SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	Public
Prepared for:	Mellow Protocol
Prepared by:	Sherlock
Lead Security Expert:	hash
Dates Audited:	June 17 - June 27, 2024
Prepared on:	July 30, 2024

Introduction

Modular Liquid Restaking Primitive for Permissionless LRTs Creation & Curation.

Scope

Repository: mellow-finance/mellow-lrt

Branch: dev-symbiotic-deploy

Commit: a7165a279330a213d7d24b0b2ea6adf2d61b8d69

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

Security experts who found valid issues

hash	Ironsidesec	X12
eeyore	h2134	

Issue M-1: `ratiosX96Value` rounds in favor of user and not vault

Source: https://github.com/sherlock-audit/2024-06-mellow-judging/issues/61

Found by

Ironsidesec, X12, eeyore, hash

Summary

`ratiosX96Value` is rounded down instead of up, causing withdrawals to favor users. This can slowly decrease the value in our vault, potentially leading to insolvency.

Vulnerability Detail

`ratiosX96Value`, calculated in calculateStack,

```
s.ratiosX96Value += FullMath.mulDiv(s.ratiosX96[i], priceX96, Q96);
```

is used as a denominator inside analyzeRequest to calculate `coefficientX96` and the user's `expectedAmounts`.

```
uint256 value = FullMath.mulDiv(lpAmount, s.totalValue, s.totalSupply);
value = FullMath.mulDiv(value, D9 - s.feeD9, D9);
uint256 coefficientX96 = FullMath.mulDiv(value, Q96, s.ratiosX96Value);
...
expectedAmounts[i] = ratiosX96 == 0 ? 0 : FullMath.mulDiv(coefficientX96,
↪  ratiosX96, Q96);
```

However, calculateStack rounds the denominator down (thanks to `mulDiv`) , which increases `coefficientX96` and thus increases what users withdraw.

This is unwanted behavior in vaults. Rounding towards users decreases the vault's value and can ultimately cause insolvency. The previous audit found a similar issue in the deposit function - M1.

Impact

The vault may become insolvent or lose small amounts of funds with each withdrawal.

Code Snippet

```
for (uint256 i = 0; i < tokens.length; i++) {
    uint256 priceX96 = priceOracle.priceX96(address(this), tokens[i]);
    s.totalValue += FullMath.mulDiv(amounts[i], priceX96, Q96);
    s.ratiosX96Value += FullMath.mulDiv(s.ratiosX96[i], priceX96, Q96);
    s.erc20Balances[i] = IERC20(tokens[i]).balanceOf(address(this));
}
```

Tool used

Manual Review

Recommendation

Round the value up instead of down, similar to how it's done inside deposit.

```
for (uint256 i = 0; i < tokens.length; i++) {
    uint256 priceX96 = priceOracle.priceX96(address(this), tokens[i]);
    s.totalValue += FullMath.mulDiv(amounts[i], priceX96, Q96);
-   s.ratiosX96Value += FullMath.mulDiv(s.ratiosX96[i], priceX96, Q96);
+   s.ratiosX96Value += FullMath.mulDivRoundingUp(s.ratiosX96[i], priceX96, Q96);
    s.erc20Balances[i] = IERC20(tokens[i]).balanceOf(address(this));
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/mellow-finance/mellow-lrt/pull/44

10xhash

Fixed Now rounded up

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-2: User may not receive profit from withdrawal fee as expected and attacker can steal value from pool

Source: https://github.com/sherlock-audit/2024-06-mellow-judging/issues/162

The protocol has acknowledged this issue.

Found by

Ironsidesec, h2134

Summary

When a user's withdrawal request is processed, the user is charged a withdrawal fee. The withdrawal fee increases the value of the LP tokens, and the LP token holders earns profit from it.

However, when multiple withdrawal requests are processed in a batch, the single withdrawal contributes no value to LP token until the end of the process, user may not receive profit as expected, and the value can be steal by an attacker through sandwich attack.

Vulnerability Detail

It is expected that a user earn profit from the withdrawal fee paid by other users, for example, when `userA` withdraws, the withdrawal fee is charged and LP token becomes more valuable, then when `userB` withdraws, they may withdraw more tokens than they deposited.

However, users earn no profit from withdrawal fee if their withdrawal requests are processed in a batch. When multiple withdrawal requests are processed in a batch, the state stack required for processing withdrawal requests is calculated:

```
    function calculateStack()
        public
        view
        returns (ProcessWithdrawalsStack memory s)
    {
        (address[] memory tokens, uint256[] memory amounts) = underlyingTvl();
        s = ProcessWithdrawalsStack({
            tokens: tokens,
            ratiosX96: IRatiosOracle(configurator.ratiosOracle())
                .getTargetRatiosX96(address(this), false),
            erc20Balances: new uint256[](tokens.length),
@>          totalSupply: totalSupply(),
            totalValue: 0,
            ratiosX96Value: 0,
            timestamp: block.timestamp,
            feeD9: configurator.withdrawalFeeD9(),
            tokensHash: keccak256(abi.encode(tokens))
        });

        ...
    }
```

The state stack is then passed to analyzeRequest() to calculate the expected token amount for each request.

```
    function analyzeRequest(
        ProcessWithdrawalsStack memory s,
        WithdrawalRequest memory request
    ) public pure returns (bool, bool, uint256[] memory expectedAmounts) {
        ...

@>      uint256 value = FullMath.mulDiv(lpAmount, s.totalValue, s.totalSupply);
@>      value = FullMath.mulDiv(value, D9 - s.feeD9, D9);
@>      uint256 coefficientX96 = FullMath.mulDiv(value, Q96, s.ratiosX96Value);

        ...

        for (uint256 i = 0; i < length; i++) {
            uint256 ratiosX96 = s.ratiosX96[i];
@>          expectedAmounts[i] = ratiosX96 == 0 ? 0 :
↪   FullMath.mulDiv(coefficientX96, ratiosX96, Q96);
            if (expectedAmounts[i] >= request.minAmounts[i]) continue;
            return (false, false, expectedAmounts);
        }

       ...

        return (true, true, expectedAmounts);
    }
```

To simplify, the expected token amount is calculated as below:

> expectedAmounts = ((lpAmount * s.totalValue / s.totalSupply) / s.ratiosX96Value) * ratiosX96

The problem is that `s.totalSupply` and `s.totalValue` are constant during the calculation process, indicating that no withdrawal fee is contributed to the value of the LP token, even if the fee has been charged. For instance, in the batch `[userA request, userB request, userC request]`, `userB` won't receive profit from the withdrawal fee paid by `userA`, and `userC` receive no profit from the fee paid by `userA` and `userB`.

The users' LP tokens are burned at the end of the batch withdrawal process, by then the value of the LP token is increased. This can be leveraged by an attacker, who may intentionally front-run a large batch withdrawal to mint LP tokens and then back-run to withdraw, hence steal the value from the pool.

Impact

1. User may receive much less profit than expected, or no profit at all;
2. Malicious user can steal withdrawal fee from the pool.

Code Snippet

https://github.com/sherlock-audit/2024-06-mellow/blob/main/mellow-lrt/src/Vault.sol#L476-L504

Tool used

Manual Review

Recommendation

It is expected the withdrawal requests are processed sequentially in the registration time order, so each withdrawal should generate profit to the following ones. This can be achieved by updating `s.totalSupply` and `s.totalValue` in the process of calculating `expectedAmounts`.

```
    function analyzeRequest(
        ProcessWithdrawalsStack memory s,
        WithdrawalRequest memory request
    ) public pure returns (bool, bool, uint256[] memory expectedAmounts) {
        ...

        for (uint256 i = 0; i < length; i++) {
-           if (s.erc20Balances[i] >= expectedAmounts[i]) continue;
+           if (s.erc20Balances[i] >= expectedAmounts[i]) {
+               s.totalSupply -= request.amount;
+               s.totalValue -= value;
+               continue;
+           }

