SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Ubiquity
Prepared by:	Sherlock
Lead Security Expert:	cergyk
Dates Audited:	January 2 - January 10, 2024
Prepared on:	February 28, 2024

Introduction

The Metaverse Bank.

Scope

Repository: ubiquity/ubiquity-dollar

Branch: development

Commit: 2d1cfeb7178481138e820a8f22405ddaff6e4975

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
4	0

Issues not fixed or acknowledged

Medium	High
0	0

Issue M-1: LibUbiquityPool::mintDollar/redeemDollar reliance on outdated TWAP oracle may be inefficient for preventing depeg

Source: https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13

Found by

0xadrii, cergyk, osmanozdemir1, rvierdiiev

Summary

The ubiquity pool used for minting/burning uAD relies on a twap oracle which can be outdated because the underlying metapool is not updated when calling the ubiquity pool. This would mean that minting/burning will be enabled based on an outdated state when it should have been reverted and inversely

Vulnerability Detail

We can see that LibTWAPOracle has an update function to keep its values up to date according to the underlying metapool:
https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibTWAPOracle.sol#L61-L102

And that this function is called when minting/burning uADs:
https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L344

https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L416

But the function update is not called on the underlying metapool, so current values fetched for it may be stale:
https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibTWAPOracle.sol#L134-L136

Impact

A malicious user can use this to mint/burn heavily in order to depeg the coin further

Code Snippet

Tool used

Manual Review

Recommendation

Call the function:

On the underlying metapool the twap is based on, with only zero values, to ensure that the values of the pool are up to date when consulted

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

gitcoindev

> 1 comment(s) were left on this issue during the judging contest.
>
> **auditsea** commented:
>
> > The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

Shouldn't the issue be marked with 'Sponsor disputed' label then if it is invalid? @rndquu @pavlovcik @molecula451

pavlovcik

It probably makes more sense to ask @auditsea (not sure if this is the corresponding GitHub handle.)

rndquu

> A malicious user can use this to mint/burn heavily in order to depeg the coin further

Economically it doesn't make sense for a malicious user to do so. User is incentivised to arbitrage Dollar tokens hence even when metapool price is stale and a huge trade happens (i.e. a huge amount of Dollar tokens is minted in the Ubiquity pool) all secondary markets (uniswap, curve, etc...) will be on parity (in terms of Dollar-ANY_STABLECLON pair) after some time hence arbitrager will have to call `_update()` (when adding or removing liquidity) in the curve's metapool to make a profit.

Meanwhile I agree that the metapool's price can be stale but I don't understand how exactly minting "too many" Dollar tokens (keeping in mind that we get collateral in return) or burning "too many" Dollar tokens (keeping in mind that it reduces the `Dollar/USD` quote) may harm the protocol.

Anyway fresh data is better than stale one and the fix looks like a one liner so perhaps it makes sense to implement it. So here we could remove liquidity from the curve's metapool with 0 values (as mentioned in the "recommendation" section) which updates cumulative balances under the hood.

@gitcoindev @molecula451 What do you think?

gitcoindev

> > A malicious user can use this to mint/burn heavily in order to depeg the coin further
>
> Economically it doesn't make sense for a malicious user to do so. User is incentivised to arbitrage Dollar tokens hence even when metapool price is stale and a huge trade happens (i.e. a huge amount of Dollar tokens is minted in the Ubiquity pool) all secondary markets (uniswap, curve, etc...) will be on parity (in terms of Dollar-ANY_STABLECLON pair) after some time hence arbitrager will have to call `_update()` (when adding or removing liquidity) in the curve's metapool to make a profit.
>
> Meanwhile I agree that the metapool's price can be stale but I don't understand how exactly minting "too many" Dollar tokens (keeping in mind that we get collateral in return) or burning "too many" Dollar tokens (keeping in mind that it reduces the `Dollar/USD` quote) may harm the protocol.
>
> Anyway fresh data is better than stale one and the fix looks like a one liner so perhaps it makes sense to implement it. So here we could

> remove liquidity from the curve's metapool with 0 values (as mentioned in the "recommendation" section) which updates cumulative balances under the hood.
>
> @gitcoindev @molecula451 What do you think?

I have the same impression. Since this is easy to remediate with updating of cumulative balances I agree we could accept and implement it there.

osmanozdemir1

Escalate

Some duplicates of this issue fail to explain the actual root cause of it, which is internal update function being called in the beginning of every action in the underlying metapool.

Only valid duplicates of this issue are #68, #134, and #181. They all explain the core problem about curve metapool updating mechanism and provide recommendations regarding how to update underlying pool to solve the problem.

Issues #34, #84, #92 and #187 are more related to `consult()` function returning stale value. Some of them mention general things about TWAP *(like how low volume pools affect TWAP etc)*, and provide vague explanations. However, none of them pinpoints the actual cause of this issue. They don't mention underlying metapool's updating mechanism and should not be considered as valid duplicates without finding the root cause.

Last 4 issues might be considered as separate group of valid issues or they might be invalidated depending on the judge's preference, but they should not be duplicates of this one.

Kind regards.

sherlock-admin2

> Escalate
>
> Some duplicates of this issue fail to explain the actual root cause of it, which is internal update function being called in the beginning of every action in the underlying metapool.
>
> Only valid duplicates of this issue are #68, #134, and #181. They all explain the core problem about curve metapool updating mechanism and provide recommendations regarding how to update underlying pool to solve the problem.
>
> Issues #34, #84, #92 and #187 are more related to `consult()` function returning stale value. Some of them mention general things about TWAP *(like how low volume pools affect TWAP etc)*, and provide vague explanations. However, none of them pinpoints the actual cause of this

> issue. They don't mention underlying metapool's updating mechanism and should not be considered as valid duplicates without finding the root cause.
>
> Last 4 issues might be considered as separate group of valid issues or they might be invalidated depending on the judge's preference, but they should not be duplicates of this one.
>
> Kind regards.

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.

nevillehuang

@osmanozdemir1 I have internally discussed this with LSW and initially did deduplicated them as you mentioned. However, the watsons are technically not wrong per say by indicating an update required by using a stale time interval, so I decided to duplicate them.

osmanozdemir1

@nevillehuang Thanks for the response.

I totally agree with you about watsons being technically not wrong. However, this issue is one step deeper than a regular stale price issue. Even **non-stale** prices *(in terms of time interval)* will be incorrect because of this issue.

For example let's say staleness threshold is 4 hours:

```
  /*           T-4 hours          T-3 hours
↪    T0 current

↪   |-----------------|---------------------------------------------------|
                              Latest action
↪   The time Ubiquity
                             in curve metapool
↪   updates & checks the price


↪   |----------------------------------------------------|
                                     Latest actions impact until now is not
↪   accounted
*/
```

In an example above, staleness interval check will not revert because it is inside the staleness threshold. However, the latest action's impact on the price is never accounted. If it is a large swap or deposit, 3 hours worth of impact of this action will be huge.

Staleness check is not a solution for this problem. Only solution is somehow invoking the internal update function of the underlying curve metapool. And only issues I mentioned above explains this problem.

Therefore, I strongly believe those other 4 issues regarding staleness check are valid issues but separate ones.

Kind regards.

