Security Audit Report

Concentrator aCVX Update by AladdinDAO

SECBIT

October 18, 2023

1. Introduction

The AladdinDAO is a decentralized network to shift crypto investments from venture capitalists to the wisdom of crowds through collective value discovery. The Concentrator is a yield enhancement product by AladdinDAO built for farmers who wish to use their Convex LP assets to farm top-tier DeFi tokens (CRV, CVX) at the highest APY possible. The current Concentrator protocol has introduced new strategies, allowing users to deposit CVX tokens into the Convex protocol and earn yields. SECBIT Labs conducted an audit from September 28 to October 18, 2023, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator aCVX contract has no critical security risks. The SECBIT team has some tips on logical implementation, potential risks, and code revising (see part 4 for details).

Type	Description	Level	Status
Design & Implementation	4.3.1 The `CvxStakingStrategy` contract might permanently lose earnings from the Convex protocol.	Medium	Fixed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about this Concentrator aCVX update is shown below:

- Smart contract code
    - initial review commit *0c6c8e0*
    - final review commit *9301e43*

2.2 Contract List

The following content shows the contracts included in the Concentrator aCVX update, which the SECBIT team audits:

Name	Lines	Description
CvxCompounder.sol	31	Together with the ConcentratorCompounderBase contract, it forms a complete functionality, providing users with CVX token deposit and withdrawal services.
CvxStakingStrategy.sol	74	Together with the AutoCompoundingStrategyBaseV2 contract, it forms a complete contract where user deposits are funneled into the Convex protocol. Additionally, this contract is responsible for managing the protocol's earnings.
AutoCompoundingStrategyBaseV2.sol	45	Together with the CvxStakingStrategy contract, it forms a complete contract where user deposits are funneled into the Convex protocol. Additionally, this contract is responsible for managing the protocol's earnings.
LinearRewardDistributor.sol	53	This module handles unexpected earnings within the protocol.
ConcentratorCompounderStash.sol	89	This contract handles unexpected user earnings, converting these earnings into CVX tokens.

3. Contract Analysis

This part describes code assessment details, including "role classification" and "functional analysis".

3.1 Role Classification

Two key roles in the Concentrator aCVX Protocol are Governance Account and Common Account.

- Governance Account
  - Description

    Contract Administrator
  - Authority
    - Update basic parameters
    - Transfer ownership
    - Migrate pool assets to a new strategy
  - Method of Authorization

    The contract administrator is the contract's creator or authorized by transferring the governance account.
- Common Account
  - Description

    Users participate in the Concentrator aCVX Protocol.
  - Authority
    - Deposit CVX tokens to receive a share token
    - Harvest pending rewards and reinvest in the pool
  - Method of Authorization

    No authorization required

3.2 Functional Analysis

Users who deposit CVX tokens into the Convex protocol via the new strategy in Concentrator can simultaneously receive earnings from both the Concentrator and Convex protocols. The SECBIT team conducted a detailed audit of some of the contracts in the protocol. We can divide the critical functions of the contract into two parts:

CvxCompounder and ConcentratorCompounderBase

These two modules provide interfaces for interacting with users. Users can deposit or redeem CVX tokens through these contracts. The main functions in these contracts are as follows:

- `deposit()`

  Users deposit CVX tokens into the contract through this function. These assets will be transferred to the Convex protocol. At the same time, users will receive corresponding shares.

- `mint()`

  Users can specify the shares they want to receive, and it will calculate the corresponding amount of CVX tokens they should deposit.

- `withdraw()`

  Users specify the amount of CVX tokens they want to withdraw, which will burn their corresponding shares.

- `redeem()`

  Users specify the shares they want to burn, which will calculate the corresponding amount of CVX tokens to withdraw.

- `harvest()`

  The harvester withdraws profits from the Convex protocol and distributes these earnings.

- `depositReward()`

  This function manages unexpected earnings from users, converting them into CVX tokens and distributing them through this function.