            return (true, false, expectedAmounts);
        }
        return (true, true, expectedAmounts);
    }
```

Discussion

0xh2134

Escalate.

This issue is valid.

sherlock-admin3

> Escalate.
>
> This issue is valid.

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.

z3s

The reason I invalidated this submission was: For the first issue about fees, I think it's a design decision.

> Design decisions are not valid issues. Even if the design is suboptimal, but doesn't imply any loss of funds, these issues are considered informational.

For the second issue, the protocol team answer for front-running issues was

> `processWithdrawals` will occur through flashbots.

In my opinion this submission is valid Low/Info.

0xh2134

@z3s Thanks for quick reply, really appreciate it. From my point of view:

1. I agree it's a design decision, but it's more than suboptimal, given that withdrawal fee is one the the few (2, I believe) ways that user can earn the profit, there can be significant profit loss to the users;
2. Sure they can, but unless the same is stated before or during the contest period, it should not be took into consideration, otherwise all the front-running findings can be invalidated by simply saying "invalid because admin/user can use flashbots";

3. Another impact I can think of is that **a large amount of collateral token can be stuck** if there is a "whole withdrawal" (this is possible and can be happening for several times, especially in the early/late stage of the project): assume that there are 100 depositors, the average deposit is 10 ether and withdrawal fee is 5%, if all of them submit to withdraw and their requests are processed in a batch, the withdrawal fee is 50 ether in total and the fee is stuck in the contract because no one has LP token left. (One may argue that there is still initial deposit from admin, but from the context of the contest, the initial deposit is for setting LP price only, it's reasonable to assume the deposit won't be withdrawn).

WangSecurity

About the first impact, I believe it's the design decision to not payout the fee to other users in the withdrawal batch and the User B and User C in your example shouldn't get the fee. About the second impact, I don't think we can say the fee is stolen, because I assume the attacker won't receive the entire fee (unless he's the only one in the vault) and they will receive their value proportional to their deposit. Hence, I believe it also works as it should.

About the new impact in the above comment. Firstly, it has to be in the report, additionally, again I believe it's the way the protocol should work, if there is no one in the vault, then no one should receive the fee.

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

0xh2134

@WangSecurity

1. It's design decision but leads to user losses profit, I don't think it's just suboptimal;
2. It's true attacker can not receive all the fees, but they will steal much more than expected, depending on the processed withdrawal in a batch, the stolen fees can be 100 times (assume 100 withdrawals processed) larger than usual, I don't believe this is expected;
3. Maybe I misunderstand it but I think an issue should be judged by the root cause, not just the impact, I can see #244 mentioned the similar impact and it is escalated to be valid.

WangSecurity

> It's design decision but leads to user losses profit, I don't think it's just suboptimal

So that's a tricky part that we've got a misunderstanding on, I'll try to explain my view. When the withdrawals are made in a batch, I don't think it makes sense to increase the profit of the next user in this batch. They're withdrawing from the pool

SHERLOCK

in the very same transaction/block basically so why they should get the profit from it, if they are also going to exit the vault in the very same transaction. From my point of view, the users that remain in the pool should receive this fee, cause they're still in the pool after that batch withdrawal. So that's what I mean it's a design decision to reward the users that are remained in the vault and not the ones that are withdrawing in the very same transaction.

> they will steal much more than expected

What do you mean exactly by more than usual? I assume even if they didn't intentionally deposit before the large withdrawal (maybe a couple of blocks before), they still get the fees proportional to their deposit. Though, I see the scenario how this accruing of value is not totally fair to other users who have been invested for longer than the attacker, I think it's the design to allocate the fees on pro rata basis to everyone invested in the vault.

> Maybe I misunderstand it but I think an issue should be judged by the root cause, not just the impact, I can see https://github.com/sherlock-audit/2024-06-mellow-judging/issues/244 mentioned the similar impact and it is escalated to be valid

You're correct, I've missed both of these have the same root cause, but I'm looking to invalidate #244 as well, since the left in the contract fees can be extracted by the admin.

Hence, the decision remains the same, reject the escalation and leave the issue as it is.

0xh2134

@WangSecurity Thanks for the clarification.

> What do you mean exactly by more than usual?

When it is compared to the normal (individual) withdrawal, the stolen fee can be much larger.

> but I'm looking to invalidate https://github.com/sherlock-audit/2024-06-mellow-judging/issues/244 as well

Likewise, the stuck funds can be much larger than expect, so if you reconsider #244 to be valid, you may kindly reconsider this issue.

Cheers!

WangSecurity

Yep, I'm considering both the scenario in #116 and this issue, since they're pretty much the same.

WangSecurity

Upon further inspecting this scenario, I agree it's valid and allows the attacker to extract value from the protocol and steal it from the honest depositors. Planning to accept the escalation and validate the issue with Medium severity, because to extract large value the attacker has to have a large capital, has to wait for the withdrawal delay and pay the withdrawal fee. I believe these arguments together make this medium severity. The duplicate is #116. Are there any other duplicates?

10xhash

> Upon further inspecting this scenario, I agree it's valid and allows the attacker to extract value from the protocol and steal it from the honest depositors. Planning to accept the escalation and validate the issue with Medium severity, because to extract large value the attacker has to have a large capital, has to wait for the withdrawal delay and pay the withdrawal fee. I believe these arguments together make this medium severity. The duplicate is #116. Are there any other duplicates?

How is this valid? This is purely a design choice which is perfectly valid

> allows the attacker to extract value from the protocol and steal it from the honest depositors

If an attacker is depositing a large amount of assets, they will obtain the proportionate fees. It doesn't matter if the protocol decides to give the fees to the remaining depositors (in this case) or to include the depositors who are withdrawing in the same call. And there is no perfect logic as to how to distribute the fees This type of design combats MEV in certain scenarios (because the attacker has to wait for processingDelay in order to gain the fees) compared to the other scenario where the fees is being distributed to the current withdrawer's too (a user having withdrawal request at the last can front run increase the deposit amount and then gain the fees in the same block)
And this type of attack is usually not profitable because the attacker has to pay the fees on the entire deposit amount while they only gain the fee from other user deposits (unless extreme scenarios like very low remaining vault amount and very high in queue withdrawals occur)

IronsideSec

> How is this valid? This is purely a design choice which is perfectly valid

Design choice from sponsor was to allow funds back to vault, increasing LP vaulue by share price. Sniping that wih MEV is loss of funds. Wrong extraction of value that supposed to go to LPers.

> It doesn't matter if the protocol decides to give the fees to the remaining depositors (in this case) or to include the depositors who are withdrawing in the same call. And there is no perfect logic as to how to distribute the fees

There is perfect logic here. Protocol decided to not charge fees there by incresing lp share price. But its prone to MEV extraction.

> And this type of attack is usually not profitable because the attacker has to pay the fees on the entire deposit amount while they only gain the fee from other user deposits (unless extreme scenarios like very low remaining vault amount and very high in queue withdrawals occur)

`very high in queue withdrawals` is not exetreme low scenario, it will occur frequently. Operator's process withdrawal for example 20+ users will have huge volume of value to be extracted.

WangSecurity

My initial thought was that it is the design decision as well. But, after thinking about it more, I changed my decision because it allows to extract basically free value from the vault without actually being invested, essentially stealing value from the honest depositors. Moreover, this can be repeated on every large withdrawal, so the invested into protocol users don't get the full value they have to every time. Hence, the decision remains the same, validate this with medium severity with #116 being a duplicate.

0x3b33

I am not sure about 116, however this issue has a number of problems:

---

1. Most obvious one - the solution would actually allow for **more MEV** as everything that needs to be done is for a user to FR withdraw, and schedule a big batch a the end. This way he will:

- leave without any penalties
- claim a huge portion of the withdraws (as he is the last and receiving the most LP value appreciation)
- be exposed to 0 risk

---

2. Either the attacker needs to have a lot of assets as @10xhash pointed out

> And this type of attack is usually not profitable because the attacker has to pay the fees on the entire deposit amount while they only gain the fee from other user deposits (unless extreme scenarios like very low remaining vault amount and very high in queue withdrawals occur)

Or in the case that @IronsideSec mentioned, the attacker will have a really small profit (most likely a loss due to point 3), while exposing himself to a lot of risk very high in queue withdrawals is not extreme low scenario, it will occur frequently. Operator's process withdrawal for example 20+ users will have huge volume of value to be extracted.

---

3. The attacker is still subjected to the withdraw fee. That combined with the increased risk makes this issue unworthy. Let me add some math:

Example - 1% withdraw fee:

Owning 50% of the pool after withdraws, will need the pool to generate 1% of our balance (or 0.5% of the pool total supply) in fees, meaning that we need to control 25% of the pool (before withdraw) and 50% to leave in order to pay **0 fee**. Not even generating profit.

() *prerequisites*	*values*
()	
Withdraw fee	1%
Pool size	20000
total withdraws	10000
Attacker balance	5000
Remaining users balance	5000
Fee generated	10k * 1% = 100
Fees paid to attacker	100 * 50% = 50
()	

When owning 25% (before deposit) we would need at least 50% to leave in every fee range, be it the fee being 1%, 5% or even 10%. That is because we would also pay that fee when withdrawing.

The numbers speek for themselves. I don't think more explanations are needed.

IronsideSec

#116's profitability and POC discussed

- likelihood & POC
- profitability & here

WangSecurity

> How is this valid? This is purely a design choice which is perfectly valid

It's a fair point, but I still think in that case the attacker can get the withdrawal fee almost for free. Almost is that they still have to suffer a withdrawal fee when they decide to exit and a withdrawal delay, but they don't have to be deposited into the protocol and acquire additional risks associated with it.

> If an attacker is depositing a large amount of assets, they will obtain the proportionate fees. It doesn't matter if the protocol decides to give the fees to the remaining depositors (in this case) or to include the depositors who are withdrawing in the same call. And there is no perfect logic as to how to distribute the fees

Yes, the attacker will indeed suffer the proportionate withdrawal fees, but as seen in the above comment (in the POC linked in the above comment), the attack still has profit even taking the fees into account. About the solution, also a fair point, but a possible solution can be distributing rewards based on the time the depositors have spent in the vault. This way, front-running the deposit will give you zero value since the block.timestamp for the deposit and processed withdrawals would be the same. I understand it's a complex solution, still I believe it can solve this issue.

> This type of design combats MEV in certain scenarios (because the attacker has to wait for processingDelay in order to gain the fees) compared to the other scenario where the fees is being distributed to the current withdrawer's too (a user having withdrawal request at the last can front run increase the deposit amount and then gain the fees in the same block) And this type of attack is usually not profitable because the attacker has to pay the fees on the entire deposit amount while they only gain the fee from other user deposits (unless extreme scenarios like very low remaining vault amount and very high in queue withdrawals occur)

This is also a fair point, but it still doesn't change the fact that the user can get the profit, essentially, stealing from other users, which again has been proven to be profitable when the user exits the protocol and pays the withdrawal fee. I agree that the withdrawal delay here is a constraint, but it doesn't eliminate the attack.

> Most obvious one - the solution would actually allow for more MEV as everything that needs to be done is for a user to FR withdraw, and schedule a big batch a the end. This way he will

As I understand you refer to the solution in this report? It may not be perfect, of course. But the decision is based on the constraints and impact, not on the optimality and correctness of the solution.

> Either the attacker needs to have a lot of assets as @10xhash pointed out Or in the case that @IronsideSec mentioned, the attacker will have a really small profit (most likely a loss due to point 3), while exposing himself to a lot of risk

Indeed, but the larger the capital of the attacker, the larger the profit they get and the loss other vault depositors suffer.

> The attacker is still subjected to the withdraw fee. That combined with the increased risk makes this issue unworthy. Let me add some math

Yep, the withdrawal fee is a constrain here, but as proven with the coded POC, the attacker indeed can receive the profit and cause losses.

I hope that answers the questions and explains why I believe it's an issue. I see that it can be a design, but I believe it's an issue since it allows the user to gain a portion of the withdrawal fee almost for free, even taking the withdrawal fee and the delay into account, without being deposited into the protocol, contributing to it and being exposed to other risks of being deposited into the protocol, essentially stealing this portion of the fee from other users.

Hence, the decision remains the same to accept the escalation and validate this report with Medium severity and set #116 as duplicate.

10xhash

> > How is this valid? This is purely a design choice which is perfectly valid
>
> It's a fair point, but I still think in that case the attacker can get the withdrawal fee almost for free. Almost is that they still have to suffer a withdrawal fee when they decide to exit and a withdrawal delay, but they don't have to be deposited into the protocol and acquire additional risks associated with it.
>
> > If an attacker is depositing a large amount of assets, they will obtain the proportionate fees. It doesn't matter if the protocol decides to give the fees to the remaining depositors (in this case) or to include the depositors who are withdrawing in the same call. And there is no perfect logic as to how to distribute the fees
>
> Yes, the attacker will indeed suffer the proportionate withdrawal fees, but as seen in the above comment (in the POC linked in the above comment), the attack still has profit even taking the fees into account. About the solution, also a fair point, but a possible solution can be distributing rewards based on the time the depositors have spent in the vault. This way, front-running the deposit will give you zero value since the block.timestamp for the deposit and processed withdrawals would be the same. I understand it's a complex solution, still I believe it can solve this issue.
>
> > This type of design combats MEV in certain scenarios (because the attacker has to wait for processingDelay in order to gain the

> fees) compared to the other scenario where the fees is being distributed to the current withdrawer's too (a user having withdrawal request at the last can front run increase the deposit amount and then gain the fees in the same block) And this type of attack is usually not profitable because the attacker has to pay the fees on the entire deposit amount while they only gain the fee from other user deposits (unless extreme scenarios like very low remaining vault amount and very high in queue withdrawals occur)

This is also a fair point, but it still doesn't change the fact that the user can get the profit, essentially, stealing from other users, which again has been proven to be profitable when the user exits the protocol and pays the withdrawal fee. I agree that the withdrawal delay here is a constraint, but it doesn't eliminate the attack.

> Most obvious one - the solution would actually allow for more MEV as everything that needs to be done is for a user to FR withdraw, and schedule a big batch a the end. This way he will

As I understand you refer to the solution in this report? It may not be perfect, of course. But the decision is based on the constraints and impact, not on the optimality and correctness of the solution.

> Either the attacker needs to have a lot of assets as @10xhash pointed out Or in the case that @IronsideSec mentioned, the attacker will have a really small profit (most likely a loss due to point 3), while exposing himself to a lot of risk

Indeed, but the larger the capital of the attacker, the larger the profit they get and the loss other vault depositors suffer.

> The attacker is still subjected to the withdraw fee. That combined with the increased risk makes this issue unworthy. Let me add some math

Yep, the withdrawal fee is a constrain here, but as proven with the coded POC, the attacker indeed can receive the profit and cause losses.

I hope that answers the questions and explains why I believe it's an issue. I see that it can be a design, but I believe it's an issue since it allows the user to gain a portion of the withdrawal fee almost for free, even taking the withdrawal fee and the delay into account, without being deposited into the protocol, contributing to it and being exposed to other risks of being deposited into the protocol, essentially stealing this portion of the fee from other users.

Hence, the decision remains the same to accept the escalation and validate this report with Medium severity and set #116 as duplicate.

`Almost is that they still have to suffer a withdrawal fee when they decide to exit and a withdrawal delay, but they don't have to be deposited into the protocol and acquire additional risks associated with it.` : What is the difference b/w suffering a withdrawal delay and being deposited into the protocol? The protocol is choosing the withdrawal delay according to their required convenience (ie. how much time they want the user's funds to be in the protocol before they can exit. And the estimated value was 1-7days)

`About the solution, also a fair point, but a possible solution can be distributing rewards based on the time the depositors have spent in the vault. This way, front-running the deposit will give you zero value since the block.timestamp for the deposit and processed withdrawals would be the same. I understand it's a complex solution, still I believe it can solve this issue.`: This is changing how the protocol is intending to work and would exactly be what is considered changing the design. The protocol is weighing on the deposit amounts rather than the time they are spending. If a user brings in 100eth and spends 2days in the protocol, he is considered more valuable to the protocol than a user who has deposited 10 eth and spent 200 days in the protocol.Uniswap gives the swap fees proportional to the concentrated liquidity provided by the user. And arguing that instead they should also factor in the time for which the deposits have been made would be fundamentally changing their design. Uniswap suffers from JIT, but here the team has decided to enforce withdrawal delay and withdrawal fee which they are comfortable with. If any user deposits a large portion of money for the withdrawal delay period, they are eligible for the fee share because they bring in that much amount of deposits to the protocol equivalent to any normal user who without any malicious intention deposits and withdraws in the said timeframe being eligible for the same amount of reward. And if there is a large amount of withdrawals compared to the remaining amounts in the protocol (or to be precise, atleast >50% of the entire vault tvl should be withdrawn in a single call which is the required condition for a user to gain a profit by performing the above sequence), the increase in LP share value would incentivize more user's to deposit which would benefit the protocol.

`This is also a fair point, but it still doesn't change the fact that the user can get the profit, essentially, stealing from other users`: This is not a stealing of deserved amount of assets from the other depositors. The project is not a whoever deposits longest will obtain the most rewards (from withdrawals), staking type project which is being broken down by a large deposit for a short timeframe. Apart from the core strategy of the protocol, withdrawal fees are being distributed to the depositors based on the amount of deposit that they provide which is being respected here (with the main intention of the fees being to avoid free swaps b/w the assets)

`Indeed, but the larger the capital of the attacker, the larger the profit` `they get and the loss other vault depositors suffer` The POC meets the required condition for the required increase in lp share value (ie. >50.125% of the vault value being withdrawn) and it doesn't mean a user can simply deposit a lot of assets and gain more and more value. They are bounded by fees that they will have to pay and the amount that is being withdrawn. But the main point is the project wants to distribute the fees proportional to the deposit amount and a withdrawal delay that they are comfortable with is being enforced (ie. if a user is spending this much time in the protocol, they are eligible for the fee share)

0xh2134

> What is the difference b/w suffering a withdrawal delay and being deposited into the protocol?

The difference is that attacker's funds won't be staked into Bound (In case Bond deposit limit is reached), they are considered no valuable to the Vault.

> The protocol is weighing on the deposit amounts rather than the time they are spending

This protocol is not Uniswap, or anything like it, there is no swap fee, if no funds deposited into Bond, the attacker make no contribution to the protocol, no matter how much funds they deposited.

> withdrawal fees are being distributed to the depositors based on the amount of deposit that they provide which is being respected here

If attacker makers no contribution to the protocol and they are receive withdrawal fees, then they are stealing from honest users.

> They are bounded by fees that they will have to pay and the amount that is being withdrawn

Even if the profit is bounded, the attacker can earn large profit by repeating on every large withdrawal.

WangSecurity

> What is the difference b/w suffering a withdrawal delay and being deposited into the protocol? The protocol is choosing the withdrawal delay according to their required convenience (ie. how much time they want the user's funds to be in the protocol before they can exit. And the estimated value was 1-7days)

As @0xh2134 they indeed may be not deposited into the underlying strategy if there's a limit reached. Moreover, the attacker in that case spends as little time as possible in the vault, basically depositing into it to get a share of the portion fee.

> This is changing how the protocol is intending to work and would exactly be what is considered changing the design. The protocol is weighing on

> the deposit amounts rather than the time they are spending. If a user brings in 100eth and spends 2days in the protocol, he is considered more valuable to the protocol than a user who has deposited 10 eth and spent 200 days in the protocol.Uniswap gives the swap fees proportional to the concentrated liquidity provided by the user. And arguing that instead they should also factor in the time for which the deposits have been made would be fundamentally changing their design. Uniswap suffers from JIT, but here the team has decided to enforce withdrawal delay and withdrawal fee which they are comfortable with. If any user deposits a large portion of money for the withdrawal delay period, they are eligible for the fee share because they bring in that much amount of deposits to the protocol equivalent to any normal user who without any malicious intention deposits and withdraws in the said timeframe being eligible for the same amount of reward. And if there is a large amount of withdrawals compared to the remaining amounts in the protocol (or to be precise, atleast >50% of the entire vault tvl should be withdrawn in a single call which is the required condition for a user to gain a profit by performing the above sequence), the increase in LP share value would incentivize more user's to deposit which would benefit the protocol

Again, I see your point for it being the design, but I also see how it is a loss of funds for the honest depositors. The solution that I shared is just one possible mitigation and I've acknowledged that it would be very complex. Additionally, there can be other mitigations and it doesn't affect the impact and the constraints of the attack. I see that for that attack there's also should be a very large deposit and large capital from the attacker, it doesn't change the fact the attacker can profit and cause losses to other depositors.

> This is not a stealing of deserved amount of assets from the other depositors. The project is not a whoever deposits longest will obtain the most rewards (from withdrawals), staking type project which is being broken down by a large deposit for a short timeframe. Apart from the core strategy of the protocol, withdrawal fees are being distributed to the depositors based on the amount of deposit that they provide which is being respected here (with the main intention of the fees being to avoid free swaps b/w the assets)

I didn't say the project is `whoever deposits longest will obtain the most rewards (from withdrawals)`. Again, I see your points in this being a design decision, but I see it as a loss of funds for the depositors of the protocol, while the attacker can spend a minimum of time in the Vault just to accrue the rewards.

> The POC meets the required condition for the required increase in lp share value (ie. >50.125% of the vault value being withdrawn) and it doesn't mean a user can simply deposit a lot of assets and gain more and more value. They are bounded by fees that they will have to pay and the

> amount that is being withdrawn. But the main point is the project wants to distribute the fees proportional to the deposit amount and a withdrawal delay that they are comfortable with is being enforced (ie. if a user is spending this much time in the protocol, they are eligible for the fee share)

In the previous comment, I've already acknowledged that I take both the fee and the delay into account. You're correct that the large withdrawal is also a pre-condition here. Still, the POC shows how the attacker exits the protocol with the profit stolen from other users. I see how it can be a design decision, but I see it as a loss of funds as well. Hence, the decision remains the same, planning to validate with medium severity and accept the escalation with #116 being a duplicate.

10xhash

I don't have any more points to make except if the limit case is decisive in determining the validity of the issue (it seems you are considering this valid even without the limit case, else the scenario of limit case should be handled separately and if protocol don't want to share fees for deposits over the limit, they should've stopped deposits from happening after the limit)

WangSecurity

Yep, I consider this valid even without the limit case. It was more of an example that during the vulnerability, the attacker's funds can be just lying in the vault, without being deposited into bonds and accruing value. But even without it, I see this as valid.

The decision remains the same, planning to accept the escalation and validate with medium severity. Duplicate is #116

WangSecurity

Result: Medium Has duplicates

sherlock-admin2

Escalations have been resolved successfully!

Escalation status:

- 0xh2134: accepted

xKeywordx

One possible fix is to prevent withdrawing into the same block that you deposited so that MEV can not happen.

Issue M-3: Predefined amount parameter can challange allocation to Obol validators

Source: https://github.com/sherlock-audit/2024-06-mellow-judging/issues/260

Found by

hash

Summary

Deposits need not be allocated to Obol validators

Vulnerability Detail

In `StakingModule`, the amount to be deposited in Lido is supposed to go to the Obol validators. But the only check kept for this is that the `stakingModuleId` is set to `SimpleDVT` module id in Lido and `unfinalizedstETH <= bufferedETH`

link