0xLogos

Escalate

Imho staleness part should be low and another should be invalid because of this

> > A malicious user can use this to mint/burn heavily in order to depeg the coin further
>
> Economically it doesn't make sense for a malicious user to do so. User is incentivised to arbitrage Dollar tokens hence even when metapool price is stale and a huge trade happens (i.e. a huge amount of Dollar tokens is minted in the Ubiquity pool) all secondary markets (uniswap, curve, etc...) will be on parity (in terms of Dollar-ANY_STABLECLON pair) after some time hence arbitrager will have to call `_update()` (when adding or removing liquidity) in the curve's metapool to make a profit.
>
> Meanwhile I agree that the metapool's price can be stale but I don't understand how exactly minting "too many" Dollar tokens (keeping in mind that we get collateral in return) or burning "too many" Dollar tokens (keeping in mind that it reduces the `Dollar/USD` quote) may harm the protocol.

I would also like to ask for a specific numerical example. From my understanding check against TWAP not for prevent depeg but for...

> prevent unnecessary redemptions that could adversely affect the Dollar price

i.e. for cases like redeeming uD when it's price above peg 1$ which is simply unprofitable because within the protocol it's still 1$.

sherlock-admin2

> Escalate

> Imho staleness part should be low and another should be invalid because of this
>
> > > A malicious user can use this to mint/burn heavily in order to depeg the coin further
> >
> > Economically it doesn't make sense for a malicious user to do so. User is incentivised to arbitrage Dollar tokens hence even when metapool price is stale and a huge trade happens (i.e. a huge amount of Dollar tokens is minted in the Ubiquity pool) all secondary markets (uniswap, curve, etc...) will be on parity (in terms of Dollar-ANY_STABLECLON pair) after some time hence arbitrager will have to call `_update()` (when adding or removing liquidity) in the curve's metapool to make a profit.
> >
> > Meanwhile I agree that the metapool's price can be stale but I don't understand how exactly minting "too many" Dollar tokens (keeping in mind that we get collateral in return) or burning "too many" Dollar tokens (keeping in mind that it reduces the `Dollar/USD` quote) may harm the protocol.
>
> I would also like to ask for a specific numerical example. From my understanding check against TWAP not for prevent depeg but for...
>
> > prevent unnecessary redemptions that could adversely affect the Dollar price
>
> i.e. for cases like redeeming uD when it's price above peg 1$ which is simply unprofitable because within the protocol it's still 1$.

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.

nevillehuang

I agree with @osmanozdemir1 analysis. I think this issues can possibly be separated. However, what does a staleness check actually do, isn't it to invoke that particular underlying update function in the first place (which is why I validated them)?

osmanozdemir1

> I agree with @osmanozdemir1 analysis. I think this issues can possibly be separated. However, what does a staleness check actually do, isn't it to invoke that particular underlying update function in the first place (which is why I validated them)?

Staleness check might revert or update underlying pool depending on how the protocol implements it. However, even if staleness check updates the underlying pool, it would only do that in case of the last action is outside of the staleness threshold. It won't update the price all the time.

Staleness check won't update the price if it **thinks the price is not stale**. The thing I was trying to point out here is that the underlying curve pool update function must be invoked all the time regardless of the staleness.

0xLogos

#181 is nice explanation, but i believe it lacks severe impact

> TWAP prices are incorrect.

It is view function used only to check thresholds in mint/redeem

> Incorrect amounts of tokens will be minted.

Mint/redeem amounts are based on chainlink oracle, not TWAP

> DollarToken's may be minted or redeemed outside of the price threshold.

It can happen, but there's no incentives to do it. Thresholds allow to mint/redeem uAD if it's actually profitable according to TWAP. If allowance is false-posive mint/redeem will not be profitable, if it is false-negative arb can themself update curve pool and thus update TWAP.

So IMO impact here is that there is possible situation when arb is forced to update curve pool themself in order to execute arb. But is this alone medium? And how often these situations will occur?

osmanozdemir1

Hi @0xLogos. Thanks for your comment.

> It is view function used only to check thresholds in mint/redeem

Indeed it is used to check thresholds in mint/redeem. This protocol has certain threshold prices and issue #181 explains how tokens can be minted outside of these threshold. It is clearly broken core functionality.

> It can happen, but there's no incentives to do it.

I can interpret this sentence in another way. If there is no incentive to do it and if it is not profitable for arbitragers etc, just a regular user may get harmed. A normal user doesn't have to track all these things and doesn't need to check Curve prices since the user expects protocol to revert outside of these thresholds.

minting/redeeming uAD above 1.01 and below 0.99 is the central feature of this protocol and in my opinion it is quite clear that minting/redeeming outside of the pre-determined thresholds is a broken functionality.

CergyK

Manipulations of the TWAP oracle have a high impact because they can cause insolvency on the Pool.

Indeed any redeeming/minting is done at the price indicated by chainlink. So what if the TWAP oracle is not aligned with the chainlink value? A malicious user which already has a large amount of uAD can drain the pool of the collateral

0xLogos

> just a regular user may get harmed

User mistake

I believe that "core functionality" in judging docs refers to something with obvious impact and I don't see it here, but it's for judges to decide.

> A malicious user which already has a large amount of uAD can drain the pool of the collateral

Don't see why it's malicious, uAD exchanged for collateral according to chainlink (fair) price, not outdated TWAP. Burning large amount of uAD will raise demand and incentivises to mint uAD for collateral and replenish pool - intended behavior.

osmanozdemir1

> User mistake

I disagree with that. User expects the protocol act like it is stated in the docs. User is not doing something wrong or giving incorrect input. For example, with your argument, every sandwich attack would be a user mistake due to not using private mempool or not being cautious enough but we consider it as lack of slippage protection.

Whole mint/redeem can be done outside of the intended thresholds with incorrect price assumptions. I don't understand how more core you want but will leave it to the judge.

nevillehuang

@Czar102 See issue #181 for a more complete issue breakdown with more thorough impact analysis (also the report I selected). I believe this is a valid medium severity issue.

0xLogos

Sandwitch attack is different. There is attacker, profit for attacker and loss for user. Here user themself making suboptimal choice, but again, he not loosing anything. Also this easier for user to prevent than sandwithcing.

Czar102

I agree with the deduplication proposed – issues #34, #84, #92 and #187 aren't describing this issue to a sufficient extent.

I was planning to consider this a low (it's a low/medium borderline issue imo) because I thought there is just protocol revenue lost due to suboptimal pricing.

> Indeed any redeeming/minting is done at the price indicated by chainlink. So what if the TWAP oracle is not aligned with the chainlink value? A malicious user which already has a large amount of uAD can drain the pool of the collateral

@CergyK than you for this comment, this seems to be the case. If the protocol had a single oracle (maybe chainlink and TWAP integrated), there would no issues related to this discrepancy. If there was a single oracle, the uAD would be *verifiably solvent* (given that the thresholds are configured correctly), but right now, due to the possibilities of these discrepancies, there is no such a *guarantee*. Tagging @pavlovcik @gitcoindev @molecula451 @rndquu, maybe it will be a valuable modification. Would also like @CergyK to sign off on this reasoning.

In real market scenarios, if the price has been radically changed, there is some market activity that will update the TWAP, so the probability of this issue having real impact is negligible, and this issue will usually have some impact on the price, nevertheless negligible. Hence, I think Medium severity is perfect for this issue.

Planning to separate and invalidate #34, #84, #92 and #187 and leave other duplicates and this issue as they are.

