Security Audit Report

Concentrator aFXN by AladdinDAO

SECBIT

July 26, 2024

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 LP assets to farm top-tier DeFi tokens at the highest APY possible. The current Concentrator protocol has introduced new strategies, allowing users to deposit FXN tokens into the Convex protocol and earn yields. SECBIT Labs conducted an audit from July 18 to July 25, 2024, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator aFXN 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 on `isTokenProtected` Parameter.	Info	Fixed
Design & Implementation	4.3.2 Remove the `_intermediate` parameter and related checks to save gas.	Info	Fixed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about Concentrator aFXN is shown below:

- Smart contract code
    - initial review commit *a35453e*
    - final review commit *55c8911*

2.2 Contract List

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

Name	Lines	Description
CvxFxnCompounder.sol	88	Together with the ConcentratorCompounderBase contract, it forms a complete functionality, providing users with FXN token deposit and withdrawal services.
CvxFxnStakingStrategy.sol	117	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.

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 aFXN 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 aFXN.

- Authority
    - Deposit FXN 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 FXN 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:

CvxFxnCompounder and ConcentratorCompounderBase

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

- `deposit()`

Through this function, users deposit cvxFXN tokens into the contract. These assets will be transferred to the Convex protocol, and users will receive corresponding shares.

- `mint()`

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

- `withdraw()`

Users specify the amount of cvxFXN 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 cvxFXN tokens to withdraw.

- `harvest()`

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

- `depositWithStkCvxFxn()`

Users deposit stkCvxFxn tokens into the contract through this function.

- `depositWithFXN()`

Users deposit FXN tokens into the contract through this function.

CvxFxnStakingStrategy

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 cvxFXN 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 cvxFXN 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 Discussion on `isTokenProtected` Parameter.

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

Location

CvxFxnStakingStrategy.sol#L71-L74

Description

The `isTokenProtected` parameter manages unexpected token transfers under the contract, preventing the administrator from maliciously transferring the principal and rewards. This parameter is only set in the constructor, which means it cannot be updated (added or removed) through function interfaces after the contract is deployed. This leads to the following two consequences:

(1). If the Convex protocol updates the corresponding token rewards, this contract cannot synchronously update the corresponding tokens.

(2). It should be noted that user funds will ultimately be converted into stkCvxFxn tokens and stored under this strategy contract. The current contract does not set stkCvxFxn tokens to protected mode, so the administrator can directly transfer stkCvxFxn tokens under the contract through the `sweepToken()` function. This should violate the intended design of the `isTokenProtected` parameter.

```
constructor(address _operator) initializer {
  address[] memory _rewards = new address[](3);
  _rewards[0] = FXN; // FXN
  _rewards[1] = 0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B; // CVX
  _rewards[2] = 0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0; // wstETH

  __ConcentratorStrategyBase_init(_operator, _rewards);

  // approval
  IERC20(cvxFXN).safeApprove(staker, type(uint256).max);
  IERC20(FXN).safeApprove(FXN_DEPOSITOR, type(uint256).max);
  IERC20(FXN).safeApprove(CURVE_POOL, type(uint256).max);

  // protect token
  isTokenProtected[cvxFXN] = true;
  isTokenProtected[FXN] = true;
  isTokenProtected[0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B] = true;
// CVX
  isTokenProtected[0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0] = true;
// wstETH
}

/// @notice Sweep non-protected tokens from this contract.
/// @param _tokens The list of tokens to sweep.
function sweepToken(address[] memory _tokens) external onlyOwner {
  for (uint256 i = 0; i < _tokens.length; i++) {
    if (isTokenProtected[_tokens[i]]) revert TokenIsProtected();
 }
  _sweepToken(_tokens);
}
```

Status

The development team has fixed this issue in commit 55c8911.

4.3.2 Remove the `_intermediate` parameter and related checks to save gas.

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

Location

CvxFxnStakingStrategy.sol#L113

Description

The `_intermediate` parameter is not actually used in the code, so it can be omitted, along with the checks for this parameter in the code, to save gas.

```
function harvest(address _converter, address _intermediate)
  external
  override
  onlyOperator
  returns (uint256 _harvested)
{
  if (_intermediate != FXN) revert IntermediateNotFXN();

  // 0. sweep balances
  address[] memory _rewards = rewards;
  _sweepToken(_rewards);

 ......
}
```

Suggestion

The recommended modification is as follows:

```
// remove the '_intermediate' parameter
function harvest(address _converter, address ) //_intermediate
  external
  override
```

```
    onlyOperator
    returns (uint256 _harvested)
  {
    // @audit remove the following code
    // if (_intermediate != FXN) revert IntermediateNotFXN();

    // 0. sweep balances
    address[] memory _rewards = rewards;
    _sweepToken(_rewards);

   ......
  }
```

Status

The development team has fixed this issue in commit 55c8911.

5. Conclusion

After auditing and analyzing the Concentrator aFXN 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.