```
function convertAndDeposit(
    uint256 amount,
    uint256 blockNumber,
    bytes32 blockHash,
    bytes32 depositRoot,
    uint256 nonce,
    bytes calldata depositCalldata,
    IDepositSecurityModule.Signature[] calldata sortedGuardianSignatures
) external onlyDelegateCall {
    if (IERC20(weth).balanceOf(address(this)) < amount)
        revert NotEnoughWeth();


    uint256 unfinalizedStETH = withdrawalQueue.unfinalizedStETH();
    uint256 bufferedEther = ISteth(steth).getBufferedEther();
    if (bufferedEther < unfinalizedStETH)
        revert InvalidWithdrawalQueueState();


    _wethToWSteth(amount);
    depositSecurityModule.depositBufferedEther(
        blockNumber,
        blockHash,
        depositRoot,
        stakingModuleId,
        nonce,
        depositCalldata,
        sortedGuardianSignatures
    );
}
```

There is no enforcement/checks kept on the `amount` param. It is possible to pass in any value for the `amount` param or the buffered ETH amount in lido to change from the value using which the amount was calculated before the call due to other deposits . This allows for scenarios where the `amount` is not deposited into validators at all (not in multiples of 32 eth), could be shared with other staking modules or could result in reverts due to the max limit restriction in the lido side. There is also no enforcement that the deposits made would be allocated to Obol validators itself since the simpleDVT staking module in Lido also contains SSV operators

Impact

The amount to be staked for obol validators can be assigned to other operators

Code Snippet

Tool used

Manual Review

Recommendation

Consult with lido team to make it possible to deposit specifically to a certain set of operators and calculate the maximum amount depositable to that set at the instant of the call rather than a predefined value

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/mellow-finance/mellow-lrt/pull/44

ctf-sec

Escalate.

this is low / info finding.

we are calling this function from SimpleDVTStakingStrategy.sol

```
function convertAndDeposit(
        uint256 amount,
        uint256 blockNumber,
        bytes32 blockHash,
        bytes32 depositRoot,
        uint256 nonce,
        bytes calldata depositCalldata,
        IDepositSecurityModule.Signature[] calldata sortedGuardianSignatures
    ) external returns (bool success) {
        (success, ) = vault.delegateCall(
            address(stakingModule),
            abi.encodeWithSelector(
                IStakingModule.convertAndDeposit.selector,
                amount,
                blockNumber,
                blockHash,
                depositRoot,
                nonce,
                depositCalldata,
                sortedGuardianSignatures
            )
        );
        emit ConvertAndDeposit(success, msg.sender);
    }
```

the parameter sortedGuardianSignatures ensure that this function cannot be called permissionlessly.

then this is issue is only called by admin and admin is consider trusted.

sherlock-admin3

> Escalate.
>
> this is low / info finding.
>
> we are calling this function from SimpleDVTStakingStrategy.sol
>
> ```
> function convertAndDeposit(
>         uint256 amount,
>         uint256 blockNumber,
>         bytes32 blockHash,
>         bytes32 depositRoot,
>         uint256 nonce,
>         bytes calldata depositCalldata,
>         IDepositSecurityModule.Signature[] calldata sortedGuardianSignatures
> ```

```
    ) external returns (bool success) {
        (success, ) = vault.delegateCall(
            address(stakingModule),
            abi.encodeWithSelector(
                IStakingModule.convertAndDeposit.selector,
                amount,
                blockNumber,
                blockHash,
                depositRoot,
                nonce,
                depositCalldata,
                sortedGuardianSignatures
            )
        );
        emit ConvertAndDeposit(success, msg.sender);
    }
```

> the parameter sortedGuardianSignatures ensure that this function cannot be called permissionlessly.
>
> then this is issue is only called by admin and admin is consider trusted.

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.

WangSecurity

@10xhash do you have any counterarguments?

10xhash

> @10xhash do you have any counterarguments?

Yes

1. As mentioned in the report, if an amount is precalculated it can be incorrect (such that the made deposits will not be allocated to obol) at the time of actual deposit due to other deposits occurring in lido. This doesn't require any external user to manipulate the `amount` param

```
..... or the buffered ETH amount in lido to change from the value using which
↪  the amount was calculated before the call due to other deposits
```

2. In contest readme, the usage of special off-chain mechanisms were denied. And the depositor bot of lido (which would be invoking the function with the

> signatures etc.) has been invoking on lido's deposit function via public mempool all these time (https://mempool.guru/tx/0xe61ea1c5160a40ba43ce4ef13a597bfa7969d235efe33010697985116cf63cb7) which allows any user to even change just the `amount` param

JeffCX

https://docs.lido.fi/contracts/deposit-security-module/

LIDO guaridan set up rigorous measure to prevent deposit related frontrunning

> Due to front-running vulnerability, we proposed to establish the Deposit Security Committee dedicated to ensuring the safety of deposits on the Beacon chain:
>
> monitoring the history of deposits and the set of Lido keys available for the deposit, signing and disseminating messages allowing deposits;
>
> signing the special message allowing anyone to pause deposits once the malicious Node Operator pre-deposits are detected.
>
> Each member must generate an EOA address to sign messages with their private key. The addresses of the committee members will be added to the smart contract.
>
> To make a deposit, we propose to collect a quorum of 4/6 of the signatures of the committee members

user cannot spoof the stakingModuleId as well, which is used to identify the staking module party.

also I think the amount parameter just convert WETH to wstETH and not related to deposit request.

also look at the recommendation from original report

> Consult with lido team to make it possible to deposit specifically to a certain set of operators and calculate the maximum amount depositable to that set at the instant of the call rather than a predefined value

I mean we should assume the LIDO behave correctly, otherwise, we can say all staked ETH are lost...

10xhash

> https://docs.lido.fi/contracts/deposit-security-module/
>
> LIDO guaridan set up rigorous measure to prevent deposit related frontrunning
>
> > Due to front-running vulnerability, we proposed to establish the Deposit Security Committee dedicated to ensuring the safety of

> > deposits on the Beacon chain:
> >
> > monitoring the history of deposits and the set of Lido keys available for the deposit, signing and disseminating messages allowing deposits;
> >
> > signing the special message allowing anyone to pause deposits once the malicious Node Operator pre-deposits are detected.
> >
> > Each member must generate an EOA address to sign messages with their private key. The addresses of the committee members will be added to the smart contract.
> >
> > To make a deposit, we propose to collect a quorum of 4/6 of the signatures of the committee members
>
> user cannot spoof the stakingModuleId as well, which is used to identify the staking module party.
>
> also I think the amount parameter just convert WETH to wstETH and not related to deposit request.
>
> also look at the recommendation from original report
>
> > Consult with lido team to make it possible to deposit specifically to a certain set of operators and calculate the maximum amount depositable to that set at the instant of the call rather than a predefined value
>
> I mean we should assume the LIDO behave correctly, otherwise, we can say all staked ETH are lost...

This comment clearly lacks in understanding the mentioned issue and my earlier comment

The front-running that is in discussion here is the ability of an user to front run the `depositor bot's` transaction with altered amount param while the front running mentioned in the above comment is regarding the ability of a node operator to deposit into beacon chain with different withdrawal credentials hence stealing funds from lido user's

I agree and I have not mentioned this ever

