.

pblivin0x

I believe this is a valid medium

bizzyvinci

I still stand by my escalation because my argument is that the manager could call `startRebalance` at any point in time during an active auction. With this function `startRebalance`, they could change price, price curve, and target unit. Therefore no matter the unfavourable price, target unit or ratio, the manager has the option to call `startRebalance` instead of `raiseAssetTarget`.

The only reason a manager would call `raiseAssetTarget` instead of `startRebalance` is when

- All target unit have been met
- And they want to uniformly raise all the target units

And (though it was not explicitly stated in the docs)

- manager is comfortable with the current price and direction. Cause if they are not, they could call `startRebalance` to update the price while raising all target units.

pblivin0x

> I still stand by my escalation because my argument is that the manager could call `startRebalance` at any point in time during an active auction. With this function `startRebalance`, they could change price, price curve, and target unit. Therefore no matter the unfavourable price, target unit or ratio, the manager has the option to call `startRebalance` instead of `raiseAssetTarget`.

> The only reason a manager would call `raiseAssetTarget` instead of `startRebalance` is when
>
> - All target unit have been met
> - And they want to uniformly raise all the target units
>
> And (though it was not explicitly stated in the docs)
>
> - manager is comfortable with the current price and direction. Cause if they are not, they could call `startRebalance` to update the price while raising all target units.

Upon further review, I actually change my opinion, and agree with this escalation.

I think that the listed remediation is not satisfactory and that it is on the SetToken manager to decide whether raiseAssetTargets or a fresh startRebalance call is preferable given their price curves.

Assume fair market prices of 1850 for ETH and 29000 for WBTC.

Suppose we have a rebalance with the following individual component auctions

- An auction that sells WETH in exchange for DAI. Auction price begins at 2200 and lowers to a price of 1800.
- An auction that buys WBTC with DAI. Auction price begins at 25000 and raises to a price of 30000.

Now suppose that the auctions fill, there are remaining DAI units, and both auction price curves have reached their final price (1800 and 30000).

If asset targets are raised, all auctions now become buy auctions.

If we do not reset `rebalanceStartTime` when we raise asset targets, we have

- Good: An auction that buys WETH with DAI at a price of 1800. This is below market value, unlikely to get filled, but not a risk for SetToken holders
- Bad: An auction that buys WBTC with DAI at a price of 30000. This is above market value, and a risk to SetToken holders to lose NAV

If we do reset `rebalanceStartTime` when we raise asset targets, we have

- Bad: An auction that buys WETH with DAI at a price of 2200. This is above market value, and a risk to SetToken holders to lose NAV
- Good: An auction that buys WBTC with DAI at a price of 25000. This is below market value, unlikely to get filled, but not a risk for SetToken holders

---

In conclusion

- I agree with the stated vulnerability - `Target raises can be highly damaging for dutch auctions with multiple components`
- I disagree with the listed remediation - resetting the `rebalanceStartTime` does not automatically lead to proper pricing of the component auctions, because sell auctions have flipped to buy auctions on the target raise.
- I agree with the escalation - It is on the SetToken manager to decide whether their auction price curves are appropriate for a nonzero `raiseTargetPercentage`. If it is not appropriate, then they need to call a fresh `startRebalance()`.

bizzyvinci

`raiseAssetTargets` is for when you are buying all components (those listed for bidding) and you want to sell `quoteAsset` till it reaches 0 or a specified limit. This was also mentioned in the docs (@pblivin0x could update docs if the wordings are not clear enough for most users)

> This helps in reducing tracking error and providing greater granularity in reaching an equilibrium between the excess quote asset and the components to be purchased.

hrishibhat

Additional Sponsor comment:

> confirming i see this as a low. manager needs to decide between proper calls (setTargetRaisePercentage or startRebalance) based on the AuctionExecutionParams they inputted

pblivin0x

This issue's remediation has been removed from https://github.com/IndexCoop/index-protocol/pull/25 pending escalation resolution