0xLogos

Sorry, i think i've lost the point.

> So what if the TWAP oracle is not aligned with the chainlink value?

@CergyK TWAP is for uAD but chainlink for collateral. What do you mean they are not aligned?

@Czar102 What verifiably solvent means? User with sufficient uAD can redeem and get all collateral when uAD fresh TWAP < 0.99. Why is't bad?

Czar102

> User with sufficient uAD can redeem and get all collateral when uAD fresh TWAP < 0.99. Why is't bad?

@0xLogos If the Chainlink oracle displays price of uAD of $1.02 and the TWAP is $0.99, thenonecanredeemandtheywillgetmoreforredeems($1.02) than pay for some mints ($1.01).

If one oracle was used for checking thresholds and exchange rates, it would be impossible to mint for a smaller cost than profit from redeeming.

CergyK

> Would also like @CergyK to sign off on this reasoning.

Agreed, as an additional remediation to the individual TWAP issues, it would be reasonable to introduce a deviation check between the two oracles

0xArz

@Czar102 The chainlink oracle is not used for uAD, its used for the collateral. There is no uAD chainlink feed so the curve pool twap is used by checking the reserves and the price of that the twap returns does not determine the amount of the collateral tokens that the user receives, it just checks the thresholds

0xLogos

@Czar102 But chainlink oracle not used for uAD pricing, there's no such feed. uAD price hardcoded to 1$ in the pool, only collateral chainlink feed used for calculating exchange rate assuming uAD worth 1$.

CergyK

@0xArz @0xLogos

By giving the right to users to mint/redeem collateral at the price given by the feed collateral/USD, the price of uAD is actually defined by this feed, so we can safely consider that the feed is also the real price collateral/uAD.

Even though I agree with your point that the chainlink feed is not technically a uAD feed.

0xArz

> @0xArz @0xLogos
>
> By giving the right to users to mint/redeem collateral at the price given by the feed collateral/USD, the price of uAD is actually defined by this feed, so we can safely consider that the feed is also the real price collateral/uAD.
>
> Even though I agree with your point that the chainlink feed is not technically a uAD feed.

Its an exchange rate that will always be 1 if the collateral is pegged to $1.00 and you will only receive more or less if the collateral price changes not the uAD price. The impact here is that it might be possible to mint/redeem outside of the thresholds, i dont get how you can “drain” the collateral.

osmanozdemir1

In the simplest way:

```
If A happens
      do B
```

You are trying to invalidate this issue by saying the price calculation is correct while performing `do B`.

The bug is in `if A happens`. This function shouldn't be performing `do B` at all. It doesn't matter `do B` is correct or not. The function should not be in that if statement in the first place. That if statement is the most central part of this protocol. It is the decision to mint or not mint this stable token. It is the decision to burn or not burn.

I am genuinely surprised that we are still discussing the validity of a bug that directly impacts the total supply and mint/redeem decision of a stable token. uAD is a stable coin. It has certain rules to mint/burn. That rule is broken.

0xArz

@osmanozdemir1 Your report describes this issue really well although this comment https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13#issuecomment-1945710223 about the impact is wrong.

> uAD is a stable coin. It has certain rules to mint/burn. That rule is broken.

This is true but this still doesnt affect the price of uAD until its sold right? If the reserves were 99 and 101 then everyone is able to mint, the arbitrage bot would only need ~1 uAD to rebalance to the pool but what if an attacker just mints the max(50k uAD), isnt this the same problem? Although this rule is broken, you are not stealing anything, the protocol doesnt become insolvent or anything. Would be great if we could confirm with the sponsor what problem could this cause

osmanozdemir1

Just an example in terms of what problem could this cause:

Let's assume TWAP price is 1.011 but the real price is 0.99.

- Normally, the protocol should prevent minting and allow redeeming to maintain the peg. This way token supply will decrease and price will move towards to 1.
- But because of the TWAP price is 1.01, it will allow minting more uAD instead of forbidding mint and allowing redeem. It will do the complete opposite, which will increase the total supply more instead of decreasing it. Which will decrease the real price more.

Decision to mint/burn and increasing/decreasing token supply are most crucial things in a stable coin and these crucial actions are performed based on mint/redeem thresholds in this protocol. Possibility to mint/redeem outside of these thresholds is not an innocent issue.

These thresholds will use outdated TWAP prices **nearly all the time** due to this issue because curve metapools are not highly active pools. They are not like uniswap pools. I provided both previous ubiquity metapool address and `lusd` metapool address in my issue if you want to check. **There are only few exchange actions in weeks**. That's why it is quite possible for this protocol to mint uAD instead of burn them or vice versa. People will keep minting until there is an external action in the underlying metapool that updates the TWAP, and that update might take too much time. This protocol should not wait an external action but it must update the underlying pool itself.

0xArz

> - But because of the TWAP price is 1.01, it will allow minting more uAD instead of forbidding mint and allowing redeem. It will do the complete opposite, which will increase the total supply more instead of decreasing it. Which will decrease the real price more.

Minting the token will not decrease the price, minting and then selling the token in the curve pool is what will decrease the price, because this isnt profitable, no one would do this.

If the real price was $0.99 then an arbitrage bot will buy uAD at a discounted price and then redeem and the pool will be updated because he just bought.

> Let's assume TWAP price is 1.011

But the lookback window of the twap is 1 block so doesnt it just return the spot price and then use the current reserves? In that case if the price is 1.011 then an arbitrage bot will mint and sell right after the transaction that made the price 1.011. Described here https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/20

osmanozdemir1

> If the real price was $0.99 then an arbitrage bot will buy uAD at a discounted price and then redeem and the pool will be updated because he just bought.

The real price I meant here is the real TWAP price after `_update` during an exchange, which can not be known without calling an exchange action in the underlying metapool. Arbitrage bot can't know it since there is no feed that actively tracks real price. That's the whole point of the Curve metapool calling the `_update` function as first thing during any kind of exchange and this way all exchanges are done with the most updated TWAP prices.

> But the lookback window of the twap is 1 block so doesnt it just return the spot price and then use the current reserves?

Firstly, no it doesn't have to be 1 block. It is the difference between current timestamp and the last updated timestamp. #20 explains that it will be only 1 block if you update it in consecutive blocks.

Secondly, no it doesn't return the spot price and that's whole point of my submission. It uses `currentCumulativePrices` function, which returns latest updated prices. If underlying metapool doesn't have any new action, it will return the same cumulative price and the same timestamp even though if you call it now, or 3 hours later, or 1 day later. It directly returns the cumulative price and the timestamp at the point of the latest action in the underlying metapool. You can read the implementation code here: https://etherscan.io/address/0x5f890841f657d90e081babdb532a05996af79fe6#code

0xArz

@osmanozdemir1 I see yeah, i somehow thought that the update is done after the tx. Not sure of the impact but ig medium is appropriate for using an incorrect value to check the thresholds.

The comment by @Czar102 about the oracles is still incorrect tho. The impact of this issue is that most of the time it will use outdated twap price which can block or allow minting/redeeming. In https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/56 you would need a lot of funds just to allow/block minting and redeeming for a small amount of time which is not a problem unlike here where most of the time the wrong price will be used which will allow/block minting and redeeming

0xLogos

> But because of the TWAP price is 1.01, it will allow minting more uAD instead of forbidding mint and allowing redeem. It will do the complete opposite, which will increase the total supply more instead of decreasing it. Which will decrease the real price more.