```
user cannot spoof the stakingModuleId as well, which is used to identify the
↪  staking module party
```

The entire deposit request and allocation to obol validators is dependent on this amount ie. how much `amount` is being converted from weth to steth and being deposited

```
also I think the amount parameter just convert WETH to wstETH and not related to
↪  deposit request
```

LIDO is never assumed to behave incorrectly in the report. The only thing mentioned as a limitation on the lido side is regarding the staking module containing both ssv and obol. The main issue mentioned is the incorrectness in the depositing amount which would have to be calculated on-chain inside the function call to be correct and not calculated offchain

```
also look at the recommendation from original report
Consult with lido team to make it possible to deposit specifically to a certain
↪  set of operators and calculate the maximum amount depositable to that set at
↪  the instant of the call rather than a predefined value
I mean we should assume the LIDO behave correctly, otherwise, we can say all
↪  staked ETH are lost...
```

WangSecurity

@10xhash as I understand the problem here is the following:

1. The transaction to call `convertAndDeposit` is submitted with a specific amount X.
2. Before it gets executed, there are other transactions to deposit into Lido are executed.
3. When our txn is executed, the conversion rate of WETH -> stETH has changed (due to txns in step 2), hence, the amount deposited into Lido will change. The result is that not all ETH deposited by Mellow will go to Obol validators.

Is the scenario above correct?

10xhash

> @10xhash as I understand the problem here is the following:
>
> ```
> 1. The transaction to call `convertAndDeposit` is submitted with a specific
> ↪  amount X.
>
> 2. Before it gets executed, there are other transactions to deposit into
> ↪  Lido are executed.
>
> 3. When our txn is executed, the conversion rate of WETH -> stETH has
> ↪  changed (due to txns in step 2), hence, the amount deposited into Lido
> ↪  will change. The result is that not all ETH deposited by Mellow will go
> ↪  to Obol validators.
> ```

> Is the scenario above correct?

No. The eth:steth ratio doesn't change. stETH is always considered 1:1 with eth for deposits (steth balance and not the shares amount)

The issue revolves around how the deposits to beacon chain from lido work. With an example: deposits to beacon chain occur in multiples of 32 eth if current excess eth queued in lido (ie. to be deposited to beacon chain) is 1 eth, then the amount calculated by the admin off-chain would be 31 eth (the calculation of this involves other factors such as the associated limits etc). and if they submit the tx and the value in lido changes to 0 eth (due to queuing of withdrawals in lido) or the amount in lido increases to 2 eth, the passed in value of 31 eth is not correct. in case the amount changes to 0 eth, the 31 eth will be leftover in lido without being allocated to obol or in the other case, 1 eth extra has been converted which will not be allocated to obol. Since here, the amount param is also modifiable by an user they can use it to modify the value to cause the maximum amount of weth to be converted to steth but not allocated to obol validators

WangSecurity

I agree with the comment above and believe the issue should remain as it is. Planning to reject the escalation.

Slavchew

> No. The eth:steth ratio doesn't change. stETH is always considered 1:1 with eth for deposits (steth balance and not the shares amount)
>
> The issue revolves around how the deposits to beacon chain from lido work. With an example: deposits to beacon chain occur in multiples of 32 eth if current excess eth queued in lido (ie. to be deposited to beacon chain) is 1 eth, then the amount calculated by the admin off-chain would be 31 eth (the calculation of this involves other factors such as the associated limits etc). and if they submit the tx and the value in lido changes to 0 eth (due to queuing of withdrawals in lido) or the amount in lido increases to 2 eth, the passed in value of 31 eth is not correct. in case the amount changes to 0 eth, the 31 eth will be leftover in lido without being allocated to obol or in the other case, 1 eth extra has been converted which will not be allocated to obol. Since here, the amount param is also modifiable by an user they can use it to modify the value to cause the maximum amount of weth to be converted to steth but not allocated to obol validators

This answer has nothing valid with how Lido and stETH is working.

Each deposit to beacon chain is when 32 eth are collected (deposited) from numerous deposits to Lido for this staking module and then will be deposited to the beacon chain. There is no thing like "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited to beacon chain and the rest 1 eth will stay for the next stack of 32 for this particular Staking Module.

Each beacon chain deposit is when 32 eth have been collected (deposited) from multiple Lido deposits for this staking module and will then be deposited into the beacon chain. There is no such thing as "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited into the beacon chain and the remaining 1 eth will be left for the next stack of 32 for that particular Staking Module.

`if current excess eth queued in lido is 1 eth, then the amount calculated by the admin off-chain would be 31 eth and if they submit the tx and the value in lido changes to 0 eth or the amount in lido increases to 2 eth, the passed in value of 31 eth is not correct. in case the amount changes to 0 eth, the 31 eth will be leftover in lido without being allocated to obol or in the other case, 1 eth extra has been converted which will not be allocated to obol.` - If this is true, you are assuming that ALWAYS every staking module deposit must be 32, then how will there be 1 eth first if you always have to add up to 32. If there are 2 eth and you deposit 30 eth to add up to 32 eth, according to your idea every subsequent deposit should always be strictly 32 eth, because you said that if there are 0 eth (which will become after you deposit exactly 30 eth to stack to 32 eth) - `in case the amount changes to 0 eth, 31 eth will remain in lido without being allocated to obol`, which is completely wrong.

The idea is to deposit whatever value for this staking module that is associated with the stakingModuleId and when the deposit value stacks up to 32 to deposit it into the beacon chain, if you deposit 35 eth it will deposit once (32 eth) in the beacon chain and the remaining 3 will remain for the next round.

Here is the code where the actual deposit happens, there is no chance of losing eth or someone to allocate on your behalf.
https://github.com/lidofinance/lido-dao/blob/5fcedc6e9a9f3ec154e69cff47c2b9e25503a78a/contracts/0.4.24/Lido.sol#L706-L717

https://github.com/lidofinance/lido-dao/blob/5fcedc6e9a9f3ec154e69cff47c2b9e25503a78a/contracts/0.8.9/StakingRouter.sol#L1118-L1134

here if the stake is less than 32 eth the number of deposits is 0 and no chain beacon deposit will be made so any amount staked by stakingModule in lido that is lower than 32 ether will be left idle in LIDO until a new stake occurs, bringing the depositValue to more than 32 but lower than 64, it is a single deposit and depositCount = 1 in this scenario

The only thing @10xhash has texting is that if 32 eth are not collected in Lido, no Obol deposit will happen, which is true, but that's how it should work. Lido deposits are not strict always deposit 32 eth to them and then to Obol, the values just accumulate.

WangSecurity

> Each deposit to beacon chain is when 32 eth are collected (deposited) from numerous deposits to Lido for this staking module and then will be deposited to the beacon chain. There is no thing like "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited to beacon chain and the rest 1 eth will stay for the next stack of 32 for this particular Staking Module. Each beacon chain deposit is when 32 eth have been collected (deposited) from multiple Lido deposits for this staking module and will then be deposited into the beacon chain. There is no such thing as "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited into the beacon chain and the remaining 1 eth will be left for the next stack of 32 for that particular Staking Module.

I don't see where @10xhash said anything like `if there are 2 eth and you deposit 31 eth, 1 eth will be lost`. I believe what hash said is if there are 2 eth, but you deposited 31 eth, then 1 will be waiting for the next 32 eth deposit or can be used for withdrawals which is not the goal here.

> if current excess eth queued in lido is 1 eth, then the amount calculated by the admin off-chain would be 31 eth and if they submit the tx and the value in lido changes to 0 eth or the amount in lido increases to 2 eth, the passed in value of 31 eth is not correct. in case the amount changes to 0 eth, the 31 eth will be leftover in lido without being allocated to obol or in the other case, 1 eth extra has been converted which will not be allocated to obol. - If this is true, you are assuming that ALWAYS every staking module deposit must be 32, then how will there be 1 eth first if you always have to add up to 32. If there are 2 eth and you deposit 30 eth to add up to 32 eth, according to your idea every subsequent deposit should always be strictly 32 eth, because you said that if there are 0 eth (which will become after you deposit exactly 30 eth to stack to 32 eth) - in case the amount changes to 0 eth, 31 eth will remain in lido without being allocated to obol, which is completely wrong.

Again I disagree with what you believe hash said. I believe the meaning is that the ETH deposited to Lido won't be deposited and will be waiting for other deposits to stack to 32 ETH **OR** can be used to payout the withdrawals. The second is the problem, the protocol wants to deposit the funds to Obol validators, while this report shows how it can be used for withdrawals instead of going to Obol validators (e.g. if there are 0 ETH and you deposit 31 ETH, all that 31 ETH can be used for withdrawals, or if there are 2 ETH and you deposit 31 ETH, 1 will be left over for the next 32 ETH stack or will be used for withdrawals).

So I believe Hash described everything correctly but not in the best way. But the report shows how the deposits from Mellow into Lido may not go to Obol validators in that deposit and can be used for withdrawals, i.e. not allocated to Obol validators.

But @10xhash I've got a question for you. Let's take an example of 2 ETH being in the stack and Mellow depositing 31 ETH. 1 ETH will be leftover waiting for the next round. In the meantime, there's a 1 ETH withdrawal, so that 1 ETH from Mellow will be used for withdrawals. In that case, it doesn't mean that Mellow lost that 1 ETH and they will be able to withdraw 31 ETH if they want and technically Obol validators do get this 1 ETH, cause basically the withdrawer received their 1 ETH (leftover from the deposit) from the buffer, but their 1 stETH was kept for Obol validators. Is it correct?

Because, otherwise, if this 1 ETH was sent to the withdrawer and their 1 stETH was unstaked, then Mellow would basically lose that 1 ETH and wouldn't be able to retrieve it.

Slavchew

> Again I disagree with what you believe hash said. I believe the meaning is that the ETH deposited to Lido won't be deposited and will be waiting for other deposits to stack to 32 ETH OR can be used to payout the withdrawals. The second is the problem, the protocol wants to deposit the funds to Obol validators, while this report shows how it can be used for withdrawals instead of going to Obol validators (e.g. if there are 0 ETH and you deposit 31 ETH, all that 31 ETH can be used for withdrawals, or if there are 2 ETH and you deposit 31 ETH, 1 will be left over for the next 32 ETH stack or will be used for withdrawals).

Yes, you are right. That's exactly what I'm explaining.

> So I believe Hash described everything correctly but not in the best way. But the report shows how the deposits from Mellow into Lido may not go to Obol validators in that deposit and can be used for withdrawals, i.e. not allocated to Obol validators.

They will be queued for this StakingModule due to the `stakingModuleId` and will be deposited into Obol when 32 eth is collected, thus nothing will be lost.

> In that case, it doesn't mean that Mellow lost that 1 ETH and they will be able to withdraw 31 ETH if they want and technically Obol validators do get this 1 ETH, cause basically the withdrawer received their 1 ETH (leftover from the deposit) from the buffer, but their 1 stETH was kept for Obol validators. Is it correct?
>
> Because, otherwise, if this 1 ETH was sent to the withdrawer and their 1 stETH was unstaked, then Mellow would basically lose that 1 ETH and wouldn't be able to retrieve it.

This is incorrect, again because of the `stakingModuleId`.

You can read this conversation with me and the Lido team, they express exactly that. - https://discord.com/channels/761182643269795850/773584934619185154/1265770650417500317

10xhash

> Each deposit to beacon chain is when 32 eth are collected (deposited) from numerous deposits to Lido for this staking module and then will be deposited to the beacon chain. There is no thing like "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited to beacon chain and the rest 1 eth will stay for the next stack of 32 for this particular Staking Module. Each beacon chain deposit is when 32 eth have been collected (deposited) from multiple Lido deposits for this staking module and will then be deposited into the beacon chain. There is no such thing as "if there are 2 eth and you deposit 31 eth, 1 eth will be lost" - No, 32 eth will be deposited into the beacon chain and the remaining 1 eth will be left for the next stack of 32 for that particular Staking Module.

I don't see where @10xhash said anything like `if there are 2 eth and you deposit 31 eth, 1 eth will be lost`. I believe what hash said is if there are 2 eth, but you deposited 31 eth, then 1 will be waiting for the next 32 eth deposit or can be used for withdrawals which is not the goal here.

> if current excess eth queued in lido is 1 eth, then the amount calculated by the admin off-chain would be 31 eth and if they submit the tx and the value in lido changes to 0 eth or the amount in lido increases to 2 eth, the passed in value of 31 eth is not correct. in case the amount changes to 0 eth, the 31 eth will be leftover in lido without being allocated to obol or in the other case, 1 eth extra has been converted which will not be allocated to obol. - If this is true, you are assuming that ALWAYS every staking module deposit must be 32, then how will there be 1 eth first if you always have to add up to 32. If there are 2 eth and you deposit 30 eth to add up to 32 eth, according to your idea every subsequent deposit should always be strictly 32 eth, because you said that if there are 0 eth (which will become after you deposit exactly 30 eth to stack to 32 eth) - in case the amount changes to 0 eth, 31 eth will remain in lido without being allocated to obol, which is completely wrong.

Again I disagree with what you believe hash said. I believe the meaning is that the ETH deposited to Lido won't be deposited and will be waiting for

> other deposits to stack to 32 ETH **OR** can be used to payout the withdrawals. The second is the problem, the protocol wants to deposit the funds to Obol validators, while this report shows how it can be used for withdrawals instead of going to Obol validators (e.g. if there are 0 ETH and you deposit 31 ETH, all that 31 ETH can be used for withdrawals, or if there are 2 ETH and you deposit 31 ETH, 1 will be left over for the next 32 ETH stack or will be used for withdrawals).
>
> So I believe Hash described everything correctly but not in the best way. But the report shows how the deposits from Mellow into Lido may not go to Obol validators in that deposit and can be used for withdrawals, i.e. not allocated to Obol validators.
>
> But @10xhash I've got a question for you. Let's take an example of 2 ETH being in the stack and Mellow depositing 31 ETH. 1 ETH will be leftover waiting for the next round. In the meantime, there's a 1 ETH withdrawal, so that 1 ETH from Mellow will be used for withdrawals. In that case, it doesn't mean that Mellow lost that 1 ETH and they will be able to withdraw 31 ETH if they want and technically Obol validators do get this 1 ETH, cause basically the withdrawer received their 1 ETH (leftover from the deposit) from the buffer, but their 1 stETH was kept for Obol validators. Is it correct?
>
> Because, otherwise, if this 1 ETH was sent to the withdrawer and their 1 stETH was unstaked, then Mellow would basically lose that 1 ETH and wouldn't be able to retrieve it.

I am not sure what is meant by `withdrawer received their 1 ETH (leftover from the deposit) from the buffer, but their 1 stETH was kept for Obol validators`. stETH is a representation for the underlying ETH hence it wouldn't make sense to send ETH out from the lido system and still keep its associated stETH if that helps

> Because, otherwise, if this 1 ETH was sent to the withdrawer and their 1 stETH was unstaked, then Mellow would basically lose that 1 ETH and wouldn't be able to retrieve it

User's only receive ETH if they forego the stETH. Nobody (including mellow) (in general sense) would be loosing their eth by depositing because for every eth deposit made, they will be receiving 1:1 steth which can be redeemed for the underlying amount of eth at any later point of time. The user is able to withdraw ETH now because he holds steth (obtained by making ETH deposit some time back). But here the 1 ETH (which could've been allocated to obol) can end up being used for withdrawals as you have correctly mentioned

Slavchew

And that is until, new 32 eth are collected to go to Obol, thus, you also confirm that there is no issue with this, not the least bit of an impact.

WangSecurity

Thank you for the clarification. Additionally thinking about this issue, we can see that Mellow will not lose tokens, and will not encounter a lock of funds. The impact here is that Mellow might allocate funds to other validators and not Obol. I believe it doesn't qualify for medium severity and indeed should be low. Additionally, this intention is not mentioned in the README.

Hence, planning to accept the escalation and invalidate the report, since allocating funds to other validators is not sufficient for M.

10xhash

> Thank you for the clarification. Additionally thinking about this issue, we can see that Mellow will not lose tokens, and will not encounter a lock of funds. The impact here is that Mellow might allocate funds to other validators and not Obol. I believe it doesn't qualify for medium severity and indeed should be low. Additionally, this intention is not mentioned in the README.
>
> Hence, planning to accept the escalation and invalidate the report, since allocating funds to other validators is not sufficient for M.

`Additionally, this intention is not mentioned in the README`: What should've been mentioned in the README? That deposits should be allocated to Obol/simpleDVTModule? You have to consider the codebase of the project (and the project's high level requirements in itself) when determining what is the expected behavior. The module itself is called simpleDVT module because the project wants to allocate their ETH to that specific module and not just do a conversion to stETH. Apart from this, there are 4 parts from the codebase which clearly demonstrates this:

1. The assets are first kept as ETH itself and only later converted to stETH via the SimpleDVTStakingStrategy. If the intention was to just obtain wstETH, it can be done at the instant of deposit
2. Documentation of the `convertAndDeposit` function having `Converts and deposits into specific staking module` and `The amount of tokens to convert and deposit` https://github.com/sherlock-audit/2024-06-mellow/blob/26aa0445ec405a4ad637bddeeedec4efe1eba8d2/mellow-lrt/src/interfaces/strategies/ISimpleDVTStakingStrategy.sol#L39-L41

