Smart Contract
Security Audit Report

Prepared for Aladdin DAO

Prepared by Supremacy

December 28, 2023

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Aladdin DAO Concentrator aCVX, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

Aladdin DAO is a decentralized builder and incubator of cutting edge DeFi protocols. Our current focus is in building new approaches to yield farming optimization and automation within the Curve ecosystem and with our latest product, f(x) Protocol, we are expanding to scalable decentralized stablecoins.

To date Aladdin has built three products: Concentrator, Clever, and f(x) Protocol. Each of these protocols offers powerful new tools for DeFi users and adds new, composable DeFi money legos which will help form the basis of what we believe will be the decentralized future of global finance.

Concentrator is a yield enhancer that boosts yields on Convex vaults by concentrating all rewards into auto-compounding top-tier tokens like aCRV (cvxCRV), aFXS (cvxFXS), aCVX (CVX), and asdCRV (sdCRV).

Item	Description
Client	Aladdin DAO
Project	Concentrator aCVX
Website	https://concentrator.aladdin.club
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/AladdinDAO/aladdin-v3-contracts/tree/main/contracts
- Commit Hash: 2ebd6bfbabd5d9ddd2e3ddc9d1bb6fba4316f10e

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
`./concentrator/cvx/CvxCompounder.sol`	`ac4ddc3a31d43514531c7045711d8a56`
`./concentrator/cvx/CvxStakingStrategy.sol`	`2d2b6e52b8f03a889bb712e87d9bd439`
`./common/rewards/distributor/LinearRewardDistributor.sol`	`ccd8fcedd6bcf76334528ecb1d39d74f`
`./concentrator/ConcentratorCompounderBase.sol`	`49d3cf5f9be198be9e306d47b579fa62`
`./concentrator/strategies/AutoCompoundingStrategyBaseV2.sol`	`c313c3543a574f0c863711377061b50f`
`./concentrator/strategies/ConcentratorStrategyBase.sol`	`a26d70c794d347aeefa3e51298a66659`
`./concentrator/strategies/ConcentratorStrategyBaseV2.sol`	`83baa3b1ea9d959050457790710bee83`

1.3 Changelogs

Version	Data	Description
0.1	December 18, 2023	Initial Draft
1.0	December 28, 2023	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

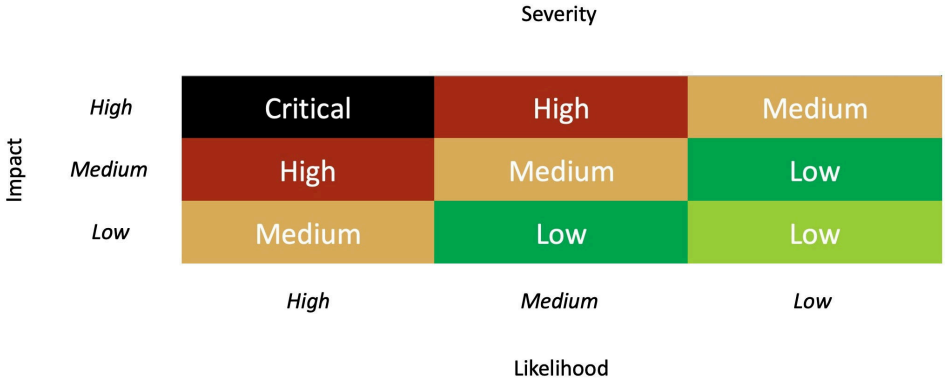
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	Centralization risk	Fixed
2	Medium	The potential freezing of funds	Confirmed
3	Low	Non-compliance with EIP-4626 specification	Confirmed
4	Low	The potential denial-of-service	Acknowledged
5	Informational	Lack of pause feature	Acknowledged
6	Informational	Lack of address validation	Acknowledged
7	Informational	Lack of event records	Acknowledged
8	Informational	Lack of comments	Confirmed

2.1 Medium

1. Centralization risk [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Fixed`

Description:

In the Concentrator aCVX, `migrateStrategy()` is used to migrate of strategies pool, and its main logic is to transfer assets from `oldStrategy` to `newStrategy`. However, the access control for this function is based on the `onlyRole(DEFAULT_ADMIN_ROLE)` modifier implementation. But, the `DEFAULT_ADMIN_ROLE` privileged account is held by the deployer, which would normally be an EOA account.

Our analysis shows that privileged accounts need to be scrutinized. In the following, we will examine privileged accounts and the associated privileged access in the current contract.

