Convergence

Security review

Version 1.0

Reviewed by

nmirchev8
deth

Table of Contents

1 About Egis Security

Egis Security is a team of experienced smart contract researchers, who strive to provide the best smart contract security services possible to DeFi protocols.

The team has a proven track record on public auditing platforms like Code4rena, Sherlock, and Cantina, earning top placements and rewards exceeding $170,000. They have identified over 150 high and medium-severity vulnerabilities in both public contests and private audits.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Convergence
Repository	https://github.com/Convergence-fi/convergence-convex-reaudit
Commit hash	fd825a5d57fba05fdc3a54bd3a40ff58cf5b56a0
Resolution	-
Documentation	-
Methods	Manual review

Scope

contracts/Rewards/CvgRewardsV3.sol
contracts/Staking/Convex/cvgCVX/CvgCVX.sol
contracts/Staking/Convex/cvgCVX/CvgCvxStakingPositionService.sol
contracts/Staking/Convex/cvgCVX/CVX1.sol
contracts/Staking/Convex/CvxRewardDistributorV2.sol
contracts/Staking/Convex/StakingServiceBase.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	2
Low risk	5
Informational	7

5 Findings

5.1 Medium risk

5.1.1 Malicious user can lock CVX1 cvxcrv rewards in cvxCrvRewards contract

Severity: *Medium risk*

Context: CVX1.sol#L168-L172

Description: When user deposit CVX to CVX1 contract, CVX tokens are staked to cvxRewardPool. Corresponding rewards can be withdrawn with CVX1#getReward function, which will transfer accured CVX_CRV to the cvgControlTower.convexTreasury():

```
    function getReward() external {
        cvxRewardPool.getReward(false);

        /// @dev transfer cvxCRV tokens
        CVX_CRV.transfer(cvgControlTower.convexTreasury(), CVX_CRV.balanceOf(address
            (this)));
    }
```

The problem is that anyone can call cvxRewardPool.getReward for CVX1 and set _stake = true, which will restake CVX_CRV balance into cvxCrvRewards contract:

```
    function getReward(address _account, bool _claimExtras, bool _stake) public
        updateReward(_account){
        uint256 reward = earnedReward(_account);
        if (reward > 0) {
            rewards[_account] = 0;
            rewardToken.safeApprove(crvDeposits,0);
            rewardToken.safeApprove(crvDeposits,reward);
            ICrvDeposit(crvDeposits).deposit(reward,false);

            uint256 cvxCrvBalance = cvxCrvToken.balanceOf(address(this));
            if(_stake){
                IERC20(cvxCrvToken).safeApprove(cvxCrvRewards,0);
                IERC20(cvxCrvToken).safeApprove(cvxCrvRewards,cvxCrvBalance);
                IRewards(cvxCrvRewards).stakeFor(_account,cvxCrvBalance);
            }
```

Those tokens are locked, because CVX1 don't have function to withdraw from cvxCrvRewards contract.

Recommendation: Consider implementing the same withdrawCvxCrv, which is in CvgCVX contract:

```
    function withdrawCvxCrv(uint256 amount, bool claim) external onlyOwner {
        if (amount != 0) {
            CVX_CRV_REWARDS.withdraw(amount, claim);
        } else {
            CVX_CRV_REWARDS.withdrawAll(claim);
        }
    }
```

Resolution: Acknowledged

5.1.2 It may be impossible to mint all CVXRush indexes

Severity: *Medium risk*

Context: CvgCVX.sol#L316

Description: When we set `capCVXRushForIndex`, we make the following check to ensure that caps in the array are in increasing order:

```
for (uint256 i; i < cvxRushIndexInfos.length; ) {
         capCVXRushForIndex.push(cvxRushIndexInfos[i]);
         if (i != 0) {
             require(cvxRushIndexInfos[i].cap > cvxRushIndexInfos[i - 1].cap, "
                 WRONG_CAP");
             require(cvxRushIndexInfos[i].ratio < cvxRushIndexInfos[i - 1].ratio,
                  "WRONG_RATIO");
         }
         unchecked {
             ++i;
         }
     }
```

But there is a problem in `_calculateCvgIncentive` function calculation, which may result in inability to use indexes in the end of the array.