```
/**
 * @notice Converts and deposits into specific staking module.
 * @param amount The amount of tokens to convert and deposit.
```

3. bufferedEther >= unfinalizedStETH check is kept so that the mellow deposits are not used up for withdrawals in lido

```
function convertAndDeposit(
  uint256 amount,
  uint256 blockNumber,
  bytes32 blockHash,
  bytes32 depositRoot,
  uint256 nonce,
  bytes calldata depositCalldata,
  IDepositSecurityModule.Signature[] calldata sortedGuardianSignatures
) external onlyDelegateCall {
  if (IERC20(weth).balanceOf(address(this)) < amount)
      revert NotEnoughWeth();

  uint256 unfinalizedStETH = withdrawalQueue.unfinalizedStETH();
  uint256 bufferedEther = ISteth(steth).getBufferedEther();
  if (bufferedEther < unfinalizedStETH)
      revert InvalidWithdrawalQueueState();
```

4. There is a maxAllowedRemainder used during withdrawals so that admin limits the amount of weth that is converted to wstETH directly without ensuring that it is being allocated to obol validators (this has to be done in cases where the admin is unable to process the withdrawal requests with the current wsteth balance of the contract and hence is forced to obtain more wsteth to satisfy withdrawals) https://github.com/sherlock-audit/2024-06-mellow/blob/26aa0445ec405a4ad637bddeeedec4efe1eba8d2/mellow-lrt/src/strategies/SimpleDVTStakingStrategy.sol#L62-L84

```
function processWithdrawals(
    address[] memory users,
    uint256 amountForStake
) external returns (bool[] memory statuses) {

    .....

