Security Audit Report

Concentrator asdPendle by AladdinDAO

SECBIT

August 8, 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 Pendle tokens into the Stake DAO protocol and earn yields. SECBIT Labs conducted an audit from July 27 to August 8, 2024, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator asdPendle 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 the handling logic of the protocol's boost fee.	Info	Discussed
Design & Implementation	4.3.2 Discussion on the logic for calculating relevant fees in the function `harvestBribe()`.	Info	Discussed
Design & Implementation	4.3.3 Discussion on the logic for distributing protocol rewards.	Info	Discussed
Design & Implementation	4.3.4 Discussion on the return value of the function `harvest()`.	Info	Fixed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about Concentrator asdPendle is shown below:

- Smart contract code
  - initial review commit *f2b4f77*
  - final review commit *6c71425*

2.2 Contract List

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

Name	Lines	Description
SdPendleBribeBurner.sol	79	This contract handles the bribe proceeds from Stake DAO protocol.
SdPendleCompounder.sol	109	Together with the ConcentratorCompounderBase contract, it forms a complete functionality, providing users with Pendle token deposit and withdrawal services.
SdPendleGaugeStrategy.sol	116	Together with the AutoCompoundingStrategyBaseV2 contract, it forms a complete contract where user deposits are funneled into the Stake DAO protocol. Additionally, this contract is responsible for managing the protocol's earnings.
SdPendleHelper.sol	28	A library contract for storing public variables and exchanging Pendle tokens for sdPendle tokens.

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 asdPendle 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 asdPendle.
    - Authority
        - Deposit sdPendle 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 Pendle tokens into the Stake DAO protocol via the new strategy in Concentrator can simultaneously receive earnings from both the Concentrator and Stake DAO 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:

SdPendleCompounder and ConcentratorCompounderBase

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

- `deposit()`

Through this function, users deposit sdPendle tokens into the contract. These assets will be transferred to the Stake DAO 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 sdPendle tokens they should deposit.

- `withdraw()`

Users specify the amount of sdPendle 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 sdPendle tokens to withdraw.

- `harvest()`

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

- `depositWithGauge()`

Users deposit Stake DAO sdPendle Gauge tokens into the contract through this function.

- `depositWithPENDLE()`

Users deposit Pendle tokens into the contract through this function.

- `harvestBribe()`

Anyone can claim the bribe rewards from the protocol through this function, and these rewards will be sent to the `bribeBurner` contract.

SdPendleGaugeStrategy

The strategy contract that connects the Concentrator and the Stake DAO protocol will facilitate the transfer of users' funds from this contract to the Stake DAO 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 Stake DAO protocol.

- `withdraw()`

This function withdraws sdPendle tokens from Stake DAO and sends them to the specified address.

- `harvest()`

This function assists users in claiming their earnings, converting them back into sdPendle tokens, and then staking them into the Stake DAO 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 the handling logic of the protocol's boost fee.

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

Location

SdPendleBribeBurner.sol#L103

Description

The bribe rewards in sdPendle tokens obtained from the protocol will be sent to the `SdPendleBribeBurner` contract and further managed by specific whitelisted users with the `WHITELIST_BURNER_ROLE`. When these whitelisted users process the sdPendle token rewards, a boost fee will be charged based on a specified ratio. This portion of sdPendle tokens will be exchanged for SDT tokens and sent to the `delegator` contract.

The current code does not specify the exact path for exchanging sdPendle tokens for SDT tokens. Observations of various DEXs on the market suggest that there is no direct trading pair for sdPendle/SDT, which may present the following issues:

(1). Multiple Trading Pairs Required: Exchanging sdPendle tokens for SDT tokens may require converting through multiple trading pairs, which incurs high transaction fees (including gas costs for executing the code and potential slippage). Since the amount of boost fee is not substantial, such operations might deplete most of the funds.

(2). Pricing and Value Discrepancies: Currently, it appears that only the sdPendle/Pendle trading pair exists under the Curve protocol for conducting sdPendle tokens exchanges. This pair is currently not in peg, meaning that exchanging 1 sdPendle tokens yields approximately only 0.6 Pendle tokens, resulting in a significant discount and potential loss. It is necessary to confirm whether this logic meets the requirements.

```
function burn(ConvertParams memory convertSDT) external
onlyRole(WHITELIST_BURNER_ROLE) {

  ......

  if (_boostFee > 0) {
    _boostFee = (_boostFee * _balance) / RATE_PRECISION;
    _rewards -= _boostFee;

    // @audit swap sdPendle tokens to SDT tokens
    _boostFee = _convert(sdPENDLE, SDT, _boostFee, convertSDT);
```

```
      IERC20(SDT).safeTransfer(delegator, _boostFee);
    }

    if (_rewards > 0) {
      IRewardDistributor(compounder).depositReward(_rewards);
    }
  }
```

Status

The development team has confirmed this issue. They explained that given the current sdPendle/Pendle liquidity, there does not appear to be a better solution.

4.3.2 Discussion on the logic for calculating relevant fees in the function `harvestBribe()`.

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

Location

SdPendleCompounder.sol#L150-L156

Description

When users call the `harvestBribe()` function to claim bribe rewards, these rewards are sent to the `bribeBurner` contract and further processed by whitelisted users. No relevant fees are collected during the invocation of the `harvestBribe()` function; hence, calculating corresponding fees at this point is unnecessary (and potentially inaccurate).