Imagine the following scenario:

- We have `capCVXRushForIndex = [{cap = 100}, {cap = 200}, {cap = 300}, {cap = 400}]`
- If we have used first 3 indexes, we will have `cvgCvxAlreadyMintedWithRush = 100 + 200 + 300 = 600`
- Now `cvxRush` designed for the last cycle cannot be used, because in `_calculateCvgIncentive` we fetch `cvgCvxAlreadyMintedWithRush` (600) and compare it against each cap in the array:

```
for (uint256 i; i < _numberOfCvxRushIndex; ) {
          /// @dev Retrieve the cap in cvgCVX
          uint256 currentCap = capCVXRushForIndex[i].cap;

          if (_cvgCvxAlreadyMintedWithRush < currentCap) { // @sus very sus
              uint256 restToMintForIndex = currentCap -
                  _cvgCvxAlreadyMintedWithRush;
              if (restToMintForIndex >= cvgCvxAmountUsedForBonus) {
                  cvgIncentive += uint128(cvgCvxAmountUsedForBonus *
                      capCVXRushForIndex[i].ratio) / 1000;
                  _cvgCvxAlreadyMintedWithRush += cvgCvxAmountUsedForBonus;
                  delete cvgCvxAmountUsedForBonus;
                  break;
              } else {
                  cvgCvxAmountUsedForBonus -= restToMintForIndex;
                  cvgIncentive += uint128(restToMintForIndex * capCVXRushForIndex[
                      i].ratio) / 1000;
                  _cvgCvxAlreadyMintedWithRush += restToMintForIndex;
              }
          }
          unchecked {
                ++i;
            }
        }
```

- The following results in inability to allocate the amount for the last index, because we we have set `cvgCvxAlreadyMintedWithRush` to value greater than the cap for the last index, because it is set to the sum of all previous indexes caps
- Even though system should allow 400 more tokens to be `mintCVXRush`ed, it won't be possible. The transaction will always revert, because `_calculateCvgIncentive` will return `cvgCvxAmountUsedForBonus > 0`

Recommendation: Add additional `filled` field to `CvxRushIndexInfo` and use it to calculate cvg incentive for the correct index.

Resolution: Acknowledged

5.2 Low risk

5.2.1 `getClaimableCyclesAndAmounts` will revert when there is `rewardAsset` with 0 amount

Severity: *Low risk*

Context: StakingServiceBase.sol#L841-L849

Description:

`StakingServiceBase#getClaimableCyclesAndAmounts` is a view function, which returns claimable rewards for an account on each cycle:

```
    struct ClaimableCyclesAndAmounts {
        uint256 cycleClaimable;
        uint256 cvgRewards;
        ICommonStruct.TokenAmount[] cvxRewards;
    }
```

But there is a problem when we try to reduce `_cvxRewardsClaimable` array length when `rewardAsset.amount == 0`:

```
if (cycleInfo[nextClaimableCvx].isCvxProcessed) {
                        _cvxRewardsClaimable = new ICommonStruct.TokenAmount[](
                            maxLengthRewards);
                        for (uint256 x; x < maxLengthRewards; ) {
                            ICommonStruct.TokenAmount memory rewardAsset =
                                cvxRewardsByCycle[nextClaimableCvx][x + 1];
                            if (rewardAsset.amount != 0) {
                                _cvxRewardsClaimable[x] = ICommonStruct.TokenAmount
                                    ({
                                    token: rewardAsset.token,
                                    amount: (amountStaked * rewardAsset.amount) /
                                        totalStaked
                                });
                            } else {
                                // solhint-disable-next-line no-inline-assembly
                                assembly {
                                    /// @dev this reduce the length of the array to
                                        not return some useless 0 at the end
                                    // @sus this is wrong! We will have the useless
                                        0 amount at the middle of the array and most
                                         probably revert when we try to assign the
                                        last reward
                                    mstore(_cvxRewardsClaimable, sub(mload(
                                        _cvxRewardsClaimable), 1))
                                }
                            }
                            unchecked {
                                ++x;
                            }
                        }
                    }
```

In the assembly block we reduce the array size by 1, but we always increment `++x`. If `maxLengthRewards == 5`. For one of the cycles we don't have rewards for token with `id = 2`, we will change `_cvxRewardsClaimable` length to be 4. However, on on the last **`for`** iteration we will try to assign value in the array as if we have the original array size `_cvxRewardsClaimable[4] (5th element)`. The following results in EVM panic revert, because we try to assign value to index above the new length of the array. The function is **`view`** and we didn't find existing integration of it in the current scope, so we decided to mark it as `Low`.

Recommendation: If you want to use the same approach, consider implementing another variable, which will be used to count the index. It should be incremented only inside the if branch.

