Security Audit Report

AladdinCRV (aCRV) V2 by AladdinDAO

SECBIT

February 3, 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. **In response to the change in the revenue mechanism of the Convex protocol, the AladdinDAO team has updated the corresponding revenue logic**. SECBIT Labs conducted an audit from January 23 to February 1, 2023, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the AladdinCRV (aCRV) V2 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
Initialization	4.3.1 Potential contract initialization risks.	Info	Eliminated after initialization
Design & Implementation	4.3.2 Discussion of `migrateStrategy()` function.	Info	Discussed
Documentation	4.3.3 Wrong comment in `_swapCRVTocvxCRV()`.	Info	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the Concentrator contract is shown below:

- Project website
  - https://concentrator.aladdin.club/
- Smart contract code
  - https://github.com/AladdinDAO/aladdin-v3-contracts
  - commit *b40219b*

2.2 Contract List

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

Name	Lines	Description
AladdinCRVV2.sol	404	The aCRV token issuance contract represents the share of cvxCRV tokens deposited in the contract.
ConcentratorStrategyBase.sol	52	Concentrator strategy base contract.
cvxCRVStakingWrapperStrategy.sol	55	Proxy contract for depositing users' funds under the Convex protocol and receiving the yield.

Notice: The `ConcentratorStrategyBase.sol` *contract adds the* `_transferTokenBack()` *function to the already audited code for sending ether or token.*

3. Contract Analysis

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

3.1 Role Classification

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

- Governance Account
  - Description

    Contract administrator
  - Authority
    - Update basic parameters
    - Add new Convex pools
    - Transfer ownership
  - 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.
  - 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 allows Convex liquidity providers to stake their LP tokens and get diversified benefits. **The new strategy adapts the code logic to fit the changes in the Convex protocol**. 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:

AladdinCRVV2

This contract uses the aCRV token to record the cvxCRV tokens deposited into the contract, which will be deposited into Convex for revenue. In addition, this protocol also allows the deposit of CRV tokens or stcvxCRV tokens.

The main functions in `AladdinCRVV2` are as below:

- `deposit()`

  This function allows the user to deposit cvxCRV tokens stored directly into the Convex protocol. At the same time, the recipient specified by the user will receive the corresponding share of aCRV tokens.

- `mint()`

  This function will calculate the number of cvxCRV tokens to be deposited based on the number of aCRV tokens the user wants to obtain.

- `withdraw()`

  The user calls this function to retrieve the deposited cvxCRV token, which also requires the aCRV token to be burned in the appropriate amount.

- `depositWithCRV()`

  The user can call this function to deposit a specified number of CRV tokens into the contract, convert them into cvxCRV tokens, and deposit them in the Convex protocol.

- `depositWithWrapper()`

  This function allows users to deposit stkcvxCRV tokens to this contract.

- `harvest()`

It allows anyone to call this function to retrieve the proceeds of this contract in the Convex protocol. These proceeds will be converted into cvxCRV tokens and deposited again in the Convex protocol.

cvxCRVStakingWrapperStrategy & ConcentratorStrategyBase

The contract receives tokens from users and deposits them in the Convex protocol. At the same time, this address will receive the stkcvxCRV tokens as a credential.

The main functions in these contracts are as below:

- `deposit()`

  This function deposits cvxCRV tokens into the Convex protocol and receives the corresponding stkcvxCRV tokens credentials.

- `withdraw()`

  This function withdraws the cvxCRV token deposited under the Convex protocol, which requires that the stkcvxCRV token be burned simultaneously.

- `harvest()`

  Specified users can call this function to harvest the pending reward and convert it to `_intermediate` tokens.

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 Potential contract initialization risks.