Allowing minting won't increase total supply. No one would mint just because it's allowed when it will cause losses. But one can trigger curve `_update` thus update TWAP and then redeem for profit. uAD is algotithmic stable coin and the market is assumed to operate efficiently. If bad actor wants to grief, arb will drain his funds.

> Let's assume TWAP price is 1.011 but the real price is 0.99.

Just bcz TWAP is stale you can't assume whatever you want. What's probability of this state? I think such curve pools indeed can be stale for long time, but only when they balanced.

Sorry, I really do not want to argue, just for @Czar102 to understand and answer my points.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/ubiquity/ubiquity-dollar/pull/893.

Czar102

After extensive discussions with the LSW and the Lead Judge, planning to resolve the escalation as previously intended (https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13#issuecomment-1945710223), as the issue doesn't only cause a DoS, but may allow for an easy value extraction strategy from the protocol, similar to the one described in #17 and duplicates.

Planning to accept the first and reject the second escalation.

0xLogos

From #17

> Chainlink price may be slightly outdated with regards to actual Dex state, and in that case users holding a depegging asset (let's consider DAI depegging) will use uAD to swap for the still pegged collateral: LUSD

You can't swap because allowed only either mint or redeem.

Also if pool works as expected it always lose value on mint/redeem, but it takes fee to mitigate this (btw this was my escalation point for #36 that fee actually used). Assume peg within desired range: 1.009, but mint is still open because twap is stale and returns 1.011, now you can mint 1 uAD = 1.009$ for 1$. But you pay same fee as usual. And usual mint expected to be more profitable and pool takes according fees. And with that fee mint is barely profitable.

This is just my thoughts about the strategy and honestly I can't think of anything profitable here.

@CergyK @nevillehuang But if there really is profitable strategy I want to learn it!

0xArz

@Czar102 @nevillehuang @CergyK https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/56 is not a valid issue. You will not be able to extract anything because it DoSes the redeem, not enables. The report also clearly mentions that. You can only make the price of uAD increase because it is impossible for the attacker to have a large amount of uAD if the only way to get it from is the curve pool, he can only make the price increase by providing a large amount of 3CRV.

I really dont undestand why we are trying to make https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/17 valid again which is a completely seperate issue from this one. As @0xLogos mentioned you are only able to mint or redeem so you cant sandwich the oracle update and it is not profitable.

> easy value extraction strategy

Can you also please tell me since when is proposing 2 consecutive blocks considered easy?

osmanozdemir1

> @Czar102 @nevillehuang @CergyK #56 is not a valid issue. You will not be able to extract anything because it DoSes the redeem, not enables. The report also clearly mentions that. You can only make the price of uAD increase because it is impossible for the attacker to have a large amount of uAD if the only way to get it from is the curve pool, he can only make the price increase by providing a large amount of 3CRV.
>
> I really dont undestand why we are trying to make #17 valid again which is a completely seperate issue from this one. As @0xLogos mentioned you are only able to mint or redeem so you cant sandwich the oracle update and it is not profitable.
>
> > easy value extraction strategy
>
> Can you also please tell me since when is proposing 2 consecutive blocks considered easy?

I think this comment is under wrong issue and nothing to do with this one.

nevillehuang

@0xLogos @0xArz

- For #17, you can see from public knowledge that fees are intended to be zero, hence leakage is possible. Additionally, even if fees are implemented, there is no guarantees that it is sufficient to cover large price fluctuations as noted here
- Proposing 2 consecutive blocks is not easy, but is a valid external vector to bypass thresholds to alternate between mint/redemption thresholds. It is possible for attacker to have externally owned uAD bought from secondary markets/minted from the protocol. Additionally, the TWAP price is influenced and reflected inaccurately since reserves are skewed similar to in #59, albeit a different cause.

0xArz

@nevillehuang The only way to obtain uAD is from the curve pool so you realistically cant have a huge amount like with 3crv. Can you also describe the whole attack path including the "profit" that the attacker gets combining https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/17 and https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/56?

nevillehuang

@0xArz I believe this is untrue based on your comment here and this statement here by sponsor. I think no further explanations is required from my end if not this discussion will be endless, better leave it to head of judging.

0xArz

Fine but they still have to get this uAD from somewhere but nvm. I just want you to realize how hard it is to perform an attack like this and looks like you cant even give me the attack path due to how hard the attack is. If you want to mint and then redeem you would have to manipulate the twap 2 times because as we mentioned you can either mint or redeem. This means that you would have to propose 2 consecutive blocks 2 times, if thats that easy is a 51% attack also a valid external vector to bypass thresholds?

I really dont want to argue but im tired of having to fight back because this issue is getting validated just because you can "manipulate a twap" without actually stating the profit that the attacker gets, the extremely low circumstances and the huge amount of resources that the attacker needs.

Talking about https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/17 or https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/56 here not https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13 which is valid i believe.

Czar102

The outline of the attack, some details may be wrong since I haven't audited the code myself:

1. Price of uAD was $0.99.
2. There was a trade that put the price at $1.01. No more trades are made.
3. The price of LUSD reported by an oracle is $1.
4. The real price of LUSD drifted to $0.99 and a few minutes pass.
5. The attacker deposits 1m LUSD to get 1m uAD.
6. The attacker triggers an update in the metapool. Roughly at the same time, the Chainlink oracle heartbeat is triggered to change the LUSD price to $0.99.
7. The TWAP is $1.01nowandtheattackercanredeem1muADforroughly1.01muADtomakea$ 10k profit.

Will close this escalation today.

0xArz

@Czar102 This probably makes sense for this issue https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13 even though the likelihood is low but can you please describe the whole attack path in https://github.com/sherlock-audit/20

23-12-ubiquity-judging/issues/56 where you have to propose 2 consecutive blocks 2 times? Really curious to see how you would do that!

Czar102

When one of deposits/redeems are available, it's sufficient to propose 2 consecutive blocks only once.

Also, simply providing liquidity after a rapid price change will help changing the operation permitted between mints and redeems.

0xArz

In that case the price of uAD has to be 1.01$ so the attacker is able to mint first. If you really think that all of these preconditions and the amount of resources that the attacker needs just to hedge a small amount of collateral is an easy value extraction strategy then i really dont know but looks like there is no point in continuing to explain why this issue is invalid so you can decide whatever you want.

Czar102

Result: Medium Has duplicates

@0xArz There are some preconditions, but the attack is possible and Medium severity is created for issues where loss of funds is limited, potentially due to many constraints.

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- osmanozdemir1: accepted
- 0xLogos: rejected

sherlock-admin

The Lead Senior Watson signed off on the fix.

Issue M-2: UbiquityPool::mintDollar/redeemDollar collateral depeg will encourage using UbiquityPool to swap for better collateral

Source: https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/17

The protocol has acknowledged this issue.

Found by

cducrest-brainbot, cergyk, fugazzi, ge6a, shaka

Summary

In the case of a depeg of an underlying collateral, UbiquityPool mechanism incentivises users to fill it up with the depegging collateral and taking out the better collateral. This means uAD ultimately depegs as well.