Resolution: Acknowledged

5.2.2 Malicious actor can call `processCvxRewards` without calling `getReward` previously

Severity: *Low risk*

Context: CvgCVX.sol & StakingServiceBase.sol

Description: Stakers accrue their rewards from `CVX_LOCKER`, which is holding CVX token of all depositors. `CvgCVX` contract is using it's balance of reward tokens to calculate the corresponding rewards for the given cycle. The problem is that someone should manually call `CVX_LOCKER#getReward` so the rewards are send to `CvgCVX` contract. If someone updates a cycle and call `processStakersRewards` => `processStakersRewards` without getting the rewards from CVX locker, he enforces them to stake for another cycle, or to not recieve any rewards for this one.

Recommendation: Consider getting accured rewards in `cvgCVX#pullRewards` before calculating `rewardsAvailable`:

```
function pullRewards(address processor) external returns (ICommonStruct.TokenAmount
    [] memory) {
    require(msg.sender == cvxStakingPositionService, "NOT_CVG_CVX_STAKING");
    CVX_LOCKER.getReward(address(this));
    ...
```

Resolution: Acknowledged

5.2.3 If `CvgRewards#writeStakingRewards` is not called for two cycles, user may steal other stakers rewards

Severity: *Low risk*

Context: CvgRewardsV3.sol#L149-L154

Description:
If `CvgRewardsV3#writeStakingRewards` is not called for two cycles, user may exploit it by using flashloan to stake, update cycles and rewards, claim, unstake and repay the loan in a single transaction.

Imagine the following scenario:

1. Bob and Alice has both staked $1000 `cvgCVX` at cycle 1
2. 2 weeks has passed and nobody has called `CvgRewardsV3#writeStakingRewards`
3. Eve sees the oppertunity and get a flashloan of $10M, and in a single tx manages to:

- stake the flashloan as `cvgCVX`
- calls `CvgRewardsV3#writeStakingRewards` => `processCvxRewards` => `CvgRewardsV3#writeStakingRewards` two times, so staking cycle is incremented from 1 => 3 and we have $X of cvx rewards distributed for cycle 2
- unstake all `cvgCVX` and repays the loan

4. Eve has inflated Bob and Alice rewards and she will receive > 99% of the amount accrued for those 2 weeks, because her weight for the distribution is ~ `10_000_000 / 2_000`

Resolution: Acknowledged

5.2.4 `CvgCVX::setCvxStakingPositionService()` - Doesn't reset `isSpecialMinter` for old `cvxStakingPositionService`

Severity: *Low risk*

Context: CvgCVX.sol#L556-L559

Description: The function sets `cvxStakingPositionService` to the new `_cvxStakingPositionService` and sets `_cvxStakingPositionService` as a special minter

```
function setCvxStakingPositionService(address _cvxStakingPositionService) external
    onlyOwner {
        cvxStakingPositionService = _cvxStakingPositionService;
        isSpecialMinter[_cvxStakingPositionService] = true;
    }
```

The issue is that, it doesn't reset `isSpecialMinter` for the old `cvxStakingPositionService`.

In case the old `cvxStakingPositionService` is compromised, he still will be considered a special minter until `toggleIsSpecialMinter` is called.

Marking this as Low, as if `cvxStakingPositionService` is compromised, he can mint `cvgCvx` up to `capCvgCvx` inflating the supply of the tokens and reaching the cap.

Although the gap in time where the compromised address has the special minter role and `toggleIsSpecialMinter` is called is small, it's still possible that the above can happen.

The same is present in `CVX1#setCvgCvx`:

```
    function setCvgCvx(ICvgCVX _cvgCVX) external onlyOwner {
        isSpecialMinter[address(_cvgCVX)] = true;
        cvgCVX = _cvgCVX;
    }
```

Recommendation: Consider revoking minter role of the old addresses when setting the new ones.

Resolution: Acknowledged

5.2.5 Under specific circumstances `withdrawableFees[token]` may become larger than `balanceOf[token]`

Severity: *Low risk*

Context: CvgCVX.sol#L450

Description: It's possible for `withdrawableFees[token] > balanceOf[token]` under the following circumstances:

- `withdrawableFees[token] = 100` and `balanceOf[token] = 100`.
- Owner removes `token` as a reward token.
- Owner calls `recoverTokens` for `token` and retrieves the entire balance.
- Now we have, `withdrawableFees[token] = 100` and `balanceOf[token] = 0`.
- If `token` is added back as a reward token, there is a chance that the incoming rewards won't be enough to cover the `withdrawableFees`, making the tx revert here:

```
function pullRewards(address processor) external returns (ICommonStruct.TokenAmount
    [] memory) {
...
uint256 withdrawableFeeAmount = withdrawableFees[rewardConfiguration.token];
            uint256 rewardsAvailable = rewardConfiguration.token.balanceOf(address(
                this)) - withdrawableFeeAmount;
...
```