    statuses = vault.processWithdrawals(users);
    address wsteth = stakingModule.wsteth();
    uint256 balance = IERC20(wsteth).balanceOf(address(vault));
    if (balance > maxAllowedRemainder) revert LimitOverflow();
}
```

WangSecurity

You have to consider the codebase of the project (and the project's high level requirements in itself) when determining what is the expected

> behavior. The module itself is called simpleDVT module because the project wants to allocate their ETH to that specific module and not just do a conversion to stETH. Apart from this, there are 4 parts from the codebase which clearly demonstrates this

I consider the design and the intention of the protocol. I didn't say in any way that you're incorrect or the protocol intention is not what you describe.

> The assets are first kept as ETH itself and only later converted to stETH via the SimpleDVTStakingStrategy. If the intention was to just obtain wstETH, it can be done at the instant of deposit Documentation of the convertAndDeposit function having Converts and deposits into specific staking module and The amount of tokens to convert and deposit

Again, I agree that the intention is clear and the protocol wants to deposit to specific validators. And for points 3 and 4 again, I don't mean to say it's not the intention to allocate deposits to obol validators.

What I mean is that I don't see this issue qualifying for Medium severity, it doesn't cause loss of funds to Mellow protocol and users and doesn't break core contract functionality, rendering the contract useless:

> The amount to be staked for obol validators can be assigned to other operators

Of course, if my assumption is incorrect, please let me know. But the current decision remains the same, planning to accept the escalation and invalidate the issue.

JeffCX

> What I mean is that I don't see this issue qualifying for Medium severity, it doesn't cause loss of funds to Mellow protocol and users and doesn't break core contract functionality, rendering the contract useless:

then the action should be planning to accept the escalation and invalidate the report..

not sure why even with the comments above, the decision changes to

> But the current decision remains the same, planning to reject the escalation and leave the issue as it is.

10xhash

> > What I mean is that I don't see this issue qualifying for Medium severity, it doesn't cause loss of funds to Mellow protocol and users and doesn't break core contract functionality, rendering the contract useless:

> then the action should be planning to accept the escalation and invalidate the report..
>
> not sure why even with the comments above, the decision changes to
>
> > But the current decision remains the same, planning to reject the escalation and leave the issue as it is.

That would be a typo and the intention of his message is to not have this as a valid med

10xhash

> > You have to consider the codebase of the project (and the project's high level requirements in itself) when determining what is the expected behavior. The module itself is called simpleDVT module because the project wants to allocate their ETH to that specific module and not just do a conversion to stETH. Apart from this, there are 4 parts from the codebase which clearly demonstrates this
>
> I consider the design and the intention of the protocol. I didn't say in any way that you're incorrect or the protocol intention is not what you describe.
>
> > The assets are first kept as ETH itself and only later converted to stETH via the SimpleDVTStakingStrategy. If the intention was to just obtain wstETH, it can be done at the instant of deposit Documentation of the convertAndDeposit function having Converts and deposits into specific staking module and The amount of tokens to convert and deposit
>
> Again, I agree that the intention is clear and the protocol wants to deposit to specific validators. And for points 3 and 4 again, I don't mean to say it's not the intention to allocate deposits to obol validators.
>
> What I mean is that I don't see this issue qualifying for Medium severity, it doesn't cause loss of funds to Mellow protocol and users and doesn't break core contract functionality, rendering the contract useless:
>
> > The amount to be staked for obol validators can be assigned to other operators
>
> Of course, if my assumption is incorrect, please let me know. But the current decision remains the same, planning to reject the escalation and leave the issue as it is.

It breaks the core contract functionality. The whole point of the entire staking module is to allocate specifically to obol validators and this breaks it. The ability of the user to change the amount to whatever he wants can make the strategy useless

WangSecurity

As you quoted in the previous comment, the documentation of the `convertAndDeposit` is `Converts and deposits into specific staking module` and this functionality is not broken and the contract converts and deposits into the staking module. The staking module documentation is `Converts wETH to wstETH and securely deposits it into the staking contract according to the specified security protocols` and indeed the protocol deposits and converts it into the staking contracts. The intention to deposit specifically to Obol validators is broken here, but this intention was introduced in a discord message by the sponsor. However, the staking contract still deposits funds into the specified protocols (Lido in this case), so the functionality is not broken.

That is why I believe the contract is not useless. Of course, you can correct me. But, for now, the decision remains the same, planning to accept the escalation and invalidate the report.

10xhash

> As you quoted in the previous comment, the documentation of the `convertAndDeposit` is `Converts and deposits into specific staking module` and this functionality is not broken and the contract converts and deposits into the staking module. The staking module documentation is `Converts wETH to wstETH and securely deposits it into the staking contract according to the specified security protocols` and indeed the protocol deposits and converts it into the staking contracts. The intention to deposit specifically to Obol validators is broken here, but this intention was introduced in a discord message by the sponsor. However, the staking contract still deposits funds into the specified protocols (Lido in this case), so the functionality is not broken.
>
> That is why I believe the contract is not useless. Of course, you can correct me. But, for now, the decision remains the same, planning to accept the escalation and invalidate the report.

It doesn't deposit the amount into the specific staking module. If your understanding is based on some of the above comments like `1 eth will be left for the next stack of 32 for that particular Staking Module` , it is incorrect. The converted assets are just deposited into lido and is not tied with the particular staking module etc. The amount can be allocated to other modules or can be used for withdrawals. A deposit to a staking module would happen when eth is allocated to any operator belonging to the staking module set.

WangSecurity

Then I admit a mistake on my side and misunderstanding.

> If your understanding is based on some of the above comments like 1 eth

> will be left for the next stack of 32 for that particular Staking Module , it is incorrect

I understand it's incorrect and it wasn't what I thought. I misunderstood what you mean by the staking module. To finally clarify, Mellow has `SimpleDVTStakingStrategy` which is responsible for depositing funds into Lido and allocating them to Obol validators, i.e. `deposit into specific staking module`. Also, Mellow has `StakingModule` which deposits into specific security protocols.

The problem is in the first since it doesn't ensure enough of the funds are indeed deposited into that specific staking module and funds can be sent to another staking module. The issue is that the `amount` value is pre-calculated before, but due to deposits that get processed before the `convertAndDeposit` call, the sent amount can be left for withdrawals or end up in different staking modules, correct?

10xhash

> Then I admit a mistake on my side and misunderstanding.
>
> > If your understanding is based on some of the above comments like 1 eth will be left for the next stack of 32 for that particular Staking Module , it is incorrect
>
> I understand it's incorrect and it wasn't what I thought. I misunderstood what you mean by the staking module. To finally clarify, Mellow has `SimpleDVTStakingStrategy` which is responsible for depositing funds into Lido and allocating them to Obol validators, i.e. `deposit into specific staking module`. Also, Mellow has `StakingModule` which deposits into specific security protocols.
>
> The problem is in the first since it doesn't ensure enough of the funds are indeed deposited into that specific staking module and funds can be sent to another staking module. The issue is that the `amount` value is pre-calculated before, but due to deposits that get processed before the `convertAndDeposit` call, the sent amount can be left for withdrawals or end up in different staking modules, correct?

`Also, Mellow has StakingModule which deposits into specific security protocols.` : Mellows StakingModule is just a contract underneath the simpleDVTStaking strategy that helps (the vault delegate call's the code) in putting the funds into lido with the intention to have the funds allocated to the simpleDVT module of lido

Yes. Not just pre-calculated before, a user has the ability to front-run and change the amount to whatever they want. And apart from deposits, withdrawals can also change the correctness of the `amount`

WangSecurity

In that sense, I again want to admin my mistake and misunderstanding. Hash is correct, indeed this vulnerability leads to the entire strategy, hence, the `SimpleDVTStakingStrategy` contract being useless, since it won't deposit into the specified staking module. Therefore, it qualifies for the Medium severity:

> Breaks core contract functionality, rendering the contract useless

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

blckhv

I think you're missing the fact that `depositBufferedEther` is always being called in a private transaction (hash’s msg in LIDO discord), so how the frontrunning is even possible then?

> Yes. Not just pre-calculated before, a user has the ability to front-run and change the amount to whatever they want. And apart from deposits, withdrawals can also change the correctness of the amount

10xhash

> I think you're missing the fact that `depositBufferedEther` is always being called in a private transaction (hash’s msg in LIDO discord), so how the frontrunning is even possible then?
>
> > Yes. Not just pre-calculated before, a user has the ability to front-run and change the amount to whatever they want. And apart from deposits, withdrawals can also change the correctness of the amount

Please see the earlier message here https://github.com/sherlock-audit/2024-06-mellow-judging/issues/260#issuecomment-2236241395 and also verify that the `depositBufferedEther` has been invoked via public mempools since the start https://etherscan.io/address/0xC77F8768774E1c9244BEed705C4354f2113CFc09 and also this message for more context https://github.com/sherlock-audit/2024-06-mellow-judging/issues/265#issuecomment-2233993493

WangSecurity

Yep, even if it's pre-calculated correctly, at the time of the transaction being executed, the `amount` can be inaccurate. This can happen with private transactions as well. Moreover, there were no mentions of the private transaction by the sponsor as well as flashbots.

Hence, the private transactions won't mitigate this issue. The decision remains the same, planning to reject the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- ctf-sec: rejected

10xhash

Fixed Now the amount is calculated inside the module and it is ensured that >= converted assets are deposited

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Issue M-4: Corrupted oracle system if more than 2 underlying tokens are used and one of them is WSTETH

Source: https://github.com/sherlock-audit/2024-06-mellow-judging/issues/266

The protocol has acknowledged this issue.

Found by

eeyore

Summary

The `priceX96()` function always enforces a cross-oracle price check, even when a direct price feed is provided by Chainlink.

Vulnerability Detail

There is no option to use direct price feeds like RETH/ETH or USDC/ETH without an extra step using `ConstantAggregatorV3` or another Chainlink price feed. The cross-oracle price check is required for WSTETH/STETH and then STETH/ETH price check but should not be enforced for every underlying token used in the protocol.

The current system works with only one underlying token, WSTETH, but with the introduction of, for example, rETH as a second underlying token, there is no option to correctly configure the current Oracle system for both tokens to be denominated in the same base token.

In such a scenario, if ETH is used as the base token, the price feed for stETH/ETH needs to be used, or `ConstantAggregatorV3` with a constant 10**18, which will lead to price corruption and indicate that 1 stETH = 1 ETH.

Impact

There is no possibility to use the current oracle system and its `priceX96()` function correctly if there is more than one underlying token and one of those tokens is WSTETH.

Code Snippet

https://github.com/sherlock-audit/2024-06-mellow/blob/main/mellow-lrt/src/oracles/ChainlinkOracle.sol#L89-L98 https://github.com/sherlock-audit/2024-06-mellow/blob/main/mellow-lrt/src/oracles/WStethRatiosAggregatorV3.sol#L16 https://github.com/sherlock-audit/2024-06-mellow/blob/main/mellow-lrt/src/oracles/ConstantAggregatorV3.sol#L20

Tool used

Manual Review

Recommendation

There should be two different options to acquire the price for a token: direct and cross-oracle. The system should be flexible to use them in parallel. A good example is Euler with its Base and Cross adapters.

Discussion

0xklapouchy

@z3s

As Vault.sol is designed to support more than one underlying token, and given that [rETH, USDC, USDT] are in scope, there is a possibility of adding new tokens as underlying tokens alongside wstETH.

It is logical to assume that if support for multiple underlying tokens exists, the current oracle system should accommodate this. However, it does not.

As pointed out in the issue, there is no way to configure the current oracle system to support multiple underlying tokens, where one of them is wstETH.

Therefore, the only way for the system to function correctly is to have only wstETH as an underlying token.

I see this issue as a medium risk because any action taken by the admin to introduce a new token into a Vault with wstETH will lead to incorrect behavior. In the live contracts, the `addToken()` function should not be touched.

More or less, what this issue indicates is that, even though Vault.sol looks like it supports multiple underlying tokens, in fact, this part of the system is broken in many places, one of which is the oracle system.

Senya123

Escalate.

Escalating this issue on behalf of the submitter. Please review the above comments.

sherlock-admin3

> Escalate.
>
> Escalating this issue on behalf of the submitter. Please review the above comments.

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.

InfectedIsm

If I understand correctly, this means that for this issue to be valid, https://github.com/sherlock-audit/2024-06-mellow-judging/issues/197#issuecomment-2230922991 must be validated too ?

As #266 rely on the fact that 1 stETH != 1 ETH

WangSecurity

> As Vault.sol is designed to support more than one underlying token, and given that [rETH, USDC, USDT] are in scope, there is a possibility of adding new tokens as underlying tokens alongside wstETH

Do I understand correctly that you have this assumption based on the code, is it somewhere in the docs?

0xklapouchy

@WangSecurity

Let me rephrase it: As Vault.sol `can support more than one underlying token`, and given that [rETH, USDC, USDT] are in scope, there is a possibility of adding new tokens `(by admin)` as underlying tokens alongside wstETH

```
/// @notice Adds a new token to the list of underlying tokens in the vault.
/// @dev Only accessible by an admin.
/// @param token The address of the token to add.
function addToken(address token) external;
```

That was the initial meaning, I can see now it could be misinterpreted.

IIIIHunterIIII

this is the comment by the sponsor in discord about oracles problems

0xklapouchy

@IIIIHunterIIII

I assume you are referring to this, as you linked to the incorrect message: `we assume 1 steth == 1 weth, at least for current deployments`

To showcase why this is Medium, I will refer to the following:

```
(External) Admin trust assumptions: When a function is access restricted, only
↪  values for specific function variables mentioned in the README can be taken
↪  into account when identifying an attack path.

If no values are provided, the (external) admin is trusted to use values that
↪  will not cause any issues.