Note that if the privileged owner account is a plain EOA, this may be worrisome and pose counter-party risk to the protocol users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

```
276         /************************
277          * Restricted Functions *
278          ************************/
279
280         /// @inheritdoc IConcentratorCompounder
281         function migrateStrategy(address _newStrategy) external
    onlyRole(DEFAULT_ADMIN_ROLE) {
```

```
282         if (_newStrategy == address(0)) revert StrategyIsZero();
283
284         // This is the actual assets deposited into strategy.
285         (uint256 _distributable, uint256 _undistributed) = pendingRewards();
286         uint256 _totalAssets = totalAssets + _distributable + _undistributed +
    rewardData.queued;
287
288         address _oldStrategy = strategy;
289         strategy = _newStrategy;
290
291         IConcentratorStrategy(_oldStrategy).prepareMigrate(_newStrategy);
292         IConcentratorStrategy(_oldStrategy).withdraw(_newStrategy,
    _totalAssets);
293         IConcentratorStrategy(_oldStrategy).finishMigrate(_newStrategy);
294
295         IConcentratorStrategy(_newStrategy).deposit(address(this),
    _totalAssets);
296
297         emit Migrate(_oldStrategy, _newStrategy);
298       }
```

ConcentratorCompounderBase.sol

```
23   function initialize(
24     string memory _name,
25     string memory _symbol,
26     address _treasury,
27     address _harvester,
28     address _converter,
29     address _strategy
30   ) external initializer {
31     __AccessControl_init(); // from AccessControlUpgradeable
32     __ReentrancyGuard_init(); // from ReentrancyGuardUpgradeable
33     __ERC20_init(_name, _symbol); // from ERC20Upgradeable
34     __ERC20Permit_init(_name); // from ERC20PermitUpgradeable
35
36     __ConcentratorBaseV2_init(_treasury, _harvester, _converter); // from
    ConcentratorBaseV2
37     __LinearRewardDistributor_init(CVX); // from LinearRewardDistributor
38     __ConcentratorCompounderBase_init(_strategy); // from
    ConcentratorCompounderBase
39
40     // access control
41     _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
42   }
```

CvxCompounder.sol

Recommendation: Add a new address parameter to the `initialize()` function to make it a `DEFAULT_ADMIN_ROLE` privileged account.

Initially onboarding could can use multisign wallets or timelocks to initially mitigate centralization risks, but as a long-running protocol, we recommend eventually transfer the privileged account to the intended DAO-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks.

Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Feedback: After the aCVX is launched, the `default_admin_role` privilege has been transferred to multi-sig account.

2. The potential freezing of funds [Medium]

`Severity: Medium` `Likelihood: Medium` `Impact: Medium`

`Status: Confirmed`

Description:

`_sweepToken()` is used to sweep specified assets within a contract, in the current scenario `_transferToken()` takes into account the possibility of an native token, but its `#L152` code's way of fetching the balance of an asset is not compatible with native token. As a result, the desired `_sweepToken()` will always fail to perform as expected if native token are transferred into the contract in the future.

In this way, a denial-of-service will be created.

```
145   /// @dev Internal function to sweep tokens from this contract.
146   ///
147   /// @param _tokens The list of tokens to sweep.
148   function _sweepToken(address[] memory _tokens) internal {
149     address _stash = stash;
150     for (uint256 i = 0; i < _tokens.length; i++) {
151       address _token = _tokens[i];
152       uint256 _balance = IERC20(_token).balanceOf(address(this));
153       if (_balance > 0) {
154         _transferToken(_token, _stash, _balance);
155       }
156     }
157   }
```

ConcentratorStrategyBase.sol

```
159   /// @dev Internal function to transfer ETH or ERC20 tokens to some
    `_receiver`.
160   ///
161   /// @param _token The address of token to transfer, user `_token=address(0)`
    if transfer ETH.
162   /// @param _receiver The address of token receiver.
163   /// @param _amount The amount of token to transfer.
164   function _transferToken(
165     address _token,
166     address _receiver,
167     uint256 _amount
168   ) internal {
169     if (_token == address(0)) {
170       Address.sendValue(payable(_receiver), _amount);
171     } else {
172       IERC20(_token).safeTransfer(_receiver, _amount);
173     }
174   }
```

ConcentratorStrategyBase.sol

Recommendation: Add a new branch so that it correctly gets the contract's current native token balance.

2.2 Low

3. Non-compliance with EIP-4626 specification [Low]

`Severity: Medium` `Likelihood: Medium` `Impact: Medium`

`Status: Confirmed`

Description:

Contracts that integrate with the `ConcentratorCompounderBase` may wrongly assume that the functions are EIP-4626 specification, which it might cause integration problems in the future, that can lead to a wide range of issues for both parties, including loss of funds.

```
86    /// @inheritdoc IERC4626Upgradeable
87    function maxDeposit(address) external pure override returns (uint256) {
88      return type(uint256).max;
89    }
```

ConcentratorStrategyBase.sol

```
96    /// @inheritdoc IERC4626Upgradeable
97    function maxMint(address) external pure override returns (uint256) {
98      return type(uint256).max;
99    }
```

ConcentratorStrategyBase.sol

```
106   /// @inheritdoc IERC4626Upgradeable
107   function maxWithdraw(address) external pure override returns (uint256) {
108     return type(uint256).max;
109   }
```

ConcentratorStrategyBase.sol

```
126   /// @inheritdoc IERC4626Upgradeable
127   function maxRedeem(address) external pure override returns (uint256) {
128     return type(uint256).max;
129   }
```

ConcentratorStrategyBase.sol

EIP-4626 specification defines the withdraw function:

```
Maximum amount of the underlying asset that can be withdrawn from the owner balance
in the Vault, through a withdraw call.

MUST return the maximum amount of assets that could be transferred from owner through
withdraw and not cause a revert, which MUST NOT be higher than the actual maximum
that would be accepted (it should underestimate if necessary).

MUST factor in both global and user-specific limits, like if withdrawals are entirely
disabled (even temporarily) it MUST return 0.

MUST NOT revert.
```

Recommendation: The related functions should be modified to meet the specifications of EIP-4626.