Resolution: Acknowledged

5.3 Informational

5.3.1 `StakingServiceBase::claimCvgCvxMultiple()` - Function is useless, as it's uncallable

Severity: *Info risk*

Context: StakingServiceBase.sol#L271

Description:
Function is useless, as it's only callable by the `CvxRewardDistributor`, but `CvxRewardDistributorV2` doesn't interact with this function, making it redundant

Resolution: Acknowledged

5.3.2 Consider emitting events on important state changes

Severity: *Info risk*

Context: `CvgCVX.sol`, `StakingServiceBase.sol`

Description:
Consider emitting events on important state changes such as:

- In `CvgCVX`
  - `CvgCVX#setCvxDelegateRegistry`
  - `setCvxStakingPositionService`
  - `setMintFees`
  - `setRewardTokensConfigs`

    - `toggleIsSpecialMinter`

- In `StakingServiceBase`
    - `toggleDepositPaused`
    - `setBuffer`

Resolution: Acknowledged

5.3.3 If the same gauge is added twice in `CvgRewardsV3`, it may lead to incrementing it's cycle twice per week

Severity: *Info risk*

Context: CvgRewardsV3.sol#L112-L115

Description:
Consider checking if a gauge with such address exists and revert if it does.

Resolution: Acknowledged

5.3.4 Usage of `block.timestamp` in deadline calculations for Uniswap isn't recommended.

Severity: *Info risk*

Context: StakingServiceBase.sol#L668

Description:
Inside `_convertEthToAsset` we use `block.timestamp + 1000` in both UniV2 and UniV3 swaps, using `block.timestamp` as a deadline isn't recommended, as whenever the tx is executed that's what `block.timestamp` will be, thus there is technically no deadline.

Note: Leaving this as info, as slippage protection is enough, but using `block.timestamp` as a deadline is bad practice.

```
if (_poolEthInfo.poolType == IStakingServiceBase.PoolType.UNIV2) {
            address[] memory path = new address[](2);
            path[0] = WETH;
            path[1] = _poolEthInfo.token;
            uint256[] memory amounts = UNISWAPV2_ROUTER.swapExactETHForTokens{value:
                 amountIn}(
                amountOutMin,
                path,
                address(this),
                //@issue I - hardcoded deadline doesn't do anything
                block.timestamp + 1000
            );
            amountOut = amounts[1];
        } else if (_poolEthInfo.poolType == IStakingServiceBase.PoolType.UNIV3) {
            amountOut = UNISWAPV3_ROUTER.exactInputSingle{value: amountIn}(
                IUniv3Router.ExactInputSingleParams({
                    tokenIn: WETH,
                    tokenOut: _poolEthInfo.token,
                    fee: uint24(_poolEthInfo.fee),
                    recipient: address(this),
                    //@issue I - hardcoded deadline doesn't do anything
                    deadline: block.timestamp + 1000,
                    amountIn: amountIn,
                    amountOutMinimum: amountOutMin,
                    sqrtPriceLimitX96: 0
                })
            );
        }
```

Resolution: Acknowledged

5.3.5 In CvxRewardDistributorV2#_withdrawRewards if we mint cvgCVX ratio is different from 1:1

Severity: *Info risk*

Context: CvgRewardsV3.sol#L112-L115

Description:
In `CvxRewardDistributorV2#_withdrawRewards` if we mint `cvgCVX` ratio is different from 1:1, because `cvgCVX` is minted with `isLock` param set to **`false`**, which will result in fee collection from the CVX amount provided.

Resolution: Acknowledged

5.3.6 CvgRewardsV3::setMaxChunkConfigs is redundant and not used anywhere.

Severity: *Info risk*

Context: CvgRewardsV3.sol#L104-L106

Description:
Consider removing dead code.

Resolution: Acknowledged

5.3.7 Setting new CvxRewardPool in CVX1 may lead to small miscalculation

Severity: *Info risk*

Context: CvgRewardsV3.sol#L104-L106

Description:
When `CVX1::setCvxRewardPool` is called, there can be tokens that are still staked in the `cvxRewardPool`, which can lead to a small DoS.

- `cvxRewardPool1` has 100 CVX staked.
- `setCvxRewardPool` is called and now we are using `cvxRewardPool2`.
- `cvxRewardPool2` has 0 CVX staked.
- If a user attempts to `withdraw` and there aren't enough tokens in `CVX1` for him, we'll attempt to withdraw from `cvxRewardPool2`, but it has 0 staked tokens in it, so the tx will revert, blocking the user's withdraw.

Recommendation:
Call `cvxRewardPool.withdrawAll(true)` so that the entire stake and all the rewards are first retrieved, before changing the reward pools.

Resolution: Acknowledged