Security Audit Report

Concentrator IFO by AladdinDAO

SECBIT

July 1, 2022

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 new version of the Concentrator adds an Initial Farming Offering (IFO) feature. During the IFO phase, the earnings of the Concentrator protocol will be converted into CTR tokens and distributed to users. SECBIT Labs conducted an audit from May 26 to July 1, 2022, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator 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 Discussion of the `harvest()` caller reward design.	Info	Discussed
Design & Implementation	4.3.2 Add restrictions on start time and end time to improve the security of parameter settings.	Info	Fixed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the Concentrator IFO contract is shown below:

- Smart contract code
  - initial review commit *c417619*
  - final review commit *acc5007*

2.2 Contract List

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

Name	Lines	Description
ConcentratorIFOVault.sol	143	CTR token rewards distribution contract.
AladdinConvexVault.sol	552	A core contract where users deposit LP tokens and receive their earnings.

Note: The Concentrator IFO version is upgraded from the previous version, and this audit focuses on the update part and the overall smart contract architecture design. Compared to the previous version of the `AladdinConvexVault.sol` *contract, only minor changes have been made to the underlying core code. This audit focused on the changed code and the files mentioned above. In addition, Other Vyper contracts in* `concentrator/token` *and concentrator/rewarder directory are forked from*

Curve DAO with minor parameter changes.

3. Contract Analysis

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

3.1 Role Classification

There are two key roles in the Concentrator IFO: Governance Account and Common Account.

- Governance Account
  - Description

    Contract administrator
  - Authority
    - Update basic parameters
    - Add new Convex pool
    - Transfer ownership
  - Method of Authorization

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

    Users participate in the Concentrator IFO.
  - Authority
    - Stake LP token by `AladdinConvexVault`.
    - Retrieval of yield from Convex
    - Convex yield reinvestment
  - Method of Authorization

No authorization required

3.2 Functional Analysis

The Concentrator IFO contract allows Convex liquidity providers to stake their LP tokens and get diversified benefits. The SECBIT team conducted a detailed audit of this contract, and the essential functions in the contract are as follows.

ConcentratorIFOVault

This contract deals with the Initial Farming Offering (IFO) phase of user revenue. During the IFO phase, the user's earnings under the Concentrator protocol will be distributed in the form of CTR tokens. At the end of the IFO phase, or if the expected number of CTR tokens is reached earlier, the old version of the code logic will continue to be used.

The main functions in `ConcentratorIFOVault` are as below:

- `claimCONT()`

  This function allows the user to claim pending CTR tokens from a specific pool.

- `claimAllCONT()`

  This function allows the user to claim pending CTR tokens from all pools.

- `harvest()`

  This function retrieves the pending award from the Convex protocol. During the IFO phase, these rewards will be replaced with CTR tokens, and after the IFO phase, the normal reward collection mechanism will resume.

AladdinConvexVault

This contract supports users to deposit LP tokens which will be deposited under the Convex protocol. The user will be rewarded with aCRV tokens based on the amount of LP tokens deposited.

New functions added to this contract are as below:

- `migrate()`

  This function allows the user to migrate all user shares to another vault.

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 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 Discussion of the `harvest()` caller reward design.

Risk Type	Risk Level	Impact	Status
Design & Implementation	Info	Design logic	Discussed