Vulnerability Detail

Chainlink price may be slightly outdated with regards to actual Dex state, and in that case users holding a depegging asset (let's consider `DAI` depegging) will use uAD to swap for the still pegged collateral: `LUSD`. By doing that they expect a better execution than on Dexes, because they swap at the price of chainlink minus the uAD fee: https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L358-L364

This in turn fills the reserves of UbiquityPool with the depegging collateral and depletes the reserves of `good` collateral.

Impact

A depegging collateral will cause uAD to depeg because users are incentivised to use the pool to swap for the `better` asset

Code Snippet

Tool used

Manual Review

Recommendation

Multiple solutions may be studied:

- Enforce a ratio between different collateral reserves (somewhat like GMX pricing algo which also enables users to swap with zero slippage using chainlink feeds)
- Use a safety minting ratio (LTV mechanism similar to borrowing protocols)
- Force chainlink feeds to stay within acceptable thresholds for stable coins (revert operations on collateral if price is out of range)

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about the protocol insolvency in case of collateral depeg. It's not avoidable, that's why the protocol has borrowing function to get yield, take fees on mint and redeem, these features will hedge the risk from protocol insolvency

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about the protocol insolvency in case of collateral depeg. It's not avoidable, that's why the protocol has borrowing function to get yield, take fees on mint and redeem, these features will hedge the risk from protocol insolvency

gitcoindev

> 1 comment(s) were left on this issue during the judging contest.
>
> **auditsea** commented:
>
> > The issue describes about the protocol insolvency in case of collateral depeg. It's not avoidable, that's why the protocol has borrowing function to get yield, take fees on mint and redeem, these features will hedge the risk from protocol insolvency

Should we also add 'Sponsor Disputed' label in this issue as well? @rndquu @pavlovcik @molecula451

pavlovcik

It probably makes more sense to ask @auditsea (not sure if this is the corresponding GitHub handle.)

rndquu

As far as I understand this issue describes the following scenario:

1. User1 mints 100 Dollar tokens and provides 100 DAI collateral
2. User2 mints 100 Dollar tokens and provides 100 LUSD collateral
3. DAI depegs
4. User1 (who initially deposited DAI) redeems 100 Dollar tokens for 100 LUSD
5. User2 (who initially deposited LUSD) is left only with depegged DAI pool

If DAI depegs then the Dollar token will also depeg mainly because DAI is used as an underlying collateral in the `Dollar-3CRVLP` curve's metapool. We shouldn't limit users in redeeming (i.e. burning) Dollar tokens anyhow because Dollar redeems bring back the `Dollar/USD` quote to `$1.00` peg. So it seems to be fine that users should be able to burn Dollars until the pools are empty no matter where they initially deposited to because this is the only way to maintain the Dollar token's USD peg.

In case of a collateral depeg the only way to hedge Dollar depeg to some extent is to acquire fees and yield from AMO minters. This is not a 100% guarantee but I guess that if chainlink works fine (i.e. provides not too stale data) and we have 5% overcollateralization (from fees and yield) the Dollar token should not depeg too much.

> Force chainlink feeds to stay within acceptable thresholds for stable coins (revert operations on collateral if price is out of range)

I don't understand how reverting on minting and redeeming helps. Reverting in this case means acquiring bad debt since all operations are paused and abritragers are not able to bring the Dollar token back to the USD peg by burning Dollars which makes the Dollar token to depeg with greater force.

pavlovcik

@rndquu as a heads up we should only start with accepting `LUSD` and then maybe, eventually, add `sUSD` next in case of future liquidity issues. `DAI` isn't part of the plan yet.

rndquu

Initially we plan to use only `LUSD` as collateral so there won't be the case with bad (i.e. depegging) and good collateral.

I think we should:

1. Create a separate issue for this one in our repo and fix it later since it is not critical

2. Mark the current issue as valid and "won't fix"

@molecula451 @gitcoindev Help

0x3agle

> In the case of a depeg of an underlying collateral, UbiquityPool mechanism incentivises users to fill it up with the depegging collateral and taking out the better collateral.

- If the uAD price is stable at $1 in the uAD-3CRV pool, then the mints (open if `uAD > $1.01`) or redeems (open if `uAD < 0.99`) won't be enabled.
- The chainlink oracle - is used to get the price of collateral. This price is then used to determine the amount of uAD to mint or burn, proportional to the collateral
- The TWAP price from the curve pool will decide whether to enable mints or enable redeems
- Hence the collateral de-pegging doesn't affect the uAD price, because we get the uAD price from the Curve pool (ref)

molecula451

The above sceneario makes much more sense on the current issue, a fix won't happen, this is most likely an invalid

0xLogos

Escalate Should be invalid Reverting redemption if one of the collateral depegs even greater evil because now no one can redeem uD as noticed by rndquu. I think collateral depeg is a very unpleasant event and you can’t simply get away with it without loss. Enforcing a ratio between different collateral reserves is not good solution as it has bad consequences for market efficiency throughout the life of the protocol and will barely (arguably) mitigate losses in case of depeg.

sherlock-admin2

> Escalate Should be invalid Reverting redemption if one of the collateral depegs even greater evil because now no one can redeem uD as noticed by rndquu. I think collateral depeg is a very unpleasant event and you can’t simply get away with it without loss. Enforcing a ratio between different collateral reserves is not good solution as it has bad consequences for market efficiency throughout the life of the protocol and will barely (arguably) mitigate losses in case of depeg.

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.

evmboi32

In my issue #144, I described that the pool can run out of collateral if the collateral gets de-pegged. This means that users cannot redeem their uAD tokens for the underlying collateral.

pavlovcik

I really feel like collateral performance is out of scope for the audit. For whatever its worth, the reason why we are starting with LUSD is because it seems to have the least points of failure out of all the stablecoins I'm aware of. The tradeoff is its slight volatility and limited ability to scale.

To me it comes across that you're saying that we need to include Liquity's entire protocol within our audit scope in order to confirm that a depeg isn't possible, and that our protocol will not fail.

If this is in scope, why not proceed with Ethereum blockchain failures? If the network gets taken over, then fraudulent transactions can be generated to withdraw all of our collateral, rendering the protocol insolvent.

etc

rndquu

> I really feel like collateral performance is out of scope for the audit. For whatever its worth, the reason why we are starting with LUSD is because it seems to have the least points of failure out of all the stablecoins I'm aware of. The tradeoff is its slight volatility and limited ability to scale.
>
> To me it comes across that you're saying that we need to include Liquity's entire protocol within our audit scope in order to confirm that a depeg isn't possible, and that our protocol will not fail.
>
> If this is in scope, why not proceed with Ethereum blockchain failures? If the network gets taken over, then fraudulent transactions can be generated to withdraw all of our collateral, rendering the protocol insolvent.
>
> etc

Collateral depeg does harm the ubiquity protocol hence it is considered a valid issue (not sure what severity though)

0xLogos

> In my issue #144, I described that the pool can run out of collateral if the collateral gets de-pegged. This means that users cannot redeem their

uAD tokens for the underlying collateral.

@evmboi32 Ok, depeg happens, collateral's price is now out of min/max range and redemptions are now failing (proposed in #144 solution). Then what? Wait and hope the collateral to repeg?

nevillehuang

@CergyK Are you aware if Ubiquity has a circuit breaker for depeg events?

