SHERLOCK

SHERLOCK SECURITY REVIEW FOR

SHERLOCK

Prepared for: Sentiment
Prepared by: Sherlock
Lead Security Expert: obront
Dates Audited: November 1 - November 4, 2022
Prepared on: November 15, 2022

Introduction

Sentiment is a permissionless undercollateralised onchain credit protocol that allows users to lend and borrow assets with increased capital efficiency and deploy them across DeFi.

Scope

Changes made after the Sentiment contest:

Github changes: +699 added, -3 deleted PRs since last Sentiment contest:

1. https://github.com/sentimentxyz/protocol/pull/239
2. https://github.com/sentimentxyz/controller/pull/47
3. https://github.com/sentimentxyz/controller/pull/46
4. https://github.com/sentimentxyz/oracle/pull/45

Files in scope:

- controller @ fd26c60
- oracle @ 101e699
- protocol @ 57ed340

All contracts in the folders (that represent the repos above) except contracts in any test folder.

Note: These repo folders will be have a "-merged" appended to them in your private repo.

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.

Total issues

Medium	High
4	1

Security experts who found valid issues

obront
pashov
Bahurum

Issue H-1: Non terminal tokens received via Balancer batchSwap are not accounted for

Source: https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/6

Found by

Bahurum, obront

Summary

Balancer's `batchSwap()` function is able to return multiple tokens along the execution path, but BalancerController.sol only accounts for receipt of the final token in the swap. The result is that other tokens received from a user's trade will not be registered with the protocol or included in their account.

Vulnerability Detail

Balancer's `batchSwap()` functionality takes in an array of `BatchSwapSteps`, each of which dictates a pool to use for the swap, assets to swap to and from, an amount, and some additional user data (which is currently unused by Balancer).

For users who would like all their funds to move from step to step (swapping all of their input token from their first step, for the maximum amount of the output token from the final step), Balancer provides the helpful option to set `amount=0` for all steps after the first, and it will automatically use the maximum amount.

Alternatively, users are able to provide their own `amounts` in each step. Any amount returned from a step that is not used for the next step is sent to the user.

Here is a simple example:

- Step 1: Trade 400 BAL for max WETH (returns ~1.7)
- Step 2: Trade 1 WETH for max USDC (returns ~1575)
- Result: 400 BAL is inputted, and both 0.7 WETH and 1575 USDC are returned

Here is a gist with a standalone Foundry test that can be run to show this behavior.

As the `canBatchSwap()` function is currently implemented, only the final asset (in this case, USDC) would be returned in the `tokensIn[]` array. While the user's WETH would be transferred to their account, it would not be registered with the protocol to count towards their account balance or interact with.

Impact

If a user uses Balancer's `batchSwap()` functionality with returned intermediate tokens that are not in the account's asset list already, the user may get liquidated sooner than expected as `RiskEngine.sol:_getBalance()` only counts in the assets in the assets list.

Code Snippet

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerController.sol#L193-L197

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerController.sol#L219-L228

Tool used

Manual Review, Foundry

Recommendation

Balancer's `batchSwap()` function returns `int256[]memoryassetDeltas`, which provide details on the final amounts of each asset that are inputted and outputted from the function.

Rather than try to manually calculate these values, the best option is likely the following:

- Use an in-line assembly block to `STATICCALL` Balancer's `canBatchSwap()` to return the asset deltas
- Any positive number in `assetDeltas` means the token belongs in `tokensOut`
- Any negative number in `assetDeltas` means the token belongs in `tokensIn`

Discussion

r0ohafza

PR: https://github.com/sentimentxyz/controller/pull/49

zobront

Confirmed. Didn't follow recommendation above, but blocked all trades with intermediate amounts > 0, so problem is solved.

Issue M-1: Curve LP Controller withdraw and claim function uses wrong signature

Source: https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/23

Found by

obront

Summary

The function signature used for `WITHDRAWCLAIM` in both CurveLPStakingController.sol and BalancerLPStakingController.sol are incorrect, leading to the function not succeeding.