When users invoke burn() function, the actual fee to be collected is calculated based on the actual amount of sdPendle tokens present in this contract. Considering that the `bribeBurner` contract may receive unexpected amounts of sdPendle tokens, using an emit event to document the collected fees is more accurate and reasonable.

```
  function harvestBribe(IMultiMerkleStash.claimParam memory _claim)
  external {
      if (_claim.token != SdPendleHelper.sdPENDLE) revert
  ErrorInvalidBribeToken();

      // claim token to converter, and it will do the rest parts.
      IMultiMerkleStash.claimParam[] memory _claims = new
  IMultiMerkleStash.claimParam[](1);
```

```
    _claims[0] = _claim;
    StakeDAOBribeClaimer(bribeClaimer).claim(_claims, bribeBurner);

    // @audit fees will only be charged when the user calls the `burn()`
function in the `bribeBurner` contract.
    uint256 _amount = _claim.amount;
    uint256 _expenseRatio = getExpenseRatio();
    uint256 _boosterRatio = getBoosterRatio();
    uint256 _performanceFee = (_amount * _expenseRatio) /
RATE_PRECISION;
    uint256 _boosterFee = (_amount * _boosterRatio) / RATE_PRECISION;

    emit HarvestBribe(SdPendleHelper.sdPENDLE, _amount, _performanceFee,
_boosterFee);
}
```

Status

The development team explained that `event` in the `harvestBribe()` function records the information of the current harvest bribe. The calculated parameters `_performanceFee` and `_boosterFee` represent the expected fees to be collected. These parameters are not directly related to the fees collected under the `bribeBurner` contract.

4.3.3 Discussion on the logic for distributing protocol rewards.

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

Location

SdPendleGaugeStrategy.sol#L116-L117

Description

Due to the use of the `LOCKER.claimRewards()` function in the current code, which sets the array of tokens to be claimed to an `address[](0)` array, if no reward forwarding addresses have been set in advance in the `SD_PENDLE_GAUGE` contract for the `LOCKER` address, the rewards to be claimed will be directly sent to the `LOCKER` address and cannot be processed (the code will not revert). Therefore, before the protocol goes live, it must be ensured that the relevant reward forwarding addresses have been correctly set.

```
function harvest(address _converter, address _intermediate)
  external
  override
  onlyOperator
  returns (uint256 _harvested)
{
  // @audit the following check can be removed, as the `_intermediate`
parameter is not actually used.
  if (_intermediate != SdPendleHelper.PENDLE) revert
ErrorIntermediateNotPENDLE();

  // 1. claim rewards from LOCKER contract.
  address[] memory cachedRewards = rewards;

  // @audit it is necessary to first set the reward forwarding address
for the LOCKER address to the stash address in the SD_PENDLE_GAUGE
contract

 IConcentratorStakeDAOLocker(SdPendleHelper.LOCKER).claimRewards(SdPendl
eHelper.SD_PENDLE_GAUGE, new address[](0));
  uint256[] memory _amounts =
StakeDAOGaugeWrapperStash(stash).withdrawTokens(cachedRewards);

  address _registry = ITokenConverter(_converter).registry();

  ......
}
```

Status

The development team confirmed that the relevant parameters will be set before the official operation of the protocol.

4.3.4 Discussion on the return value of the function `harvest()`.

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

Location

SdPendleGaugeStrategy.sol#L169-L172

Description

The return value `_harvested` represents the amount of sdPendle tokens deposited into the `LOCKER` contract by the strategy. The `deposit()` function in the `LOCKER` contract directly deposits all adPendle tokens held by the `LOCKER` into the corresponding gauge. To prevent discrepancies between the `_harvested` return value and the actual amount deposited—due to unexpected sdPendle tokens being present in the `LOCKER` contract (transferred in from sources other than the strategy)—the return value of the `LOCKER.deposit()` function should be used as the return value for the `harvest()` function.

```
function harvest(address _converter, address _intermediate)
  external
  override
  onlyOperator
  returns (uint256 _harvested)
{
  ......

  // 5. transfer
  // part of the sdPENDLE is transferred to LOCKER in step 4, we only
transfer rest of them
  if (_harvested > 0) {
    IERC20(SdPendleHelper.sdPENDLE).safeTransfer(SdPendleHelper.LOCKER,
_harvested);
  }
  _harvested += _swapped;

  // 6. deposit
  if (_harvested > 0) {
    IConcentratorStakeDAOLocker(SdPendleHelper.LOCKER).deposit(
      SdPendleHelper.SD_PENDLE_GAUGE,
      SdPendleHelper.sdPENDLE
    );
  }

  return _harvested;
}

// @audit LOCKER.deposit()
function deposit(address _gauge, address _token) external override
onlyOperator(_gauge) returns (uint256 _amount) {
  _amount = IERC20Upgradeable(_token).balanceOf(address(this));
  if (_amount > 0) {
    IERC20Upgradeable(_token).safeApprove(_gauge, 0);
    IERC20Upgradeable(_token).safeApprove(_gauge, _amount);
    // deposit without claiming rewards
```

```
      ICurveGauge(_gauge).deposit(_amount);
    }
  }
```

Status

The development team has confirmed this issue and fixed it in commit 6c71425.

5. Conclusion

After auditing and analyzing the Concentrator asdPendle 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.