Risk Type	Risk Level	Impact	Status
Initialization	Info	Design logic	Eliminate after initialization

Description

Once the contract upgrade is complete, the operator must execute the `initializeV2()` function to migrate the funds from the old strategy to the new one. However, the `initializeV2()` function is unrestricted, so anyone can call it during initialization, which may pose a security risk. If a malicious user calls this function first and transfers the funds under the contract to the malicious target address, it could cause unacceptable losses. Because this function is an initialization function and can only be executed once, this risk will be solved when the operator calls this function in time.

```
function initializeV2(address _strategy) external {
    require(strategy == address(0), "initialized");
    strategy = _strategy;

    // make sure harvest is called before upgrade.

 require(IConvexBasicRewards(cvxCRV_STAKING).earned(address(th
is)) == 0, "not harvested");

    // withdraw all cvxCRV from staking contract
    uint256 _totalUnderlying =
IConvexBasicRewards(cvxCRV_STAKING).balanceOf(address(this));
```

```
 IConvexBasicRewards(cvxCRV_STAKING).withdraw(_totalUnderlying
, false);

    // transfer all cvxCRV to strategy contract.
    IERC20Upgradeable(cvxCRV).safeTransfer(_strategy,
_totalUnderlying);
    IConcentratorStrategy(_strategy).deposit(address(0),
_totalUnderlying);
    totalUnderlying = _totalUnderlying;

    // approves
    IERC20Upgradeable(CRV).safeApprove(curvecvxCRVPool,
uint256(-1));
    IERC20Upgradeable(CRV).safeApprove(CRV_DEPOSITOR,
uint256(-1));
  }
```

Status

This issue has been discussed. The team will call the `initializeV2()` function during the upgrade process.

4.3.2 Discussion of `migrateStrategy()` function.

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

Description

The `migrateStrategy()` function is used to migrate the funds under the strategy to the new one. In the current code, the `prepareMigrate()` function and the `finishMigrate()` function is empty, and only the interfaces for these two functions are provided. Therefore it is necessary to verify that the `migrateStrategy()` function is complete.

```
/// @notice Migrate pool assets to new strategy.
  /// @param _newStrategy The address of new strategy.
  function migrateStrategy(address _newStrategy) external
onlyOwner {
    require(_newStrategy != address(0), "AladdinCRV: zero new
strategy address");

    uint256 _totalUnderlying = totalUnderlying;
    address _oldStrategy = strategy;
    strategy = _newStrategy;


 IConcentratorStrategy(_oldStrategy).prepareMigrate(_newStrate
gy);
    IConcentratorStrategy(_oldStrategy).withdraw(_newStrategy,
_totalUnderlying);

 IConcentratorStrategy(_oldStrategy).finishMigrate(_newStrateg
y);

    IConcentratorStrategy(_newStrategy).deposit(address(this),
_totalUnderlying);

    emit MigrateStrategy(_oldStrategy, _newStrategy);
  }
```

Status

This issue has been discussed, and the development team confirmed this logic.

4.3.3 Wrong comment in `_swapCRVTocvxCRV()`.

Risk Type	Risk Level	Impact	Status
Documentation	Info	Wrong comments of code	Discussed

Description

The parameter `_lockIncentive` represents an incentive to users who spend gas to lock CRV tokens. When this value is not 0, the user is actively locking their CRV token. There is an error in the comment here.

```
function _swapCRVTocvxCRV(uint256 _amountIn, address
_recipient) internal returns (uint256) {
      ......
      uint256 _lockIncentive =
IConvexCRVDepositor(CRV_DEPOSITOR).lockIncentive();
      _amountOut =
IERC20Upgradeable(cvxCRV).balanceOf(address(this));
      if (_lockIncentive == 0) {
        // no lock incentive, use `lock = false`
        IConvexCRVDepositor(CRV_DEPOSITOR).deposit(_amountIn,
false, address(0));
      } else {
         //@audit The comment is wrong
        // no lock incentive, use `lock = true`
        IConvexCRVDepositor(CRV_DEPOSITOR).deposit(_amountIn,
true, address(0));
      }
      _amountOut =
IERC20Upgradeable(cvxCRV).balanceOf(address(this)) -
_amountOut; // never overflow here
      if (_recipient != address(this)) {
        IERC20Upgradeable(cvxCRV).safeTransfer(_recipient,
_amountOut);
      }
    }
    return _amountOut;
  }
```

Status

This issue has been discussed.

5. Conclusion

After auditing and analyzing the AladdinCRV (aCRV) V2 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