To me if they lack one, this will constitute as a valid medium severity finding as a depeg does directly undermine the protocols available collateral. This is in addition to the depeg scenario not being stated as an accepted risk by the protocol in the contest details (to my knowledge, most of the time, a stablecoin protocol would acknowledge the risks of a depeg scenario and even have a circuit breaker in place).

molecula451

no we don't have circuit breaker @nevillehuang

Czar102

The maximum divergence of the market price from the oracle feed should be within the fee charged. Planning to accept the escalation and invalidate the issue.

pavlovcik

We have our staking contract that allows us to manipulate single sided liquidity on our metapool to directly change the price right now.

Czar102

Result: Invalid Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: accepted

gstoyanovbg

@Czar102 I don't understand the argument for invalidating this report, could you explain a little more?

From my point of view, I submitted report #217 because when calculating the amount of collateral a user will receive when redeeming, the real price of the dollar token is not used, but it is assumed to always be $1. Let me give an example:

1. User A mints 120 dollar tokens for 120 DAI

2. User B mints 120 dollar tokens for 120 LUSD
3. DAI depegs with 20%. Now collateral to dollar tokens ratio is (120 + 120*0.8) / 240 = 0.9
4. User A redeems 100 dollar tokens for 120 DAI (because the price of DAI is $0.8) and 20 dollar tokens for 20 LUSD. After this operation, there are 120 dollar tokens in circulation and 100 LUSD collateral. Now the collateral to dollar tokens ratio is 100/120 = 0.83, less than the ratio at step 3. If there are more users, those who are last would not be able to get anything for their tokens because there will be no remaining collateral. This scenario is harmful to both the protocol and the users. I want to note that in redeemDollar() we only have upper bound for the dollar token price.

Wouldn't it be better to use the real price of the dollar token when determining the amount of collateral to be received in exchange for the dollar tokens? This way, each user will receive a proportional share of the available collateral, and there will not be a situation where there are dollar tokens in circulation without collateral behind them.

molecula451

> @Czar102 I don't understand the argument for invalidating this report, could you explain a little more?
>
> From my point of view, I submitted report #217 because when calculating the amount of collateral a user will receive when redeeming, the real price of the dollar token is not used, but it is assumed to always be $1. Let me give an example:
>
> 1. User A mints 120 dollar tokens for 120 DAI
> 2. User B mints 120 dollar tokens for 120 LUSD
> 3. DAI depegs with 20%. Now collateral to dollar tokens ratio is (120 + 120*0.8) / 240 = 0.9
> 4. User A redeems 100 dollar tokens for 120 DAI (because the price of DAI is $0.8) and 20 dollar tokens for 20 LUSD. After this operation, there are 120 dollar tokens in circulation and 100 LUSD collateral. Now the collateral to dollar tokens ratio is 100/120 = 0.83, less than the ratio at step 3. If there are more users, those who are last would not be able to get anything for their tokens because there will be no remaining collateral. This scenario is harmful to both the protocol and the users. I want to note that in redeemDollar() we only have upper bound for the dollar token price.
>
> Wouldn't it be better to use the real price of the dollar token when determining the amount of collateral to be received in exchange for the

> dollar tokens? This way, each user will receive a proportional share of the available collateral, and there will not be a situation where there are dollar tokens in circulation without collateral behind them.

we will leavy it as invalid

Czar102

@gstoyanovbg thank you for the comment. We had an extensive internal discussion about this issue and some of the other ones, considerations about the exact behavior you are describing. There is a chance we may revert this escalation's resolution, if we determine that this has been a misjudgment.

I would recommend looking at #60 as a core cause of this issue.

Czar102

Will consider this a valid Medium and change the escalation resolution status.

Issue M-3: LibUbiquityPool::mintDollar/redeemDollar reliance on arbitrarily short TWAP oracle may be inefficient for preventing depeg

Source: https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/20

Found by

0xLogos, Arz, Coinstein, Krace, b0g0, cducrest-brainbot, cergyk, evmboi32, infect3d, nirohgo, rvierdiiev, the-first-elder

Summary

The ubiquity pool used for minting/burning uAD relies on a twap oracle which can be outdated because the underlying metapool is not updated when calling the ubiquity pool. This would mean that minting/burning will be enabled based on an outdated state when it should have been reverted and inversely

Vulnerability Detail

We can see that `LibTWAPOracle::consult` computes the average price for uAD on the metapool vs 3CRV. However since it uses the duration since last update as a TWAP duration, it will always get only the value of price at the previous block it was updated at; https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibTWAPOracle.sol#L80

Let's consider the following example:

metapool initial state at block N: reserveA: 1000 reserveB: 1000

metapool state at block N+1 (+12 seconds): reserveA: 1500 reserveB: 500

if we have executed the update at each of these blocks, this means that if we consult the twap at block N+2, we have: ts.priceCumulativeLast: `[A, B]` priceCumulative: `[A+1500*12, B+500*12]` (reserve * time_elapsed); blockTimestamp - ts.pricesBlockTimestampLast = 12;

which means that when we call `get_twap_balances` the values returned are simply [1500, 500], which are the values of the previous block.

Impact

A malicious user which can control two successive blocks (it is relatively feasible since the merge), can put the twap in any state for the next block:

- Can Dos any minting/burning in the block after the ones he controls

- Can unblock minting/burning for the next block, and depeg uAD further

Code Snippet

Tool used

Manual Review

Recommendation

Use a fixed duration for the TWAP:

```
uint256[2] memory twapBalances = IMetaPool(ts.pool)
    .get_twap_balances(
        ts.priceCumulativeLast,
        priceCumulative,
        15 MINUTES
    );
```

Discussion

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

sherlock-admin2

1 comment(s) were left on this issue during the judging contest.

auditsea commented:

> The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

gitcoindev

> 1 comment(s) were left on this issue during the judging contest.

> **auditsea** commented:
>
>> The issue describes about TWAP can be manipulated because `update` function can be called anytime and by anyone, thus TWAP period can be as short as 1 block. It seems like a valid issue but after caeful consideration, it's noticed that the TWAP issue does not come from its period but the logic itself is incorrect, thus marking this as Invalid

Also invalid then and 'Sponsor Disputed' label should be added? @rndquu @pavlovcik @molecula451

pavlovcik

It probably makes more sense to ask @auditsea (not sure if this is the corresponding GitHub handle.)

rndquu

Yes, it is possible (especially in the early Dollar token stage when market activity is low) to skew the curve's TWAP value since our TWAP (in some cases) may simply take the latest block price thus TWAP can be manipulated with low effort.

This is not critical (i.e. medium severity seems to be valid) since TWAP's price is used only in `require()` statements on mint and redeem operations but must be fixed.

Not sure what's the best possible solution but the following similar issues are worth to check:

- https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/138
- https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/175

gitcoindev

@rndquu just to double check, the solution is to set a constant time window for the TWAP price.

A question to Sherlock: should the window be set to 15, 30 min or any other value?

pavlovcik

In my experience with software development, anything time based is better implemented to be event based. Although unfortunately I'm not sure what events would make a great substitute in this case. What if we checked that they aren't consecutive blocks?

rndquu

> @rndquu just to double check, the solution is to set a constant time window for the TWAP price.

> A question to Sherlock: should the window be set to 15, 30 min or any other value?

I think the solution is to use the latest curve's metapool with built-in TWAP oracle and adjustable time window as described here.