IAm0x52

I disagree that this is admin's responsibility. The feature is dangerous in this scenario. There is no "safe" parameter that admin can use. Their only option is to not use the feature, which I don't think is a valid. It also has to be considered that the admin can't turn off rebalances after they have been enabled. I still hold this is a valid medium. #44 also provides another way this can be abused. I know that sponsor has commented that donation doesn't work but the donation occurs before the buy which causes the donated balance to be counted and to reflect in the set token balances when it updates the balances of the set token. Both this and #44 have the same root cause (not resetting the price after increase) which is why I didn't escalate that one.

hrishibhat

Result: Medium Unique Although the suggested remediation does not solve the problem the issue identified is valid. Sponsor:

> I agree with the stated vulnerability - Target raises can be highly damaging for dutch auctions with multiple components

Additional Sponsor comment:

> raising asset targets is a legacy feature from the GeneralIndexModule, and as your issue correctly points out, it doesnt really make sense with dutch auctions
>
> i am fine with any issue validity ruling here.

Considering this a valid medium based on the additional sponsor comment and the Lead Watson comment

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- bizzyvinci: rejected

Issue M-7: Malicious users can exploit the auction and make profit when the SetToken is not locked.

Source: https://github.com/sherlock-audit/2023-06-Index-judging/issues/57

Found by

ast3ros

Summary

The SetToken can be minted and redeemed by anyone when it is not locked during rebalancing. This can allow malicious users to front-run and back-run the bidders and manipulate the auction outcome.

Vulnerability Detail

When rebalancing, the token manager can configure if the SetToken is locked or not. If the SetToken is not locked, anyone can mint and redeem the SetToken using BasicIssuanceModule. The token manager can also configure the pricing mechanism via the priceAdapter. There are some mechanisms:

- ConstantPriceAdapter: the price is fixed - similar to place limit orders.
- BoundedStepWise adapters: like Dutch Auction which the price can increase/decrease over time.

Let's see an example:

The current price of WETH is 1940 USDC. Total supply of the SetToken is 10.

A Set Token with component WETH and current unit(1 WETH) wants to achieve target unit (0.5 WETH - 975 USDC).

- Current unit: 1 WETH => Current notional: 10 WETH
- Target unint: 0.5 WETH - 975 USDC => Target notional: (5 WETH - 9750 USDC)

To achieve this, it needs to sell WETH to buy USDC. The manager starts rebalancing using linear price curve: start at $2000, lower to minimum $1900, take steps of $0.1 every minute. It also chooses USDC as the quote token.

Assuming when the price of WETH reaches 1950 USDC, a bidder bids for all of the available WETH for the rebalance process, which is 0.5 WETH per Set Token or 5 WETH in total for 9750 USDC (5*1950). The expected result should be that the SetToken will meet the target and the rebalancing process will finish. The end position will be:

- Expected position: 0.5 WETH - 975 USDC => Expected notional (5 WETH - 9750 USDC)

However, the module is deployed on mainnet and polygon, a malicious user can front-run the bidder and mint the SetToken to make profit and disrupt the auction. The malicious user mints 10 SetToken using 10 WETH. It increases the total supply of the SetToken to 20.

- After the malicious user front-run the bidder:
  - Current unit: 1 WETH => Current notional: 20 WETH.
  - Target unit: 0.5 WETH - 975 USDC => Target notional: (10 WETH - 19500 USDC)
- After bidder bids 5 WETH for 9750 USDC:
  - Current: 0.75 WETH - 487.5 USDC => Current notional: (15 WETH - 9750 USDC)

After that, the malicious user can back-run the bidding transaction and redeem his 10 SetToken for 7.5 ETH and 4875 USDC. Malicious user balance:

- Before: 10 WETH = 19400 USDC
- After: 7.5 WETH + 4875 USDC = 7.5 * 1940 + 4875 = 19425 USDC.

The malicious user can make a profit of 25 USDC and disrupt the auction because the auction cannot finish as it should be.

He cannot make a profit directly by bidding because the bidder may need to be whitelisted by the manager.

In conclusion, if the price of auction is above the market price and a bid is placed, a malicious user can front-run and back-run the bidder and make a profit and disrupt the auction in the unlocked rebalancing process.

Impact

The malicious user can make a profit and prevent the auction from meeting the target and finishing.