4. The potential denial-of-service [Low]

`Severity: Low` `Likelihood: Low` `Impact: Medium`

`Status: Acknowledged`

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities.

The Openzeppelin's Ownable used in `ConcentratorStrategyBase` contract implements `renounceOwnership()`. This can represent a certain risk if the ownership is renounced for any other reason than by design. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.

```
13 // solhint-disable func-name-mixedcase
14 // solhint-disable no-empty-blocks
15
16 abstract contract ConcentratorStrategyBase is Initializable, Ownable2Step,
   IConcentratorStrategy {
```

ConcentratorStrategyBase.sol

```
8  /**
9   * @dev Contract module which provides access control mechanism, where
10  * there is an account (an owner) that can be granted exclusive access to
11  * specific functions.
12  *
13  * By default, the owner account will be the one that deploys the contract.
   This
14  * can later be changed with {transferOwnership} and {acceptOwnership}.
15  *
16  * This module is used through inheritance. It will make available all
   functions
17  * from parent (Ownable).
18  */
19 abstract contract Ownable2Step is Ownable {
```

Ownable2Step.sol

```
8       /**
9        * @dev Leaves the contract without owner. It will not be possible to call
10       * `onlyOwner` functions. Can only be called by the current owner.
11       *
12       * NOTE: Renouncing ownership will leave the contract without an owner,
13       * thereby disabling any functionality that is only available to the owner.
14       */
15      function renounceOwnership() public virtual onlyOwner {
16          _transferOwnership(address(0));
17      }
```

Ownable.sol

Recommendation: Either reimplement the function to disable it or to clearly specify if it is part of the contract design.

2.3 Informational

5. Lack of pause feature [Informational]

`Status: Acknowledged`

Description:

The pause feature is a key risk control component of smart contract(s), and it is necessary to add it. In this way, once malicious behavior occurs against the smart contract, access restrictions on key features can be completed in a timely manner. At the same time, there are already relatively mature libraries on the market, and it is recommended to use pausable utils to enable this feature.

6. Lack of address validation [Informational]

`Status: Acknowledged`

Description:

The `initialize()` function lacks zero address checking for the address parameters of multiple external contract(s), and the same is true for the corresponding function implementation.

```
23    function initialize(
24      string memory _name,
25      string memory _symbol,
26      address _treasury,
27      address _harvester,
28      address _converter,
29      address _strategy
30    ) external initializer {
31      __AccessControl_init(); // from AccessControlUpgradeable
32      __ReentrancyGuard_init(); // from ReentrancyGuardUpgradeable
33      __ERC20_init(_name, _symbol); // from ERC20Upgradeable
34      __ERC20Permit_init(_name); // from ERC20PermitUpgradeable
35
36      __ConcentratorBaseV2_init(_treasury, _harvester, _converter); // from
    ConcentratorBaseV2
37      __LinearRewardDistributor_init(CVX); // from LinearRewardDistributor
38      __ConcentratorCompounderBase_init(_strategy); // from
    ConcentratorCompounderBase
39
40      // access control
41      _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
42    }
```

CvxCompounder.sol

```
55    // solhint-disable-next-line func-name-mixedcase
56    function __LinearRewardDistributor_init(address _rewardToken) internal
    onlyInitializing {
57      rewardToken = _rewardToken;
58    }
```

LinearRewardDistributor.sol

```
51    /***************
```

```
52      * Constructor *
53      **********/
54
55      function __ConcentratorCompounderBase_init(address _strategy) internal
    onlyInitializing {
56          strategy = _strategy;
57      }
```

ConcentratorCompounderBase.sol

Recommendation: Add zero address validation.

7. Lack of event records [Informational]

`Status: Acknowledged`

Description:

Many smart contract(s) generally lack event records. However, events are important because off-chain monitoring tools rely on them to index important state changes to the smart contract(s).

Recommendation: Always ensure that all functions that trigger state changes have event logging capabilities.

8. Lack of comments [Informational]

`Status: Confirmed`

Description:

Throughout the codebase there are numerous functions missing or lacking documentation. This hinders reviewers' understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, comments improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Recommendation: Consider thoroughly documenting all functions (and their parameters) that are part of the smart contracts' public interfaces. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing comments, consider following the Ethereum Natural Specification Format (NatSpec).

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.