Vulnerability Detail

In both the CurveLPStakingController.sol and BalancerLPStakingController.sol contracts, the function selector `0x00ebf5dd` is used for `WITHDRAWCLAIM`. This selector corresponds to a function signature of `withdraw(uint256,address,bool)`.

```
bytes4 constant WITHDRAWCLAIM = 0x00ebf5dd;
```

However, the `withdraw()` function in the Curve contract does not have an address argument. Instead, the function signature reads `withdraw(uint256,bool)`, which corresponds to a function selector of `0x38d07436`.

Impact

Users who have deposited assets into Curve pools will not be able to claim their rewards when they withdraw their tokens.

Code Snippet

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerLPStakingController.sol#L30-L31

Tool used

Manual Review

Recommendation

Change the function selector in both contracts to `0x38d07436`.

Discussion

ruvaag

The Curve docs don't reflect this but both 2CRV and crv3Crypto expose `withdraw(uint256,address,bool)` so this issue is invalid when looking at CRV LP tokens

That being said, Balancer exposes `withdraw(uint256,bool)` instead of `withdraw(uint256,address,bool)` so we will implement a fix to that effect.

ruvaag

Disagree with severity because users could use the `claim_rewards` function to claim pending rewards. But agree with the issue.

r0ohafza

Fix: https://github.com/sentimentxyz/controller/pull/48

zobront

Fix confirmed.

Issue M-2: Curve LP staking allows adding false tokensIn to account by setting _claim_rewards to false

Source: https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/20

Found by

obront

Summary

In CurveLPStakingController.sol and BalancerLPStakingController.sol, `canDepositAndClaim()` and `canWithdrawAndClaim()` assume that the user is claiming reward tokens, and add these tokens to the `tokensIn` array (to add to their account).

However, the only requirement to go down this code path is to use the full function signature (ie `deposit(uint256,address,bool)`), so false tokens will be added to a user account if they use the full signature with the `_claim_rewards` bool set to `false`.

Vulnerability Detail

Each Controller returns an array of `tokensIn` that will be added to the user's account, if they don't already hold a balance.

In CurveLPStakingController.sol and BalancerLPStakingController.sol, there are two overlapping pairs of function signatures:

- `deposit(uint256)` and `deposit(uint256,address,bool)` are the same function, but the middle parameter (`_addr`) defaults to `msg.sender` and the final parameter (`_claim_rewards`) defaults to `false`
- `withdraw(uint256)` and `withdraw(uint256,bool)` are the same function, but the final parameter (`_claim_rewards`) defaults to `false`

The assumption in the Controller logic is that, if a user uses the full signature, they are claiming their rewards. However, this isn't always the case. It is perfectly valid for a user to use the full function and pass `false` (the default argument) to the `_claim_rewards` parameter.

In this case, the function signature would lead to `canDepositAndClaim()` or `canWithdrawAndClaim()`, and all the rewards tokens would be added to the user's account.

Impact

Accounting on user accounts can be thrown off (intentionally or unintentionally), resulting in mismatches between their assets array and hasAsset mapping and the reality of their account.

Code Snippet

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerLPStakingController.sol#L24-L25

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerLPStakingController.sol#L52

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerLPStakingController.sol#L72-L92

Tool used

Manual Review

Recommendation

In `canCall()`, decode the calldata to get the value of the `_claim_rewards` bool. If this value is false, call the non-claim version of the function (ie `canDeposit()` instead of `canDepositAndClaim()`).

Discussion

ruvaag

While I agree with the issue, I'm not sure if it qualifies as a medium because I don't think this can lead to loss of funds or functionality.

zobront

Confirmed fix in https://github.com/sentimentxyz/controller/pull/48

Evert0x

I think a medium severity is valid as there is a risk of to let users add tokens to their account that they don't really have.

Issue M-3: User can trade away an asset using Balancer batchSwap without removing it from account

Source: https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/7

Found by

obront

Summary

Balancer's `batchSwap()` function takes in an `amount` of tokens to trade at each step of the swap. This amount can actually exceed the amount returned from the previous step. Because BalancerController.sol uses only the first token in the chain as `tokensOut`, a user can abuse this functionality to empty their account of a token while bypassing the expected `removeAsset` calls.

Vulnerability Detail

Balancer's `batchSwap()` functionality takes in an array of `BatchSwapSteps`, each of which dictates a pool to use for the swap, assets to swap to and from, an amount, and some additional user data (which is currently unused by Balancer).

