Security Audit Report

aFXS V2 by AladdinDAO

SECBIT

August 15, 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 new version of the aFXS (v2) protocol allows users to directly deposit Staked CvxFxs tokens or FXS tokens, and in return, they will receive aFXS tokens representing their proportional share of funds. Users holding aFXS tokens can continue to receive profits. SECBIT Labs conducted an audit from July 24 to August 15, 2023, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator aFXS (v2) protocol 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 contract redundantly implements functions to update the profit token type.	Info	Fixed
Design & Implementation	4.3.2 Discussion on the logic of exchanging FXS tokens for cvxFXS tokens.	Info	Discussed
Design & Implementation	4.3.3 The absence of authorization for the `FXS_DEPOSITOR` contract in the `CvxFxsStakingStrategy` contract will result in a failure when sending FXS tokens.	Medium	Fixed
Design & Implementation	4.3.4 Discussion on the function `_approve()` authorization limit.	Info	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the aFXS (v2) contract is shown below:

- Smart contract code
  - initial review commit *10920aa*
  - final review commit *22593de*

2.2 Contract List

The following content shows the contracts included in Aladdin aFXS (v2) protocol, which the SECBIT team audits:

Name	Lines	Description
AladdinFXSV2.sol	177	It inherits the `AladdinCompounder` contract and implements specific functions such as deposit, withdrawal, and reward distribution.
CvxFxsStakingStrategy.sol	98	The strategy contract for managing user funds and earnings.
PlatformFeeSpliter.sol	145	Claim and distribute pending rewards to staker/treasury/locker/ecosystem contract.
BurnerBase.sol	35	A base contract to handle dust assets.
ConvexFraxCompounderBurner.sol	17	Burn compounder asset and claim unlocked by this contract.
PlatformFeeBurner.sol	50	Convert the platform fees received by the contract into the specified token and send them to the designated address.
StakeDAOCompounderBurner.sol	17	Convert all tokens in this contract with given routes.
ConverterBase.sol	35	A base contract for converting tokens.
ConverterRegistry.sol	24	Withdraw dust assets from a converter contract.
GeneralTokenConverter.sol	473	Auxiliary contract for token swapping.

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 Aladdin aFXS (v2) protocol: Governance Account and Common Account.

- Governance Account
  - Description

    Contract Administrator
  - Authority
    - Update basic parameters
    - Update the percentage of various fees charged
    - 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 Aladdin aFXS (v2) protocol.
  - Authority
    - Deposit Staked CvxFxs tokens or FXS tokens and claim rewards
    - Harvest gains from Convex and distribute it
  - Method of Authorization

    No authorization required

3.2 Functional Analysis

The new contract allows users to deposit Staked CvxFxs tokens or FXS tokens, which will be transferred to the Convex protocol to earn interest. 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:

AladdinFXSV2

This contract is an abstract contract, which, together with the AladdinCompounder contract, forms a complete system. Users deposit Staked CvxFxs tokens or FXS tokens into this contract to receive a share of aFXS tokens. And as time goes on, the aFXS tokens held by the user will be redeemed for more cvxFxs tokens.

The main functions in `AladdinFXSV2` are as below:

- `depositWithStkCvxFxs()`

  This function allows the user to deposit the Staked CvxFxs tokens in this contract, and these tokens will be transferred to the Convex protocol simultaneously.
- `depositWithFXS()`

  This function allows the user to deposit the cvxFxs tokens in this contract, and these tokens will be transferred to the Convex protocol simultaneously.
- `harvest()`

  Users can call this function to harvest rewards from the Convex protocol.

CvxFxsStakingStrategy

This contract deposits users' cvxFxs tokens into the Convex protocol and earns corresponding profits. The main functions in `CvxFxsStakingStrategy` are as below:

- `deposit()`

This function deposits cvxFxs tokens into the Convex protocol, and in return, this contract receives an equivalent amount of Staked CvxFxs tokens as a deposit certificate.

- `withdraw()`

  This function withdraws cvxFxs tokens from the Convex protocol and simultaneously burns an equivalent amount of Staked CvxFxs tokens from this contract.