CvxStakingStrategy and AutoCompoundingStrategyBaseV2

The strategy contract that connects the Concentrator and the Convex protocol will facilitate the transfer of users' funds from this contract to the Convex protocol. Simultaneously, the profits earned by users are withdrawn from this contract and reinvested. The main functions in these contracts are as follows:

- `deposit()`

This function deposits the funds from the strategy into the Convex protocol.

- `withdraw()`

  This function withdraws CVX tokens from Convex and sends them to the specified address.

- `harvest()`

  This function assists users in claiming their earnings from Convex, converting them back into CVX tokens, and then staking them into the Convex protocol.

4. Audit Detail

This part describes the process, and the detailed audit results also demonstrate the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Lab. We analyzed the project from code bugs, logical implementation, and potential risks. The process consists of four steps:

- Fully analysis of contract code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the contract code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools, including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line, and the result could be categorized into the following types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g., overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20 standard	✓
7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓
9	Explicit implementation, visibility, variable type, and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓
13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in the design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓

17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No non-essential minting method	✓
21	Correct managing hierarchy	✓

4.3 Issues

4.3.1 The `CvxStakingStrategy` contract might permanently lose earnings from the Convex protocol.

Risk Type	Risk Level	Impact	Status
Design & Implementation	Medium	Design logic	Fixed

Description

The contract retrieves earnings from the Convex protocol using the `staker.getReward()` function. It's worth noting that the `staker` contract has a public `getReward()` function that anyone can call to claim rewards on behalf of the `CvxStakingStrategy` contract. Unexpectedly, the claimed rewards can be directly deposited into the `cvxCrvRewards` contract instead of transferred to the `CvxStakingStrategy` contract. Since the `CvxStakingStrategy` contract does not provide an interface to withdraw the principal and earnings from the `cvxCrvRewards` contract, this directly results in the loss of these earnings.

```
function harvest(address _zapper, address _intermediate)
external override onlyOperator returns (uint256 _harvested) {
    // 1. claim rewards from staking staker contract.
    address[] memory _rewards = rewards;
    uint256[] memory _amounts = new uint256[](rewards.length);
    for (uint256 i = 0; i < rewards.length; i++) {
      _amounts[i] =
IERC20(_rewards[i]).balanceOf(address(this));
    }
    ICvxRewardPool(staker).getReward(false);
    for (uint256 i = 0; i < rewards.length; i++) {
      _amounts[i] =
IERC20(_rewards[i]).balanceOf(address(this)) - _amounts[i];
    }

    // 2. zap all rewards to staking token.
    _harvested = _harvest(_zapper, _intermediate, CVX,
_rewards, _amounts);

    // 3. deposit into convex
    if (_harvested > 0) {
      ICvxRewardPool(staker).stake(_harvested);
    }
  }


// @audit link:
https://etherscan.io/address/0xCF50b810E57Ac33B91dCF525C6ddd988
1B139332
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));

            //@audit deposit earnings into the cvxCrvRewards
contract.
            if(_stake){
```

```
 IERC20(cvxCrvToken).safeApprove(cvxCrvRewards,0);

 IERC20(cvxCrvToken).safeApprove(cvxCrvRewards,cvxCrvBalance);

 IRewards(cvxCrvRewards).stakeFor(_account,cvxCrvBalance);
            }else{
                cvxCrvToken.safeTransfer(_account,
cvxCrvBalance);
            }
            emit RewardPaid(_account, cvxCrvBalance);
        }

      ......
}
```

Status

The development team fixed this issue in commit 9301e43. In the updated `CvxStakingStrategy` contract, they have introduced an interface to withdraw assets from the `cvxCrvRewards` contract. These assets will then be sent to a separate stash contract for further processing.

5. Conclusion

After auditing and analyzing the Concentrator aCVX Protocol contract, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above.

Disclaimer

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock assets inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality of the smart contract could bring risks.

SECBIT Lab is devoted to constructing a common-consensus, reliable, and ordered blockchain economic entity.

https://secbit.io

audit@secbit.io

@secbit_io