Note: if the attack path is possible with any possible value, it will be a valid
↪  issue.
```

And to the currently live code that `assume 1 steth == 1 weth`:

Mellow currently has multiple live Vaults with a combined TVL of around $550M+. They are using wstETH as the underlying asset. Any action on `addToken()` at this point leads to the corruption of the system. ANY addition will lead to protocol misconfiguration and users losing money.

IIIIHunterIIII

quoting this from the submission,

> but with the introduction of, for example, rETH as a second underlying token, there is no option to correctly configure the current Oracle system for both tokens to be denominated in the same base token.

where is the corruption coming from when adding another underlying?

sorry but i think there is something missing

0xklapouchy

The current ChainlinkOracle.sol always forces you to determine the base token, and the priceX96() will be based on its price. This requirement forces you to always use cross price querying Z/Y -> Y/X.

Preconditions:

- The base token is X (it can be WETH or USDC; querying ETH/USD directly is not possible).
- The vault wants to use two underlying tokens, Y and Z.

Let's consider we want X to be WETH and query the price in USD. We set X to use the ETH/USD price feed. At this point, if any of the Y or Z tokens don't have Y/X or Z/X price feeds, we can't use them in combination. This appears to be a misconfiguration by the admin, since the admin is trusted, it should not be an issue.

Now, let's look at a real scenario with rETH and wstETH, where we want to price them in ETH:

1. If we want to set up rETH first, we need to use the rETH/ETH oracle and set the base token as WETH. Since we don't have a WETH/ETH Chainlink price feed, we will use `ConstantAggregatorV3` for a 1-to-1 conversion. If we then try to set up wstETH, we can't do it correctly. There is no wstETH/ETH Chainlink oracle, and if we use `WStethRatiosAggregatorV3`, our return price will be in stETH, which will then be converted by `ConstantAggregatorV3` for a 1-to-1 conversion to ETH.
2. If we want to set up wstETH first, we need to use `WStethRatiosAggregatorV3` and set the base token as stETH. There is a stETH/ETH Chainlink price feed, which we will use to query the price for the base token. If we then try to set up rETH, we need to use the rETH/ETH oracle. Since there is no rETH/stETH oracle, our price will be returned in ETH, and querying it against the stETH/ETH price will give an incorrect price.
3. Now, we have a corner case from live contracts:

- WETH is set as the base token, with `ConstantAggregatorV3` as the address to query its price.
- wstETH is set in `ChainlinkOracle` with `WStethRatiosAggregatorV3` as the address to query its price.

Such a setup can be correct, but it needs to block any future interaction with addToken(). Since it does not block interaction with addToken(), any interaction will break the system. As this corner case is possible and any interaction by the admin with the trusted function leads to breaking the system, I see this as a Medium issue.

IIIIIHunterIIII

okey, thx so much for clarification, i think this should be my last comment `if the attack path is possible with any possible value, it will be a valid issue.` i think i understand now where the problem coming from, the problem here is the fact that any act from admin will cause the issue. (always corrupted design)

but isn't this statement `assume 1 steth == 1 weth` from sponsor totally accepts the risk associated with using `steth -> weth` interchangeably during the interactions with oracles, cause from what i see from the above comment show the overhead of dealing with those discrepancies through 2 oracles price retrievals.

0xklapouchy

As I mentioned in another comment on a different issue, we are now trying to find all possible edge cases where the system works, and I can confirm it will work with a single underlying token.

However, the issue is not about a single underlying token but the possibility of adding a new token and how the system reacts to it.

WangSecurity

I agree with the escalation, indeed the protocol wants to integrate 2 underlying tokens in their system, while this report shows how it would break the system. Planning to accept the escalation and leave the issue as it is.

WangSecurity

Result: Medium Unique

sherlock-admin4

Escalations have been resolved successfully!

Escalation status:

- Senya123: accepted

WangSecurity

@z3s @0xklapouchy are there any duplicates?

Slavchew

@WangSecurity The discussion above shows highly misunderstanding of the codebase.

> The current ChainlinkOracle.sol always forces you to determine the base token, and the priceX96() will be based on its price. This requirement forces you to always use cross price querying Z/Y -> Y/X.
>
> Preconditions:
>
> The base token is X (it can be WETH or USDC; querying ETH/USD directly is not possible). The vault wants to use two underlying tokens, Y and Z. Let's consider we want X to be WETH and query the price in USD. We set X to use the ETH/USD price feed. At this point, if any of the Y or Z tokens don't have Y/X or Z/X price feeds, we can't use them in combination. This appears to be a misconfiguration by the admin, since the admin is trusted, it should not be an issue.
>
> Now, let's look at a real scenario with rETH and wstETH, where we want to price them in ETH:
>
> If we want to set up rETH first, we need to use the rETH/ETH oracle and set the base token as WETH. Since we don't have a WETH/ETH Chainlink price feed, we will use ConstantAggregatorV3 for a 1-to-1 conversion. If we then try to set up wstETH, we can't do it correctly. There is no wstETH/ETH Chainlink oracle, and if we use WStethRatiosAggregatorV3, our return price will be in stETH, which will then be converted by ConstantAggregatorV3 for a 1-to-1 conversion to ETH.
>
> If we want to set up wstETH first, we need to use WStethRatiosAggregatorV3 and set the base token as stETH. There is a

> stETH/ETH Chainlink price feed, which we will use to query the price for the base token. If we then try to set up rETH, we need to use the rETH/ETH oracle. Since there is no rETH/stETH oracle, our price will be returned in ETH, and querying it against the stETH/ETH price will give an incorrect price.
>
> Now, we have a corner case from live contracts:
>
> WETH is set as the base token, with ConstantAggregatorV3 as the address to query its price. wstETH is set in ChainlinkOracle with WStethRatiosAggregatorV3 as the address to query its price. Such a setup can be correct, but it needs to block any future interaction with addToken(). Since it does not block interaction with addToken(), any interaction will break the system. As this corner case is possible and any interaction by the admin with the trusted function leads to breaking the system, I see this as a Medium issue.

This is incorrect, the base token will be WETH and therefore all listed tokens have a token/ETH price feed.

Can be seen in the scripts and also said by the sponsor.

https://github.com/sherlock-audit/2024-06-mellow/blob/26aa0445ec405a4ad637bddeeedec4efe1eba8d2/mellow-lrt/scripts/mainnet/DeployScript.sol#L126-L128

> If we want to set up rETH first, we need to use the rETH/ETH oracle and set the base token as WETH. Since we don't have a WETH/ETH Chainlink price feed, we will use ConstantAggregatorV3 for a 1-to-1 conversion. If we then try to set up wstETH, we can't do it correctly. There is no wstETH/ETH Chainlink oracle, and if we use WStethRatiosAggregatorV3, our return price will be in stETH, which will then be converted by ConstantAggregatorV3 for a 1-to-1 conversion to ETH.

If referring to this explanation, wstETH can be set exactly with WStethRatiosAggregatorV3 and this statement only shows that the price of wstETH will be taken as stETH, identifying that stETH : ETH is not 1:1, which is already reported and invalidated, So if this report is valid it must be a duplicate of `stETH:ETH is not 1:1` issues because the rest of the explanation about the denomination on priceFeed is incorrect.

0xklapouchy

@Slavchew, you can have a perfectly fine system if you use ONE underlying token, `wstETH`, and the `stETH:ETH ratio is not 1:1` is not a problem. Most of the live contracts are set up like this, as in such cases the real base token is `stETH`, and the system doesn't even use Chainlink oracles.

This issue explains exactly how the oracle system is broken when you combine multiple underlying tokens where one of them is `wstETH`, and why you can't configure it correctly. It also shows why in current live contracts, using the addToken() function will break the system.

Most of the Wardens (including myself in #126) noted the issue `stETH:ETH is not 1:1`, just in case, but no one has explained how this can break a system exactly.

@WangSecurity, at this moment it is unique, but if you combine it with `stETH:ETH is not 1:1`, then it is not. It is up to you to decide if all issues with `stETH:ETH is not 1:1` have sufficient explanations of the problem.

Slavchew

> @Slavchew, you can have a perfectly fine system if you use ONE underlying token, wstETH

Wrong, adding more tokens will work fine because all the tokens announced by the sponsor (wstETH, rETH, USDC, USDT) have a token/ETH price feed.

> Most of the live contracts are set up like this, as in such cases the real base token is stETH, and the system doesn't even use Chainlink oracles.

Wrong, like I said, the base token IS WETH, not stETH. This is evident from my comment above, stated by the sponsor. And it can also be seen in the deployed contracts that the configured baseToken is WETH.

Address of the PriceOracle.sol (ChainlinkOracle.sol now) - https://etherscan.io/address/0x1Dc89c28e59d142688D65Bd7b22C4Fd40C2cC06d#readContract#F3

Address of the Vault.sol - https://etherscan.io/address/0xBEEF69Ac7870777598A04B2bd4771c71212E6aBc

Result - https://etherscan.io/address/0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2

Another thing, it's just imagination if a different baseToken is used, BUT as strictly stated in the readme, the deployment scripts contain the values that are correct and will be used.

> Q: Are there any limitations on values set by admins (or other roles) in the codebase, including restrictions on array lengths? Only the parameters from the deployment script below: https://github.com/mellow-finance/mellow-lrt/blob/dev-symbiotic-deploy/scripts/mainnet/Deploy.s.sol#L9

Having clarified this again, there is no actual fact-based comment showing the use of other baseTokens, only if you have discovered that I will wait to discuss, but until then the problem is only highly hypothetical and does not meet Sherlock's rules of possible "Future Issues"

> Future issues: Issues that result out of a future integration/implementation that was not mentioned in the docs/README

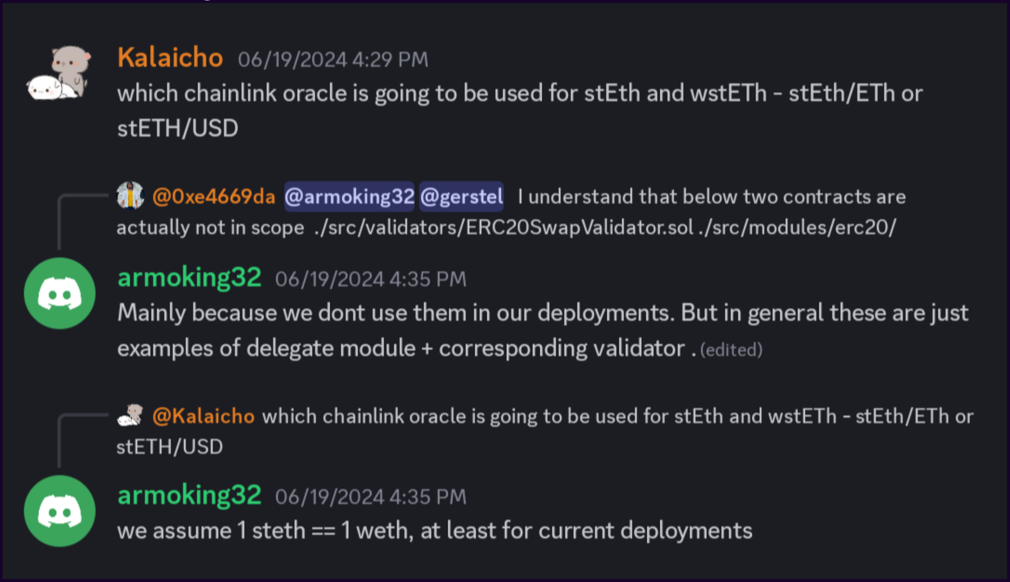
> or because of a future change in the code (as a fix to another issue) are not valid issues.
>
> ---
>
> Most of the Wardens (including myself in https://github.com/sherlock-audit/2024-06-mellow-judging/issues/126) noted the issue stETH:ETH is not 1:1, just in case, but no one has explained how this can break a system exactly.
>
> It is up to you to decide if all issues with stETH:ETH is not 1:1 have sufficient explanations of the problem.

Regarding your state that `stETH:ETH is not 1:1`, I also agree with that, but that is not the explanation here, here you are mainly focusing on a different baseToken. In case @WangSecurity decides to accept any of the `stETH:ETH is not 1:1` issues as valid, yes this should be a duplicate. But the explanation of the baseToken problem is completely wrong as I already said above.

@WangSecurity I think that's enough to put your final words on the issue.

IIIIHunterIIII

is this message have been considered?



Kalaicho 06/19/2024 4:29 PM
which chainlink oracle is going to be used for stEth and wstETh - stEth/ETh or stETH/USD

@0xe4669da @armoking32 @gerstel I understand that below two contracts are actually not in scope ./src/validators/ERC20SwapValidator.sol ./src/modules/erc20/

armoking32 06/19/2024 4:35 PM
Mainly because we dont use them in our deployments. But in general these are just examples of delegate module + corresponding validator . (edited)

@Kalaicho which chainlink oracle is going to be used for stEth and wstETh - stEth/ETh or stETH/USD

armoking32 06/19/2024 4:35 PM
we assume 1 steth == 1 weth, at least for current deployments

Slavchew

@IIIIHunterIIII

This is not appropriate for this discussion and you have already commented this on it. Please refrain from such comments because they do not help in any case.

0xklapouchy

> Wrong, adding more tokens will work fine because all the tokens announced by the sponsor (wstETH, rETH, USDC, USDT) have a token/ETH price feed.

You are wrong here, in current live code and in code in audit, adding any of (rETH, USDC, USDT) to the system that have `wstETH` as underlying will brick the system. (Not to mention USDC, USDT decimal problem).

> Wrong, like I said, the base token IS WETH, not stETH.

You missed the `real` part, `the real base token is stETH`. @IIIIHunterIIII This is exactly this assumption that protocol have 1 steth == 1 weth.

In current live code the `addToken()` function will break the system, you can add any token that have token/ETH price feed and you still will not be able to configure current oracle system correctly to work with `wstETH`.

Slavchew

> > Wrong, adding more tokens will work fine because all the tokens announced by the sponsor (wstETH, rETH, USDC, USDT) have a token/ETH price feed.
>
> You are wrong here, in current live code and in code in audit, adding any of (rETH, USDC, USDT) to the system that have `wstETH` as underlying will brick the system. (Not to mention USDC, USDT decimal problem).
>
> > Wrong, like I said, the base token IS WETH, not stETH.
>
> You missed the `real` part, `the real base token is stETH`. @IIIIHunterIIII This is exactly this assumption that protocol have 1 steth == 1 weth.
>
> In current live code the `addToken()` function will break the system, you can add any token that have token/ETH price feed and you still will not be able to configure current oracle system correctly to work with `wstETH`.

No examples and no facts, you just say you can't add new tokens and that the base token is stETH, which is wrong and you can't prove it. As I said I am open to discussion, but if you give facts where and how it is written that stETH is a base token, until then everything written in the issue is wrong.

0xklapouchy

> 3. Now, we have a corner case from live contracts:
>
>   - WETH is set as the base token, with `ConstantAggregatorV3` as the address to query its price.
>   - wstETH is set in `ChainlinkOracle` with `WStethRatiosAggregatorV3` as the address to query its price.

@Slavchew I think you did not read my explanations.

Here are the tx from live code for the Steakhouse vault:

1. https://etherscan.io/tx/0x5dbd039e9774bbe28cdf52b55c9657381bc042763db99ffd3f0aacfd1bc1d0fa

WETH - set as base token.

2. https://etherscan.io/tx/0x21f4cc7f85a9896800a545b8ba3d5e74b71d7c51525e8942c9707693df477822

Oracle configured as described. No use of real Chainlink price feed.

aggregatorV3 for WETH used:
0x6A8d8033de46c68956CCeBA28Ba1766437FF840F (ConstantAggregatorV3)
aggregatorV3 for wstETH used:
0x94336dF517036f2Bf5c620a1BC75a73A37b7bb16 (WStethRatiosAggregatorV3)

Please reread the https://github.com/sherlock-audit/2024-06-mellow-judging/issues/266#issuecomment-2235958848

The case 1 and case 2 are the possible approaches how admin can start configuring the oracle system, and why he is unable to do it.

Case 3 is real configuration of the Mellow Steakhouse vault, and it is showing why addToken() will brake the system.

Slavchew

> > 3. Now, we have a corner case from live contracts:
> >
> > - WETH is set as the base token, with `ConstantAggregatorV3` as the address to query its price.
> > - wstETH is set in `ChainlinkOracle` with `WStethRatiosAggregatorV3` as the address to query its price.
>

> @Slavchew I think you did not read my explanations.
>
> Here are the tx from live code for the Steakhouse vault:
>
> 1. https://etherscan.io/tx/0x5dbd039e9774bbe28cdf52b55c9657381bc042763db99ffd3f0aacfd1bc1d0fa
>
> WETH - set as base token.
>
> 2. https://etherscan.io/tx/0x21f4cc7f85a9896800a545b8ba3d5e74b71d7c51525e8942c9707693df477822
>
> Oracle configured as described. No use of real Chainlink price feed.
>
> aggregatorV3 for WETH used:
> 0x6A8d8033de46c68956CCeBA28Ba1766437FF840F

> (ConstantAggregatorV3) aggregatorV3 for wstETH used: 0x94336dF517036f2Bf5c620a1BC75a73A37b7bb16 (WStethRatiosAggregatorV3)
>
> Please reread the #266 (comment)
>
> The case 1 and case 2 are the possible approaches how admin can start configuring the oracle system, and why he is unable to do it.
>
> Case 3 is real configuration of the Mellow Steakhouse vault, and it is showing why addToken() will brake the system.

Indeed you also confirm that the baseToken is WETH, wstETH is the underlying and of course WStethRatiosAggregatorV3 is used.

By this comment, you agree that the baseToken is WETH and only showing the aggregators linked to different tokens which is known to me.The base token is WETH, wstETH is underlying using WStethRatiosAggregatorV3()

> No use of real Chainlink price feed. Yes, there is no WETH/ETH priceFeed.

That being said, I'm stopping responding to your comments as none of them provide facts and an example of where and when baseToken will be stETH as you stated - `"Most of the live contracts are set up like this, as in such cases the real base token is stETH"`. You just keep throwing out comments without actual facts and examples, only increasing the complexity and time of escalations, as they are a place to give you facts, not assumptions without understandings.

@WangSecurity I hope this https://github.com/sherlock-audit/2024-06-mellow-judging/issues/266#issuecomment-2249910713 of mine is the last to provide facts, that accepting a different `baseToken` is not specified anywhere and not true, I will only answer to you if have questions or if I can help you understand the situation. Thank you.

WangSecurity

I agree with @Slavchew arguments. Indeed, WETH is used as a base token, we don't have any information in the README and code comments that there will be other base tokens, only the script linked in the README which shows WETH is used as the base token every time. Based on @Slavchew comments I don't think there will be any problems with adding more underlying tokens and the admin can set custom oracles in ChainlinkOracle::setChainlinkOracles, since as I understand `ConstantAggregator` and `WStethRatiosAggregatorV3` are specific to WETH and wstETH, respectively. I won't repeat the arguments exactly since there is no point in repeating the same information.

Hence, I'm planning to revert the decision and invalidate this issue. @0xklapouchy if still disagree and believe there will be an issue, I'll need you to show a coded POC or an extremely detailed written POC with exact functions and values that would cause an issue. I'll revert the decision on Sunday morning (CEST timezone).

0xklapouchy

@WangSecurity

As mentioned in the issue description, a potential scenario in the Mellow protocol involves a Vault with multiple underlying tokens. Let's consider a Vault with `[wstETH, rETH]`.

The issue arises in an edge case where `wstETH` is the first underlying token added. Any subsequent use of the `addToken()` function leads to incorrect operation of the Oracle system.

There is no mechanism to prevent the Admin from using the `addToken()` function, and its use results in a loss of funds.

> If no values are provided, the (external) admin is trusted to use values that will not cause any issues.
> Note: if the attack path is possible with any value, it is a valid issue.

Example Scenario

Now, let's consider a live Vault, `Steakhouse`: `0xBEEF69Ac7870777598A04B2bd4771c71212E6aBc`. We'll add `rETH` as a second underlying token and configure it to have a 50/50 ratio with `wstETH` for deposits and withdrawals. After the setup, a user will deposit tokens worth 100 ETH into the Vault.

- **Vault**: `0xBEEF69Ac7870777598A04B2bd4771c71212E6aBc`
- **Configurator**: `0xe6180599432767081beA7deB76057Ce5883e73Be`
- **Ratios Oracle**: `0x955Ff4Cc738cDC009d2903196d1c94C8Cfb4D55d`
- **Price Oracle**: `0x1Dc89c28e59d142688D65Bd7b22C4Fd40C2cC06d`
- **rETH**: `0xae78736Cd615f374D3085123A210448E74Fc6393`

Following the deployment script steps:

1. **Add rETH as a new underlying token.**