Code Snippet

https://github.com/sherlock-audit/2023-06-Index/blob/main/index-protocol/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L254-L257

Tool used

Manual Review

Recommendation

The SetToken should be always locked when rebalancing.

Discussion

thangtranth

Escalate

This is not the duplication of #21 since it does not require ERC777. Please help to review.

sherlock-admin2

> Escalate
>
> This is not the duplication of #21 since it does not require ERC777. Please help to review.

You've created a valid escalation!

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.

pblivin0x

```
In conclusion, if the price of auction is above the market price and a bid
is placed, a malicious user can front-run and back-run the bidder and make a
profit and disrupt the auction in the unlocked rebalancing process.
```

If the bidder is bidding on an auction that is above market price, then yes, there is profit to be made in the system. That is by design.

An example of this would be a bidder desperate to exit a position, and willing to incur slippage to exit, just so happens that the exit is market making the SetToken auction.

Invalid issue, note that we've added logic to settle remaining units as part of remediation for #41

hrishibhat

@thangtranth

pblivin0x

For the security purposes of this audit, it can be considered that the price set by the trusted SetToken manager is valid, and any NAV decay of the SetToken needs to be measured against these prices.

By allowing bid's, mints, and redeems there is no clear way to decay NAV as defined by the auction prices.

Any arbitrage open between the auction prices and market prices is to be handled by the SetToken manager.

thangtranth

Hi @pblivin0x,

> If the bidder is bidding on an auction that is above market price, then yes, there is profit to be made in the system. That is by design.

In this issue, it shows that the profit goes to the malicious user who is not the current SetToken holder. He only buys SetToken when there is a profit bidding auction and sells immediately after that by back running.

He can make **risk free money** and can **prevent** the auction from getting the target. In the example above, if the bidder bids 9750 USDC, the auction should be completed, however it is not because profit is extracted by malicious user. Actually a MEV can increase the buying and selling amount and extract most of the profit from the bidding.

Because the protocol is deployed in mainnet and polygon then it is very common to happen. Therefore it should be addressed.

> An example of this would be a bidder desperate to exit a position, and willing to incur slippage to exit, just so happens that the exit is market making the SetToken auction.

There is nothing wrong with the bidder here. He gets his expected bidding price for the assets.

pblivin0x

> Hi @pblivin0x,
>
> > If the bidder is bidding on an auction that is above market price, then yes, there is profit to be made in the system. That is by design.
>
> In this issue, it shows that the profit goes to the malicious user who is not the current SetToken holder. He only buys SetToken when there is a profit bidding auction and sells immediately after that by back running.
>
> He can make **risk free money** and can **prevent** the auction from getting the target. In the example above, if the bidder bids 9750 USDC, the auction should be completed, however it is not because profit is extracted by malicious user. Actually a MEV can increase the buying and selling amount and extract most of the profit from the bidding.

> Because the protocol is deployed in mainnet and polygon then it is very common to happen. Therefore it should be addressed.
>
> > An example of this would be a bidder desperate to exit a position, and willing to incur slippage to exit, just so happens that the exit is market making the SetToken auction.
>
> There is nothing wrong with the bidder here. He gets his expected bidding price for the assets.

My current understanding is this, I would love to get more opinions.

If:

- A `bid()` is placed on a `component` sell auction
- The auction price is greater than the price perceived by some external actor

Then:

- The external actor can sandwich attack the `bid()` by issuing a size much greater than `totalSupply`, allowing the `bid()` to go through, and redeeming the same size. The full perceived value difference can be extracted.
- The SetToken manager and holders do not have sufficient protections that a `bid()` will push the `positionUnit` closer to the `targetUnit` by an amount proportional to `componentAmount` and `totalSupply` before the bid.

Fix:

- Apply a supply cap on `SetToken.totalSupply` when the manager calls `startRebalance()`. This will facilitate honest issuance and redemption while preventing the size needed for effective sandwich attacks.