> the solution is to set a constant time window for the TWAP price

You're right, the solution is to increase it to 15 or 30 minutes.

rndquu

Since there is no direct loss of funds the "high" severity doesn't seem to be correct.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/ubiquity/ubiquity-dollar/pull/893.

sherlock-admin

The Lead Senior Watson signed off on the fix.

Issue M-4: LibTWAPOracle::update Providing large liquidity will manipulate TWAP, DOSing redeem of uADs

Source: https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/56

Found by

GatewayGuardians, KupiaSec, cducrest-brainbot, cergyk, evmboi32

Summary

The twap oracle used by ubiquity does not compute a TWAP (Time weighted average price) but an instant price based on time weighted average of balances, and thus is more vulnerable to manipulation by an actor temporarily allocating a large amount to the metapool.

Vulnerability Detail

We can see the `LibTWAPOracle::update()` calls on `IMetapool::get_twap_balances(uint256[2] memory _first_balances, uint256[2] memory _last_balances, uint256 _time_elapsed)` which gets the time-weighted average of the pool reserves given time_elapsed and cumulative balances.

Then the time-weighted average is deduced by using the average of reserves over the `time_elapsed`. However we can see that a user with a large amount of capital and controlling 2 consecutive blocks can either add large liquidity in an imbalanced way in block N, and remove in block N+1. In that case the `higher` reserves ratio will have a higher weight in the computation and will last for many blocks.

Scenario

1/ In block N-1:

Metapool contains 10 of uAD and 10 of 3CRV, the price is balanced, and withdrawals of uAD are accepted.

2/ In block N:

Alice updates both metapool twap and ubiquityPool twap.

Alice provides 100 uAD and 200 3CRV to the pool, now reserves are 110 uAD and 210 3CRV

3/ In block N+1:

Alice removes the previously added liquidity putting the reserves back to 10 uAD and 10 3CRV, but does not updates UbiquityPool right away.

4/ In block N+40:

Alice updates both Metapool and ubiquity TWAP oracle,

Let's see the value which is computed for the returned TWAP:

uAD cumulative reserve difference = 12*110+12*40*10 = 6120 3CRV cumulative reserve difference = 12*210+12*40*10 = 7320

We can see that the reserves are still very imbalanced even 40 blocks after the 2 blocks manipulation, blocking withdrawals, because uAD price versus 3CRV is too high: https://github.com/sherlock-audit/2023-12-ubiquity/blob/main/ubiquity-dollar/packages/contracts/src/dollar/libraries/LibUbiquityPool.sol#L418-L421

Impact

A malicious user with access to considerable capital and controlling two consecutive blocks (feasible since the merge), can DOS the withdraw functionality of UbiquityPool for many blocks.

Code Snippet

Tool used

Manual Review

Recommendation

Find a way to compute the twap of the price (as Uniswap v3 does), instead of using twap of balances as a proxy

Discussion

gitcoindev

@rndquu @pavlovcik @molecula451 what do you think about this one? Technically possible, but would likely require lots of staked ETH. I am wondering if this is just hypothetical and controlling 2 blocks would require funds of e.g. Coinbase size. If this is the case, then perhaps impact is high but probability is very low. There are currently ~900k ETH validators https://beaconcha.in/charts/validators with each staked at least 32 ETH.

pavlovcik

I spent some time looking through our old issues and discussions but unfortunately I couldn't find my writeup on this. Basically as I recall, after The Merge there was a research report that proved how a two-consecutive-block TWAP attack could be performed with a realistic amount of funds.

Unfortunately I don't remember the conclusion of how to mitigate.

rndquu

> uAD cumulative reserve difference = 12*110+124010 = 6120 3CRV*
> *cumulative reserve difference = 12*210+124010 = 7320

TWAP window here is ~8 minutes. If we take a 30 minutes TWAP window (suggested everywhere) we get `uAD cumulative reserves = 19320 & 3CRV cumulative reserves = 20520` which is much closer to a balanced pool.

Overall this seems to be a duplicate of https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/20 since increasing the TWAP window fixes the issue (to some extent).

rndquu

So, as far as I understand, the core issue is that the curve metapool uses accumulated balances instead of accumulated spot prices hence the TWAP is not as accurate as it could be.

I agree that it could be improved but disagree with the "high" severity.

0xLogos

Escalate Should be low. According to sherlock rules here.

> 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.

In this issue attacker would need a large amount of funds (exactly 15 times more than curve pool reserves according to example) and control over 2 consecutive block (feasible, but very difficult) just to dos withdrawals for short time (in example 40 blocks = 12*40/60 = 8 min).

sherlock-admin2

> Escalate Should be low. According to sherlock rules here.
>
> > 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.
>
> In this issue attacker would need a large amount of funds (exactly 15 times more than curve pool reserves according to example) and control over 2 consecutive block (feasible, but very difficult) just to dos withdrawals for short time (in example 40 blocks = 12*40/60 = 8 min).

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.

nevillehuang

@CergyK any comments?

0xLogos

I also want to note that the proposed solution is quite difficult to implement correctly and I believe that it is an acceptable risk to leave it as is

0x3agle

Is uAD TWAP manipulatable? short answer - Yes. Long answer - https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/72#issuecomment-1909410503 Pointing out that #212's main focus is on shorting the collateral at the cost of UbiquityPool. So, it should not be a duplicate of this issue

0xLogos

@0x3agle It is manipulatable because of #20 I think firstly this issue should be fixed (not so trivial as recommended in report) Also collateral tokens are "stealed" during arb

0x3agle

@0xLogos, even after addressing issue 20, the system remains susceptible to manipulation.

In a standard arbitrage situation, specific conditions enable arbitrageurs to profit.

However, #212 deviates from this. Here, the attacker can intentionally trigger redemptions to acquire collateral at a lower price by influencing the uAD price. This means the attacker can engineer favorable conditions, rather than relying on chance, effectively manipulating the protocol to their advantage.

0xLogos

> The attacker opts to swap 3,000 uAD for X amount of 3CRV in the Curve pool, strategically lowering the "spot price" of uAD.

from https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/72#issuecomment-1909410503

Doesn't look like manipulation

0x3agle

The attacker can lower the price to enable redeems as they can now get collateral for lower price. If this isn't manipulation, then what is it?

Edit: improved the wording in the comment.

0xArz

@0x3agle but why would you even need to manipulate the twap? I think your issue describes pretty much the same thing as https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/72 and the root cause is the same - collateral depegging. The attack will be possible without the TWAP manipulation(assuming the the price of uAD stays at $1.00)

Total Spent: 97,000 LUSD (for minting uAD) + ~$50 (Slippage for swapping twice [uAD to 3CRV then 3CRV to uAD]) Total Received: 101,000 LUSD Profit: ~3,800 LUSD

There is no actual profit here, because 97,000 LUSD at $1.03 is worth the same as 101,000 LUSD at $0.99

Also i believe the redeemPriceThreshold is $1.01 not $0.99, it wouldnt make sense to not allow users to redeem when the price is $1.00 right? Would be great if sponsor could confirm this

I think what you are saying is that you can manipulate the price so that redeems are opened but i believe that the redeemPriceThreshold is $1.01 so you dont have to manipulate anything but just wait when the collateral drops

0xLogos

> There is no actual profit here, because 97,000 LUSD at $1.03 is worth the same as 101,000 LUSD at $0.99

Good point

0x3agle

@0xArz

This is called as hedging. There is no profit but if the attacker held those 97,000 LUSD at $1.03 and when it drops to $0.99, the attacker loses 4%.