```
vault.addToken(rETH);
```

2. **Update the ratios for deposits and withdrawals.**

```
uint256[2] memory ratiosX96;
ratiosX96[0] = 2 ** 96 / 2;
ratiosX96[1] = 2 ** 96 / 2;
```

```
ratiosOracle.updateRatios(vault, true, ratiosX96);
ratiosOracle.updateRatios(vault, false, ratiosX96);
```

3. **Update the price oracle.**

```
address[] memory tokens = new address[](2);
tokens[0] = weth; // 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
tokens[1] = rETH; // 0xae78736Cd615f374D3085123A210448E74Fc6393

IChainlinkOracle.AggregatorData[] memory data = new
↪  IChainlinkOracle.AggregatorData[](2);
data[0].aggregatorV3 = 0x6A8d8033de46c68956CCeBA28Ba1766437FF840F; //
↪  ConstantAggregatorV3
data[0].maxAge = 0;
data[1].aggregatorV3 = 0x536218f9E9Eb48863970252233c8F271f554C2d0; //
↪  rETH/ETH Chainlink price feed
data[1].maxAge = 86400; // 24h

priceOracle.setChainlinkOracles(vault, tokens, data);
```

4. **User deposits 50 wstETH and 50 rETH:**

Based on the Mellow.finance page, the Vault currently has 51,528 wstETH, used to calculate the current TVL. As of 26.07.2024, 51,528 wstETH 60,487.1086820949 stETH 60,440.6545826271 ETH.

```
vault.deposit(user, [50000000000000000000, 50000000000000000000], minLpAmount,
↪  deadline); // user address, minLpAmount, and deadline are not important
```

Calculation in the Deposit Function

Current Oracle System:

- **priceX96 (wstETH)**: `((1173868744800786254 * 10**18) * 2**96) / (10**18 * 10**18) = 93003463683492183526026379957`
  - Overvalued as it should be in WETH but it is in stETH since `WStethRatiosAggregatorV3` is used, base token is WETH.
- **priceX96 (rETH)**: `((1119033350396089900 * 10**18) * 2**96) / (10**18 * 10**18) = 88658956144063119450293135167`
  - Correctly returned as WETH.

Total Value:

```
totalValue = (51528000000000000000000 * 93003463683492183526026379957) / 2**96 +
↪  0
           = 60487108682094914096111
            60487.1086820949 WETH (but overvalued)
```

Deposit Value:

```
depositValue = (50000000000000000000 * 93003463683492183526026379957 / 2**96) +
               (50000000000000000000 * 88658956144063119450293135167 / 2**96)
             = 114645104759843807698 (rETH deposit undervalued)
```

Current Total Supply:

```
currentTotalSupply = 13137111513365174591322
```

LP Amount:

```
lpAmount = 114645104759843807698 * 13137111513365174591322 /
↪  60487108682094914096111
         = 24899611809967870953
          24.899611809967870953 LP
```

Correct Oracle System:

- **stETH / ETH price from roundId 18446744073709552703**: 999232162227420800
- **priceX96 (wstETH)**: (((1173868744800786254 * 999232162227420800) / 10**18) * 10**18 * 2**96) / (10**18 * 10**18) = 92932052111095300295443068851
  - Correctly returned as WETH.
- **priceX96 (rETH)**: ((1119033350396089900 * 10**18 * 2**96) / (10**18 * 10**18) = 88658956144063119450293135167)
  - Correctly returned as WETH.

Total Value:

```
totalValue = (51528000000000000000000 * 92932052111095300295443068851) / 2**96 +
↪  0
           = 60440664395294698300127
```

```
            = 60440.664395294698300127
```

Deposit Value:

```
depositValue = (50000000000000000000 * 92932052111095300295443068851 / 2**96) +
               (50000000000000000000 * 88658956144063119450293135167 / 2**96)
             = 114600037721728398798
```

LP Amount:

```
lpAmount = 114600037721728398798 * 13137111513365174591322 /
↪  60440664395294698300127
         = 24908949794790898821
          24.908949794790898821 LP
```

Difference in LP Amounts

The difference in LP amounts is:

```
9337984823027868 = 0.009337984823027868 LP
                 = 9337984823027868 * 60487108682094914096111 /
                 ↪  13137111513365174591322
                  0.042994816804866599 ETH loss for the user
```

WangSecurity

@0xklapouchy I see what the problem is now, since the protocol assumes 1 stETH == 1 ETH, then it just gets the value of wstETH in stETH and counts it as a value in WETH. Hence, the deposit amounts are calculated incorrectly, since the current stETH price is 1.000580 ETH. Is it correct (I know it may sound silly to confirm it but to make sure I understand your argument)?

0xklapouchy

@WangSecurity yes, this is correct.

I just want to add why this is not the same issue as other with '1 stETH == 1 WETH', it explain the root problem.

The problem here is that current oracle system always force you to use to use cross-price fetch.

It would be perfectly fine if it will allow to use Chainlink price feed directly, then you will be able to configure wstETH -> stETH -> WETH as cross, plus rETH / ETH directly.

But as it is not allowing that, the base token needs to be the same. And as such it lead to a problem that 1 stETH == WETH if first underlying is wstETH.

WangSecurity

About the other 1 stETH == 1 WETH issue, is it about #299? @0xklapouchy

0xklapouchy

#29

WangSecurity

Thank you, but in that sense, I believe these are duplicates. The root cause of assuming 1 stETH == 1 WETH gives two vulnerability paths. The first is in case of an increased difference between the prices of 2 assets, leading to an incorrect share amount minted and a loss for a user. The second path is in this report where this assumption would break the protocol when the second token is added, cause the exchange rate would be incorrect, thus, the loss for a user.

Both can be fixed with the same fix, of using the chainlink stETH/ETH price feed, for example. In that sense, I believe these issues can be duplicated together because the root cause is the same, the fix is the same, and the impact and the vulnerability path are different, which is sufficient to consider them duplicates.

0xklapouchy

I agree it is similar to #29 I can even agree it is duplicate of it, but the issue is only valid when you add additional underlying to wstETH. Most of the duplicates of #29 don't even mention a valid path to this edge case issue.

Having 1 stETH == 1 WETH is perfectly fine if you only use 1 underlying token, it is even desired to not fetch stETH / ETH price then, and use constant 1e18 instead.

@WangSecurity if you plan to confirm that this issue is valid, and #29 is a duplicate or this issue a duplicate of #29, please recheck on all of #29 duplicates to determine if the explanations were sufficient.

WangSecurity

Thank you for pointing out, I'll check out the duplicates and comment on the ones I don't find sufficient.

0x3b33

Besides the above 5 issues I think 29 can also be considered a duplicate.

WangSecurity

@0x3b33 I think I've already set #29 as duplicate, or I'm missing something (#29 has `Duplicate of #266` in the bottom of the report)?

0x3b33

It still had excluded so I wanted to mention it just in case. Sorry if I opened more work.

WangSecurity

Oh, I see, the excluded label doesn't mean it's not duplicated or not rewarded. This label is set if the issue was invalidated initially during the judging contest (or initial phase of judging, when there's no judging contest as for Mellow).

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.