If there is no supply cap on an unlocked auction, then anytime a malicious actor sees a `bid()` at a price sufficiently higher than their perceived price, then the malicious actor `can make risk free money and can prevent the auction from getting the target.` - @thangtranth

FlattestWhite

```
After that, the malicious user can back-run the bidding transaction and redeem
↪  his 10 SetToken for 7.5 ETH and 4875 USDC. Malicious user balance:

Before: 10 WETH = 19400 USDC
After: 7.5 WETH + 4875 USDC = 7.5 * 1940 + 4875 = 19425 USDC.
```

Why is Before 10 WETH = 19400 USDC? Isn't it 10 WETH = 19500 USDC After: 7.5 WETH + 4875 USDC = 7.5 * 1950 + 4875 = 19500 USDC

pblivin0x

Would a mint/redeem fee prevent the sandwich attack? @thangtranth

thangtranth

> Would a mint/redeem fee prevent the sandwich attack? @thangtranth

I think it makes the attack more expensive. Then the attacker needs to consider the amount of fees that he has to pay for mint + redeem and the profit gained (the gap between bid price and current price). If we config the fees large enough then it may. However honest users will have to pay the fee as well.

thangtranth

> ```
> After that, the malicious user can back-run the bidding transaction and
> ↪  redeem his 10 SetToken for 7.5 ETH and 4875 USDC. Malicious user
> ↪  balance:
>
> Before: 10 WETH = 19400 USDC
> After: 7.5 WETH + 4875 USDC = 7.5 * 1940 + 4875 = 19425 USDC.
> ```
>
> Why is Before 10 WETH = 19400 USDC? Isn't it 10 WETH = 19500 USDC
> After: 7.5 WETH + 4875 USDC = 7.5 * 1950 + 4875 = 19500 USDC

Please refer to the scenario: The current price of WETH is 1940 USDC. The 1950 is the price of the bid from bidder.

0xauditsea

I don't think this is valid, when those kind of MEV is allowed, auction managers will allow tokens not being locked, but otherwise `_shouldLockSetToken` will be set to true.

thangtranth

> I don't think this is valid, when those kind of MEV is allowed, auction managers will allow tokens not being locked, but otherwise `_shouldLockSetToken` will be set to true.

Hi, it is already confirmed with the protocol team that this is not the intended behaviour when MEV is allowed when unlocked. From the protocol team:

> it is concerning...it is ideal for SetToken's to actually not be locked during rebalancing bc we want users to always have access to their underlying

bizzyvinci

The issue with this issue is that it assumes a logical bidder would trade at a loss against SetToken.

I'll start with some axioms which I believe are True

- market is DEX. market price is DEX price.
- No logical bidder would trade at a loss. Because bidders would want to trade at a DEX instantly for profit.
- price is how much quoteAsset you need to purchase 1 component (which is base asset).
- Minting and burning does not manipulate unit.
- Notional values are what's minted, burned and traded, which is equal to `unit * tokenAmount` (tokenAmount could be totalSupply).

If the axioms listed above are True. Then the following should be True

- SetToken would sell component at a price lower than market/DEX. So that bidder would buy it and sell on DEX for an instant profit.
- SetToken would buy component at a price higher than market/DEX. So that bidder would buy it cheaper on DEX and sell it for SetToken for an instant profit.
- SetToken is trading at a discount loss compared to market.

P.S: Another reason I believe SetToken is comfortable at trading at discount loss is because Auction could be used against DEX slippage or as an order book that would be executed at a future time when price reaches the limit set by manager.

And

- If totalSupply increases when SetToken is selling component, It would be provided with more component notional to sell at a loss. Because It needs more component notional to buy quoteAsset at a loss.
- If totalSupply increases when SetToken is buying component, It would be provided with more quote notional to sell at a loss. Because It needs more quote notional to buy component at a loss.
- The loss is distributed to all holders of SetToken

P.S: quote asset must be a component for second point to be True. And I believe that's done because the contract checks that bidded component is not quoteAsset to avoid attacks when quoteAsset is part of SetToken components. And most importantly, it updates both quoteAsset and bidded component position here.

Therefore:

- Higher notional means SetToken is provided with more tokens to trade at a loss to market/DEX.
- Anyone sandwiching this trade by minting and burning SetToken is partaking in the loss.