By applying this strategy, the attacker offsets the losses at the cost of protocol. Therefore, the attacker does not lose any value.

Regarding redeem threshold: you are saying that to enable redeem, the uAD should go up instead of down. Think about it: if uAD goes up to $1.01, there is more demand. We need the uAD to move down to $1. You can't control demand, you can control the supply.

What will you do to level the uAD back? - Mint. Increase the supply to match the demand. So, we want to enable mints when uAD goes up.

When uAD goes down to $0.99, the supply is more demand is less. Therefore we need to burn the uAD to match supply and demand. Hence redeem will be enabled if uAD goes below $0.99

0xArz

@0x3agle

```
require(
            getDollarPriceUsd() >= poolStorage.mintPriceThreshold,
            "Dollar price too low"
        );
```

So you are saying that the mintPriceThreshold is $1.01? How will then users be able to mint uAD when the curve pool was just created and the price is $1.00?

$0.99 for minting and $1.01 for redeeming makes more sense.

Same impact like in https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/72 btw

0x3agle

When the pool is first deployed, users can neither mint or redeem.

The demand for uAD arises. Users start buying it using 3CRV from curve pool.

So, eventually the twap will show an increased price. Let's say $1.02 Now, the mints on the pool will be unlocked. So users can technically mint more uAD for exact amount from Ubiquity Pool instead of 3CRV. This will lead to arbitrage, balancing the curve pool. So, now the pool will reflect the stable price of $1 for uAD.

How will redeems be unlocked? In case a lot if uAD is minted, users will swap it for 3CRV in the curve pool. Increasing the supply of uAD in curve pool. Therefore reducing the price of uAD, let's say $0.98

Now redeems will be enabled (mints are disabled) which will give you more collateral for same uAD. Again arbitrage, leading to curve pool balance. Stabilising price of uAD back to $1- disabling any mints or redeems

0xArz

hey @pavlovcik @gitcoindev @rndquu what are the correct thresholds that will be used? In the tests the mintPriceThreshold is set to $1.01 and the redeemPriceThreshold is set to $0.99 which means that when the price is $1.00 users will not be able to mint or redeem uAD. This doesnt really makes sense as users are supposed to be able to obtain uAD on a 1:1 ratio or is the UbiquityPool just going to be used to rebalance the price? Thank you in advance!

pavlovcik

> Also i believe the redeemPriceThreshold is $1.01 not $0.99, it wouldnt make sense to not allow users to redeem when the price is $1.00 right?

> Would be great if sponsor could confirm this

Minting above $1.00 and redemptions below $1.00. The threshold should be adjustable so for now `redeemPriceThreshold` should be $0.99. That way a user can purchase $0.99 "dollars" from the market and then redeem them for $1.00 worth of collateral. This stabilizes the price floor.

> which means that when the price is $1.00 users will not be able to mint or redeem uAD. This doesnt really makes sense as users are supposed to be able to obtain uAD on a 1:1 ratio or is the UbiquityPool just going to be used to rebalance the price? Thank you in advance!

That's a good question. We have an application to manage and incentivize distributed teams which offers incentives to settle payments in our Ubiquity Dollars. Assuming that there is some demand generated for our Ubiquity Dollars with our DevPool system, we should easily be able to move the price above the threshold.

If there is no demand, I think it makes sense for the protocol to not respond with incentives to change the supply.

CergyK

> @CergyK any comments?

It seems the escalation is based on the fact that a relatively short (?) duration was used in the example. This duration can be arbitrarily extended by using more capital. Overall this should not be happening in a TWAP, and for example in the TWAP as implemented in Uniswap, it is not possible to manipulate more efficiently by providing more liquidity during 2 blocks.

rndquu

> hey @pavlovcik @gitcoindev @rndquu what are the correct thresholds that will be used? In the tests the mintPriceThreshold is set to $1.01 and the redeemPriceThreshold is set to $0.99 which means that when the price is $1.00 users will not be able to mint or redeem uAD. This doesnt really makes sense as users are supposed to be able to obtain uAD on a 1:1 ratio or is the UbiquityPool just going to be used to rebalance the price? Thank you in advance!

Users may also get Dollar(`uAD`) tokens on secondary markets

0xLogos

> It seems the escalation is based on the fact that a relatively short (?) duration was used in the example. This duration can be arbitrarily extended by using more capital. Overall this should not be happening in a TWAP, and for example in the TWAP as implemented in Uniswap, it is not possible to manipulate more efficiently by providing more liquidity during 2 blocks.

@pavlovcik @rndquu First of all we need to know max aceptable by protocol team time that pool can be DOSed taking into account that admin can in any time prevent DOS by adjusting threshholds. Based on example you need 100K pool liquidity * 15 = 1.5M$ AND control over 2 block to DOS for 8 min (correct me if i am wrong)

rndquu

> > It seems the escalation is based on the fact that a relatively short (?) duration was used in the example. This duration can be arbitrarily extended by using more capital. Overall this should not be happening in a TWAP, and for example in the TWAP as implemented in Uniswap, it is not possible to manipulate more efficiently by providing more liquidity during 2 blocks.
>
> @pavlovcik @rndquu First of all we need to know max aceptable by protocol team time that pool can be DOSed taking into account that admin can in any time prevent DOS by adjusting threshholds. Based on example you need 100K pool liquidity * 15 = 1.5M$ AND control over 2 block to DOS for 8 min (correct me if i am wrong)

I don't think that any time >0 seconds is an acceptable DOS duration for any protocol :)

0xLogos

> I don't think that any time >0 seconds is an acceptable DOS duration for any protocol :)

Sorry but this is not true. There's no perfect systems, you have to come up with suitable threat model.

Also I believe that in order to understand the severity of the issue you need to somehow evaluate the difference between calculating price based on time average balances and using time average price like uni v3 because this is proposed solution. (If you have control over 2 block you can do similar thing with prices swapping back and forth in 2 blocks)

My point is example in report is very vague and there's need at least for more realistic PoC and ideally comparison with proposed solution.

Czar102

I believe oracle manipulation can have a more severe impact than a DoS for a few minutes, see https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13#issuecomment-1945710223.

Planning to reject the escalation and leave the issue as is.

0xLogos

https://github.com/sherlock-audit/2023-12-ubiquity-judging/issues/13#issuecomment-1945710223 is inaccurate

CergyK

Since we are discussing this issue, it seems #184 #197 #212 are invalid/unrelated

Czar102

Thank you @0xLogos for noting a mistake in my previous statement. @CergyK there seems to be no loss of funds impact here, I think it's just a DoS for a few minutes (TWAP length) and that's the risk of using TWAP. No serious loss is inflicted on anyone.

We are also discussing the severity of TWAP manipulation on #59 right now, it may be relevant.

sherlock-admin

The protocol team fixed this issue in PR/commit https://github.com/ubiquity/ubiquity-dollar/pull/893.

Czar102

After extensive discussions with the LSW and the Lead Judge, planning to leave this issue a Medium severity one, as it doesn't only cause a DoS, but may allow for an easy value extraction strategy from the protocol, similar to the one described in #17 and duplicates.

Planning to reject the escalation and leave the issue as is.

Czar102

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xLogos: rejected

sherlock-admin

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.