CvxCrvStakingWrapper Audit

This report was made by reviewing the https://github.com/convex-eth/platform repository on commit 8decdf.

The review started on *Thursday, January 5, 2023*.

This report was updated on *Friday, January 20, 2023*.

Findings

1. Faulty reward token can prevent withdrawals

IMPACT HIGH LIKELIHOOD LOW

Reward tokens that revert on `balanceOf` can prevent user withdrawals, as the call in line 261 would revert. Even though it is possible to invalidate a reward token it would require the `owner` to act so users can withdraw their tokens again.

Recommendations

A potential solution would be to allow users to "renounce" to specific token rewards forever, in which case those tokens would be skipped when performing a checkpoint for the user.

Update: Acknowledged, the Convex team plans to implement an owner layer in the future that will mitigate this by using timelocks for specific operations, requiring stricter conditions to add a token, or giving a DAO additional control over the system. The reasoning behind this is that implementing this solution in the wrapper itself would likely result in additional complexity that could cause other unexpected issues.

2. `rewards` array is unbounded

IMPACT MEDIUM LIKELIHOOD LOW

New reward tokens can be added by the `owner` or by synchronizing the extra rewards from `cvxCrvStaking`. For most operations, the `rewards` array is iterated and multiple storage reads are performed for each reward. This could eventually result in important operations such as `withdraw` becoming prohibitively expensive to execute.

This is unlikely to arise from a normal operation of the system. However, it must be noted that it could actually happen due to a bug in `cvxCrvStaking` or from a faulty behavior from the `owner`.

Recommendations

Consider limiting the amount of reward tokens that can be added, or supporting the removal of tokens from the `rewards` array.

Update: As of commit `e6ee38`, this issue has been resolved by limiting the array's length.

3. Rewards could get stuck due to extreme weights

IMPACT LOW LIKELIHOOD LOW

If all users set the same extreme weight (`0` or `10000`), all new received rewards for the opposite group will get stuck. This happens because the `reward_integral` for the tokens in the opposite group won't get updated when executing a checkpoint. This will continue to happen until a user sets its weight to receive rewards from the opposite group, but the previously received rewards will remain stuck in the contract.

PoC

https://github.com/nomoixyz/convex-cvxcrv-staking-wrapper/blob/e1ba5ebdfd6515dc98b3b209065adc2d10d035f7/test/CvxCrvStakingWrapper.t.sol#L111

Recommendations

Consider returning early in `_calcRewardIntegral` for rewards that belong to the group for which there is no weight allocated.

Update: Acknowledged, won't be implemented as it can only happen under extreme conditions and changes to critical parts of the code would be needed

4. Changing reward groups could lead to inconsistencies

IMPACT LOW LIKELIHOOD LOW

Changing the group of a reward token can result in inconsistent calculations and other unexpected issues.

One example of this is the calculation of `reward_integral`. Conceptually, `reward_integral` tracks the total amount of reward tokens per wrapper token (only considering wrapper tokens that correspond to the same group as the reward) that the contract has received. Thus, when the reward is moved to the other group, the `reward_integral` becomes detached from the amount it was tracking originally, as the total amount of wrapper tokens that correspond to the new group will be different. This can result in some users getting more rewards than they should and some users losing some of their rewards, depending on their reward weights (the extreme case being when a user's reward weight is `0` or `WEIGHT_PRECISION`).

Recommendations

Consider removing the functionality to change reward groups. Another solution would be to track a `reward_integral` per reward per group, but the additional complexity of such an implementation might not be worth it, as group changes shouldn't happen frequently anyways.

Update: As of commit `e6ee38`, this issue has been partially resolved by removing the permissionless syncing of reward tokens through `addRewards`, and acknowledging that it is not desirable to call `setRewardGroup` unless an error is made when adding a reward.

5. `supplyWeight` can be slightly manipulated

IMPACT LOW LIKELIHOOD LOW

It is possible for anyone to increase or decrease the `supplyWeight` by `1` without changing the individual user weights. This can happen due to rounding errors in the `_beforeTransfer` hook. For example, a user with a `balance` of `10000` (equal to `WEIGHT_PRECISION`) and a `weight` of `1`, can perform transfers to *himself* of a single token, resulting in `supplyWeight` decrementing by `1`.

This could potentially be repeated thousands of times to make other parts of the code break, such as underflowing the `supplyWeight` itself, underflowing the subtraction in line 268, or affecting the `reward_integral` calculation in line 275.

We believe this would only be an issue in extreme cases, such as `supplyWeight` being too close to `0` or to `totalSupply`. We weren't able to find ways to increase or decrease `supplyWeight` by more than `1`.

Recommendations

Revert in `_beforeTokenTransfer` if both accounts are non-zero and `supplyWeight` changed, or if one account is non-zero and `supplyWeight` did not change. This solution could result in unexpected failed transactions so these edgecases must be handled by the frontend.

Update: As of commit `e6ee38`, this issue has been resolved by not adjusting `supplyWeight` for self transfers.

6. `_rewardGroup` is ignored when reactivating a token reward

ENHANCEMENT

When calling `addTokenReward` to reactivate a previously invalidated token, the `_rewardGroup` argument is completely ignored. This could be considered unexpected behavior from the perspective of the `owner`, as they might believe that the reactivated token would get assigned to the provided group.

Update: Acknowledged, won't be implemented.

7. Prevent "special" tokens from being added as rewards

ENHANCEMENT

Even though we weren't able to find concrete attacks that exploit this, as an additional safety measure consider preventing the addition of `CvxCrvStakingWrapper`, `cvxCrvStaking`, and `cvxCrv` as reward tokens.

Update: As of commit `e6ee38`, this issue has been resolved by preventing the addition of safe tokens as rewards.

8. Use `nonReentrant` for `external`/`public` functions instead of `internal` ones

ENHANCEMENT

Using `nonReentrant` only for the internal checkpoints could lead to unexpected reentrancy attacks.

For example, the `withdraw` function calls `_burn`, which triggers a `_checkpoint` (`nonReentrant`). However, the external calls that come after `_burn` could actually reenter `withdraw` or any of the other contract functions. This could lead to inconsistent states where the internal balances and total supply don't match the actual deposited tokens.

Update: Acknowledged, won't be implemented.

9. Use `nonReentrant` where possible

ENHANCEMENT

To prevent unexpected reentrancy issues, consider using `nonReentrant` for all functions that modify the state or perform external calls, such as `shutdown`, `reclaim` and `addTokenReward`.

For example, if there was a way to reenter the contract in the middle of a `withdraw` (after the `totalSupply` has been decreased), the `owner` could call `shutdown` and `reclaim` and make the `extraTokens` variable big enough to steal all deposits. Note: we were not able to identify any scenarios that would enable a malicious reentrancy when looking at the `cvxCrvStaking` and `cvxCrv` contracts.

Update: As of commit `e6ee38`, this issue has been resolved by using `nonReentrant` for `shutdown`, `reclaim` and `addTokenReward`.

10. Unused variables

ENHANCEMENT

- `threeCrvRewards` is unused.
- `isInit` is unused.

Update: As of commit `e6ee38`, this issue has been resolved removing unused variables and imports.

11. Missing events

ENHANCEMENT

Some functions don't emit events, which could lead to inconsistencies and hinder the monitoring of contract interactions.

- `RewardGroupSet` event not emitted in the `addRewards` function.

- Event missing in the `setHook` function.
- Events missing in both `getReward` functions.
- Event missing in the `shutdown` function.

Update: As of commit `e6ee38`, this issue has been resolved by emitting events in `addRewards`, `setHook`, `shutdown` and `getReward`.