Using numbers would be more complicated cause there are several parameters. And the proof provided by the issue is flawed because SetToken is selling component at a price higher than market/DEX. This means that bidder is taking a loss and SetToken is taking a profit. That's the source of the $25. Proof:

- Price of 1900-2000 means USDC is quote asset while WETH is component (base asset)
- SetToken is selling WETH for USDC
- A bidder decides to buy it 1950 which is higher than 1940 of market price
- The bidder bought 5 WETH, so the bidder loses $50 while the SetToken gains $50
- Since the sandwicher owns 50% of supply, they get 50% of $50 which is $25

If the bidder is logical he would wait till price is below market price so that he'll make a profit on the trade. Therefore sandwicher would take part in the loss.

sinarette

> Should not be able to decay the SetToken Net-Asset-Value according to this price. with any combination of actions (bids, mints, or redeems)

According to the contest readme it requires damage calculated according to the bid price. Here the stated attack scenario is just buying ETH at 1940 USDC and selling at 1950 USDC; we don't tell this kind of profitable trading stategy an 'attack'. In fact, in ETH units it's not a profit; the attacker who had 19400/1940 = 10 ETH now has 19425/1950 = 9.96 ETH.

pblivin0x

Agree with @bizzyvinci here that bidder's in the system are expected to be rationale profit seeking actors - `If the bidder is logical he would wait till price is below market price so that he'll make a profit on the trade. Therefore sandwicher would take part in the loss.`

thangtranth

Yes, I also agree with @bizzyvinci . A very good point about rationale bidder

pblivin0x

Thank you for all the input here @thangtranth @bizzyvinci @0xauditsea @sinarette @Oot2k

After much consideration I'm of the following opinion

- When a `bid()` is placed on a component sell auction that is above market price, a malicious actor can sandwich attack the `bid()`, and prevent the `bid()` from contributing meaningfully towards the auction getting closer to target.

- Bidder's are expected to be rationale profit-seeking actors, so the situation where a `bid()` is placed above market price is not expected to happen often.
- To prevent such a sandwich attack, a reasonable supply cap can be placed on the SetToken, such that normal user issuance and redemption is possible, but large sandwich attack issuance is not possible.

Index is prepared to take the following remediations

- Add a warning about sandwich attacks on unlocked rebalances to the `AuctionRebalanceModuleV1` natspec
- In production, utilize a supply cap on the SetToken during unlocked rebalances (e.g., 2x supply at the start of rebalance)

I believe this is a Medium severity issue because

- While the sandwich attack requires that the bidder is acting "irrationally" and is not expected to happen often, part of the dutch auction mechanism is that the auctioneer benefits from poorly priced bids. This sandwich attack could take all the benefits from the auctioneer, by preventing the auction from getting any closer to target.
- This is exactly the edge case I wanted examined during the audit, with the addition of broader market dynamics - `Should not be able to decay the SetToken Net-Asset-Value according to this price. with any combination of actions (bids, mints, or redeems)` https://github.com/sherlock-audit/2023-06-Index#q-are-there-any-additional-protocol-roles-if-yes-please-explain-in-detail
- Index is taking meaningful remediations from this issue

Oot2k

Agree with escalation and agree that this is valid.

hrishibhat

Result: Medium Unique Considering this a medium issue based on above comments https://github.com/sherlock-audit/2023-06-Index-judging/issues/57#issuecomment-1666224199

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- thangtranth: accepted

pblivin0x

The remediation for this issue is open for review here https://github.com/IndexCoop/index-protocol/pull/25

The changes are to add warnings to use a supply cap in order to avoid large front running issuance and redemption

- https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L48-L51
- https://github.com/IndexCoop/index-protocol/blob/839a6c699cc9217c8ee9f3b67418c64e80f0e10d/contracts/protocol/modules/v1/AuctionRebalanceModuleV1.sol#L239

IAm0x52

Sponsor has not made any changes to the smart contract but acknowledges this scenario and expects manager (trusted party) to set appropriate supply caps to prevent this.

MLON33

Confirming no changes to the smart contract. Classified as acknowledged instead of fixed.