Description

The new Concentrator protocol will produce CTR tokens based on the earnings under the Convex protocol, at which point the user of this protocol will receive the CTR token earnings. The `harvest()` function is used to retrieve the earnings from the Convex protocol, convert them into cvxCRV tokens, and then deposit all the cvxCRV tokens into the AladdinCRV contract to obtain the corresponding aCRV token. Ultimately, CTR tokens are minted as the user's reward based on the number of aCRV tokens. At the IFO phase, the rewards retrieved by the `harvest()` function are used to reward the Concentrator protocol user, and the caller of `harvest()` is not rewarded in any way, so there is no incentive for the caller to call the `harvest()` function. It is necessary to confirm the design logic of the contract here.

```
function harvest(
    uint256 _pid,
    address _recipient,
    uint256 _minimumOut
  ) external override onlyExistPool(_pid) nonReentrant returns
(uint256 harvested) {
    PoolInfo storage _pool = poolInfo[_pid];

    // 1. harvest and convert to aCRV
    uint256 _rewards;
```

```
    (harvested, _rewards) = _harvestAsACRV(_pid, _minimumOut);

    // 2. do IFO if possible
    // solhint-disable-next-line not-rely-on-time
    if (startTime <= block.timestamp && block.timestamp <=
endTime) {
      uint256 _pendingCONT = MAX_MINED_CONT - contMined;
      if (_pendingCONT > _rewards) {
        _pendingCONT = _rewards;
      }
       if (_pendingCONT > 0) {
        accCONTPerShare[_pid] =
accCONTPerShare[_pid].add(_pendingCONT.mul(PRECISION) /
_pool.totalShare);

        contMined += uint128(_pendingCONT);

        // Vault Mining $CONT
        ICONT(cont).mint(address(this), _pendingCONT);

        // Liquidity Mining $CONT
        // @audit send rewards to the specified rewarder
address
        ICONT(cont).mint(rewarder, (_pendingCONT * 6) / 100);

        // transfer aCRV to platform
        IERC20Upgradeable(aladdinCRV).safeTransfer(platform,
_pendingCONT);

        emit IFOMineCONT(_pendingCONT);
      }

      _rewards -= _pendingCONT;
    }

    if (_rewards > 0) {
      // 3. distribute rewards to platform and _recipient
      address _token = aladdinCRV; // gas saving
      uint256 _platformFee = _pool.platformFeePercentage;
```

```
        uint256 _harvestBounty = _pool.harvestBountyPercentage;
        if (_platformFee > 0) {
          _platformFee = (_platformFee * _rewards) /
FEE_DENOMINATOR;
          _rewards = _rewards - _platformFee;
          IERC20Upgradeable(_token).safeTransfer(platform,
_platformFee);
        }

        //@audit rewards for caller
        if (_harvestBounty > 0) {
          _harvestBounty = (_harvestBounty * _rewards) /
FEE_DENOMINATOR;
          _rewards = _rewards - _harvestBounty;
          IERC20Upgradeable(_token).safeTransfer(_recipient,
_harvestBounty);
        }

        // 4. update rewards info
        _pool.accRewardPerShare =
_pool.accRewardPerShare.add(_rewards.mul(PRECISION) /
_pool.totalShare);

        emit Harvest(msg.sender, _rewards, _platformFee,
_harvestBounty);
      }
    }
```

Status

The team fixed the issue in commit b542c8a.

4.3.2 Add restrictions on start time and end time to improve the security of parameter settings.

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

Description

When setting the IFO start time parameter `_startTime` and the end time parameter `_endTime` it is recommended to add restrictions on the relationship between these two parameters to improve the security of the parameter settings.

```
function updateIFOConfig(
    address _rewarder,
    address _cont,
    uint64 _startTime,
    uint64 _endTime
  ) external onlyOwner {
    rewarder = _rewarder;
    cont = _cont;

    //@audit add judgment condition
    startTime = _startTime;
    endTime = _endTime;

    emit UpdateIFOConfig(_rewarder, _cont, _startTime,
_endTime);
  }
```

Suggestion

Corresponding changes are proposed as follows.

```
function updateIFOConfig(
    address _rewarder,
    address _cont,
    uint64 _startTime,
    uint64 _endTime
  ) external onlyOwner {
    require(_startTime < _endTime, "invalid time");
    rewarder = _rewarder;
    cont = _cont;
```

```
        startTime = _startTime;
        endTime = _endTime;

        emit UpdateIFOConfig(_rewarder, _cont, _startTime,
    _endTime);
      }
```

Status

The team fixed the issue in commit b542c8a.

5. Conclusion

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

Disclaimer

SECBIT smart contract audit service assesses the contract's correctness, security, and performability in code quality, logic design, and potential risks. The report is provided "as is", without any warranties about the code practicability, business model, management system's applicability, and anything related to the contract adaptation. This audit report is not to be taken as an endorsement of the platform, team, company, or investment.

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 possibly 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