

Convex F(x) Review

BROKEN WITH ❤️ BY  NOMOI

We reviewed the https://github.com/convex-eth/function_x/tree/feature/vaults repository at commit [b85fb4e](#).

The review started on *Monday, December 4, 2023*.

This report was updated on *Friday, December 15, 2023*.

Introduction

Our team has performed a security review of the Convex-Function(x) Staking Platform, specifically the vault implementation and related components, with the objective of independently assessing the security aspects, code quality, and overall functionality of the project's smart contracts.

As always, the Convex team takes an extreme approach to build their systems to be as trustless as possible, having no access to user funds. The codebase was clean and easy to understand, and we were not able to find major issues.

The smart contracts in scope for this audit were:

```
.
├── boosting
│   ├── FeeRegistry.sol
│   ├── MockGauge.sol
│   ├── MultiRewards.sol
│   ├── PoolRegistry.sol
│   ├── PoolUtilities.sol
│   ├── StakingProxyBase.sol
│   └── StakingProxyERC20.sol
```

Findings

1. MultiRewards can't be initialized

LIKELIHOOD MEDIUM IMPACT LOW

The `initialize` function of the `MultiRewards` contract will always revert due to [incorrectly checking the `rewardState`](#) to be different from `RewardState.NotInitialized`.

Recommendation

Fix the `rewardState` check by requiring that it is equal to `RewardState.NotInitialized`.

Update: As of commit [5ddf119](#), this issue has been resolved by requiring that the `rewardState` is equal to `RewardState.NotInitialized`.

2. FeeReceiverCvxFxn doesn't support `rewardToken == 0`

LIKELIHOOD MEDIUM IMPACT LOW

The `processFees` function of the `FeeReceiverCvxFxn` contract [incorrectly checks](#) that the `rewardAddress` is not the zero address before processing the `rewardToken` fees. This means that if the `rewardToken` is zero, all calls to `processFees` will fail.

The impact of this issue is limited, as it can easily be solved by setting a fake reward token that always returns `0` for `balanceOf`.

Note: This contract was out of scope and we discovered it while studying the rest of the codebase. The Convex team was already aware of this issue. Fixed in commit [aa9e005](#).

3. Bad reward token can brick `MultiRewards.getReward`

LIKELIHOOD **LOW** IMPACT **MEDIUM**

The `MultiRewards.getReward` function iterates over all the reward tokens. For each token, it `executes safeTransfer` to distribute rewards. However, this process is vulnerable to a failure scenario: if any of these reward tokens becomes non-functional (for instance, due to an improper proxy update), causing the `safeTransfer` operation to consistently fail, it will consequently result in the failure of the entire `getReward` function. This dependency on each reward token's functionality poses a significant risk, as a single malfunctioning token can disrupt the reward distribution mechanism in an unrecoverable way.

Recommendation

- Consider adding a `bool` field to the `Reward` struct that determines if a reward is disabled or not. In the `getReward` function skip the disabled rewards.
- Alternatively, modify the `getReward` function to accept an array of reward tokens. This would require changes in functions such as `StakingProxyBase._processExtraRewards` and `StakingProxyERC20.getReward`. Due to the additional complexity introduced by this option, we prefer the first recommendation.

Update: As of commit [fe146ee](#), this issue has been resolved by supporting a token list when claiming rewards.

4. Optimizations

OPTIMIZATION

Disclaimer: Please be advised that the code optimizations suggested in our audit report are intended to enhance the gas efficiency of your project's smart contracts. However, it is crucial to understand that implementing these changes can potentially alter the original execution logic of your code.

- In the `PoolRegistry` contract there are multiple instances where the same storage variable is read within the same scope, for example:
 - In `addPool` there are multiple reads of the `rewardImplementation` variable.
- In the `MultiRewards` contract there are multiple instances where the same storage variable `id` is read within the same scope, for example:

- In `deposit` and `getReward` there are multiple reads of the `rewardHook` variable.
- In the `StakingProxyBase` contract there are multiple instances where the same storage variable is read within the same scope, for example:
 - In `initialize` there are multiple reads of the `owner` variable.
 - In `_checkpointRewards` and `_processExtraRewards` there are multiple reads of the `rewards` variable.
- In the `StakingProxyERC20` contract there are multiple instances where the same storage variable is read within the same scope, for example:
 - In `deposit` and `withdraw` there are multiple reads of the `stakingToken` variable.
 - In `earned` there are multiple reads of the `rewards` variable.

Update: Several minor optimizations were performed on commit [ed3a5f8](#) .