- `harvest()`

  Users can call this function to retrieve revenue from the Convex protocol and distribute it to all users.

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 contract redundantly implements functions to update the profit token type.

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

Description

Administrators can update users' reward tokens through the `updateRewards()` function in the `AladdinFXSV2` contract. Additionally, anyone can read and update the latest user reward token type through the `syncRewardToken()` function in the `CvxFxsStakingStrategy` contract. Both of them serve the purpose of updating the reward tokens. Comparatively, the `syncRewardToken()` function doesn't require users to provide any parameters and is more accurate. It's necessary to confirm the intended usage of both functions. It is recommended to keep only the `syncRewardToken()` function and remove the unnecessary `updateRewards()` function.

```
// @audit located in AladdinFXSV2.sol
function updateRewards(address[] memory _rewards) external
onlyOwner {
// @audit invoke the function in the `CvxFxsStakingStrategy.sol`
contract
  IConcentratorStrategy(strategy).updateRewards(_rewards);
}

// @audit located in the `ConcentratorStrategyBase.sol`
contract, inherited by the `CvxFxsStakingStrategy.sol` contract
function updateRewards(address[] memory _rewards) public virtual
override onlyOperator {
    _checkRewards(_rewards);
```

```
    delete rewards;
    rewards = _rewards;
  }

// @audit located in CvxFxsStakingStrategy.sol
function syncRewardToken() external {
    delete rewards;

    uint256 _length =
ICvxFxsStaking(staker).rewardTokenLength();
    for (uint256 i = 0; i < _length; i++) {
      rewards.push(ICvxFxsStaking(staker).rewardTokens(i));
    }
  }
```

Status

The team confirmed this issue and removed the `updateRewards()` function in commit fb8e08f.

4.3.2 Discussion on the logic of exchanging FXS tokens for cvxFXS tokens.

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

Description

When exchanging FXS tokens for cvxFXS tokens, the contract provides two options. The first one involves using the FXS - cvxFXS pool in the Curve protocol. When exchanging for cvxFXS tokens, there are uncertainties such as slippage, making it difficult to determine the exact quantity of tokens the user will receive. The second option is to use the FXS_DEPOSITOR contract to deposit FXS tokens and receive an equivalent amount of cvxFXS tokens. Before exchanging for cvxFXS tokens, the contract first estimates the number of cvxFXS tokens that can be obtained by calling the `get_dy()` function in the Curve protocol, and this

estimate is stored in the `_amountOut` variable. Based on this value, the contract then decides which exchange method is more favorable. However, the parameter `_amountIn` does not accurately represent the number of cvxFXS tokens that will be obtained by depositing FXS tokens into the `FXS_DEPOSITOR` contract. Therefore, the value of the `useCurve` parameter obtained from this might not be accurate. Assuming that the incentive value `_lockIncentive` in the `FXS_DEPOSITOR` contract is not 0, the actual quantity of cvxFXS tokens that the user can receive will be `_amountIn + _lockIncentive`.

```
// @audit located in AladdinFXSV2.sol
function _swapFXSToCvxFXS(uint256 _amountIn, address _recipient)
internal returns (uint256) {
    // CRV swap to cvxFXS or stake to cvxFXS
    uint256 _amountOut =
ICurveCryptoPool(CURVE_FXS_cvxFXS_POOL).get_dy(0, 1, _amountIn);
    bool useCurve = _amountOut > _amountIn;

    if (useCurve) {
      IERC20Upgradeable(FXS).safeApprove(CURVE_FXS_cvxFXS_POOL,
0);
      IERC20Upgradeable(FXS).safeApprove(CURVE_FXS_cvxFXS_POOL,
_amountIn);

      _amountOut =
ICurveCryptoPool(CURVE_FXS_cvxFXS_POOL).exchange_underlying(0,
1, _amountIn, 0, _recipient);
    } else {
      uint256 _lockIncentive =
IConvexFXSDepositor(FXS_DEPOSITOR).incentiveFxs();
      // if use `lock = false`, will possible take fee
      // if use `lock = true`, some incentive will be given
      _amountOut =
IERC20Upgradeable(cvxFXS).balanceOf(address(this));
      if (_lockIncentive == 0) {
        // no lock incentive, use `lock = false`
        IConvexFXSDepositor(FXS_DEPOSITOR).deposit(_amountIn,
false);
      } else {
        // no lock incentive, use `lock = true`
```

```
            IConvexFXSDepositor(FXS_DEPOSITOR).deposit(_amountIn,
true);
        }
        _amountOut =
IERC20Upgradeable(cvxFXS).balanceOf(address(this)) - _amountOut;
// never overflow here
        if (_recipient != address(this)) {
            IERC20Upgradeable(cvxFXS).safeTransfer(_recipient,
_amountOut);
        }
    }
    return _amountOut;
 }
```