The expectation is that users will either leave `amount=0` (which will autofill the amount returned from the last step) or will set `amount` manually to some smaller value.

However, it is also possible to input an `amount` that is greater than the value returned by the previous step, if you own those tokens and you have already approved the Balancer contract to trade them.

Using this technique, a user could send their tokens to Balancer without having them flagged by Sentiment as `tokensOut`. In an extreme case, this would allow them to get rid of all their tokens, without their account being updated to reflect this.

Here is an example:

- A user has 400 BAL and 100 WETH
- They call `batchSwap()` with the path BAL => WETH => USDC
- For the first step, they set `amount=400`
- For the second step, they set `amount=[amountreturnedfromstep1]+100`
- The result is that they will have traded all of their BAL and WETH into USDC, but only BAL will appear in `tokensOut`

Here is a gist with a proof of concept showing a user decreasing their WETH balance without WETH appearing in tokensOut.

Impact

Users have the ability to use the `batchSwap()` functionality to throw off the accounting on their account, removing a token but keeping the token in their `assets` array and keeping `hasAsset[token]=true`.

Code Snippet

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerController.sol#L193-L197

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/controller-merged/src/balancer/BalancerController.sol#L219-L222

Tool used

Manual Review, Foundry

Recommendation

Balancer's batchSwap() function returns int256[] memory assetDeltas, which provide details on the final amounts of each asset that are inputted and outputted from the function.

Rather than try to manually calculate these values, the best option is likely the following:

- Use an in-line assembly block to STATICCALL Balancer's canBatchSwap() to return the asset deltas
- Any positive number in assetDeltas means the token belongs in tokensOut
- Any negative number in assetDeltas means the token belongs in tokensIn

Discussion

r0ohafza

PR: https://github.com/sentimentxyz/controller/pull/49

zobront

Confirmed fix. Didn't follow recommendation above, but blocked all trades with intermediate amounts > 0, so problem is solved.

Issue M-4: No check for active Arbitrum Sequencer in WSTETH Oracle

Source: https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/3

Found by

pashov, obront

Summary

Chainlink recommends that all Optimistic L2 oracles consult the Sequencer Uptime Feed to ensure that the sequencer is live before trusting the data returned by the oracle. This check is implemented in ArbiChainlinkOracle.sol, but is skipped in WSTETHOracle.sol.

Vulnerability Detail

If the Arbitrum Sequencer goes down, oracle data will not be kept up to date, and thus could become stale. However, users are able to continue to interact with the protocol directly through the L1 optimistic rollup contract. You can review Chainlink docs on L2 Sequencer Uptime Feeds for more details on this.

As a result, users may be able to use the protocol while oracle feeds are stale. This could cause many problems, but as a simple example:

- A user has an account with 100 tokens, valued at 1 ETH each, and no borrows
- The Arbitrum sequencer goes down temporarily
- While it's down, the price of the token falls to 0.5 ETH each
- The current value of the user's account is 50 ETH, so they should be able to borrow a maximum of 200 ETH to keep account healthy (`(200+50)/200=1.2`)
- Because of the stale price, the protocol lets them borrow 400 ETH (`(400+100)/400=1.2`)

Impact

If the Arbitrum sequencer goes down, the protocol will allow users to continue to operate at the previous (stale) rates.

Code Snippet

https://github.com/sherlock-audit/2022-11-sentiment/blob/main/oracle-merged/src/wsteth/WSTETHOracle.sol#L45-L57

Tool used

Manual Review

Recommendation

Add the same check to WSTETHOracle.sol that exists in ArbiChainlinkOracle.sol:

```
function getPrice(address token) external view override returns (uint) {
    if (!isSequencerActive()) revert Errors.L2SequencerUnavailable();
    ...
}
```

```
function isSequencerActive() internal view returns (bool) {
    (, int256 answer, uint256 startedAt,,) = sequencer.latestRoundData();
    if (block.timestamp - startedAt <= GRACE_PERIOD_TIME || answer == 1)
        return false;
    return true;
}
```

Discussion

r0ohafza

Fix PR: https://github.com/sentimentxyz/oracle/pull/46

zobront

PR confirmed.