Status

This issue has been discussed. No need to change.

4.3.3 The absence of authorization for the `FXS_DEPOSITOR` contract in the `CvxFxsStakingStrategy` contract will result in a failure when sending FXS tokens.

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

Description

In the contract `CvxFxsStakingStrategy`, when calling the `deposit()` function in the `FXS_DEPOSITOR` contract, there is a lack of authorization to that contract. It can fail to send FXS tokens.

```
// @audit located in CvxFxsStakingStrategy.sol
function _swapFXSToCvxFXS(uint256 _amountIn, address _recipient)
internal returns (uint256) {
    // CRV swap to cvxFXS or stake to cvxFXS
    uint256 _amountOut =
ICurveCryptoPool(CURVE_FXS_cvxFXS_POOL).get_dy(0, 1, _amountIn);
```

```
    bool useCurve = _amountOut > _amountIn;

    if (useCurve) {
     ......
    } else {
      uint256 _lockIncentive =
IConvexFXSDepositor(FXS_DEPOSITOR).incentiveFxs();
      // if use `lock = false`, will possible take fee
      // if use `lock = true`, some incentive will be given
      _amountOut = IERC20(cvxFXS).balanceOf(address(this));
      if (_lockIncentive == 0) {

        // @audit lack of authorization
        // no lock incentive, use `lock = false`
        IConvexFXSDepositor(FXS_DEPOSITOR).deposit(_amountIn,
false);
      } else {

        // @audit lack of authorization
        // no lock incentive, use `lock = true`
        IConvexFXSDepositor(FXS_DEPOSITOR).deposit(_amountIn,
true);
      }
      ......
  }
```

Suggestion

Authorize the transfer of FXS tokens to the `FXS_DEPOSITOR` contract in the constructor function:

```
constructor(address _operator) {
    address[] memory _rewards = new address[](2);
    _rewards[0] = 0x3432B6A60D23Ca0dFCa7761B7ab56459D9C964D0; //
FXS
    _rewards[1] = 0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B; //
CVX

    _initialize(_operator, _rewards);

    IERC20(cvxFXS).safeApprove(staker, uint256(-1));

    // @audit add the following code
    IERC20(FXS).safeApprove(FXS_DEPOSITOR, uint256(-1));
  }
```

Status

The team confirmed this issue and modified the logic in commit fb8e08f.

4.3.4 Discussion on the function `_approve()` authorization limit.

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

Description

We recommend to avoid excessive token approval to mitigate potential security risks. It appears that the authorization amount in the `_approve()` function here should be `_amount` rather than the infinite value `uint256(-1)` used currently.

```
//@audit located in ConverterBase.sol

function _approve(
    address _token,
    address _spender,
    uint256 _amount
  ) internal {
```

```
    if (!_isETH(_token) &&
IERC20(_token).allowance(address(this), _spender) < _amount) {
      // hBTC cannot approve 0
      if (_token != 0x0316EB71485b0Ab14103307bf65a021042c6d380)
{
        IERC20(_token).safeApprove(_spender, 0);
      }

      // @audit approve `_amount` ?
      IERC20(_token).safeApprove(_spender, uint256(-1));
    }
  }
```

Status

This issue has been discussed. From the perspective of saving gas, the development team has decided to keep the code unchanged.

5. Conclusion

After auditing and analyzing the Aladdin aFXS (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