Security Audit Report

Furnace Update of CLever by AladdinDAO

SECBIT

Nov 11, 2022

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 CLever protocol helps users automatically lock CVX tokens into the Convex protocol. It will automatically retrieve the rewards and convert them into CVX tokens. The new version of the CLever protocol changes the revenue distribution strategy from a one-time distribution to a linear release. SECBIT Labs conducted an audit from Nov 7 to Nov 11, 2022, including an analysis of the smart contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Concentrator 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 Undistributed reward could be neglected in extreme cases.	Info	Discussed
Design & Implementation	4.3.2 Harvest reward could be neglected in extreme cases.	Info	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the CLever contract is shown below:

- Project website
  - https://clever.aladdin.club/
- Smart contract code
  - initial review commit *2372e66*
  - final review commit *595421f*

2.2 Contract List

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

Name	Lines	Description
MetaFurnace.sol	371	The generic version contract offers a service to exchange debt tokens for base tokens.
Furnace.sol	372	This contract offers a service to exchange clevCVX tokens for CVX tokens.

Notice: The above contracts have been modified from an old audited version in commit 8761cb3. We mainly focus on the modified parts.

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 CLever: 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 the transfer of the governance account.
- Common Account
    - Description

      Users participate in the Aladdin CLever.
    - Authority
        - Deposit CVX tokens and receive clevCVX tokens reward
        - Borrow clevCVX token
        - Exchange clevCVX tokens for CVX tokens
    - Method of Authorization

      No authorization required

3.2 Functional Analysis

The CLever protocol automatically locks the CVX token deposited by the user into the Convex protocol. It also provides the user with a leveraged lending service to significantly increase the user's revenue. The adjusted code changes the revenue distribution strategy for CVX tokens from a one-time revenue distribution to a linear release based on time. 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:

MetaFurnace

This contract provides a service to exchange debt tokens for base tokens. The main functions in `MetaFurnace` are as below:

- `deposit()`

  Deposit debt token in this contract to change for base token.

- `withdraw()`

  The user withdraws the unrealized debt token of the caller from this contract.

- `claim()`

  The user claims all realized base token of the caller from this contract.

- `distribute()`

  The whitelist users distribute base tokens from the origin address to pay the debt.

Furnace

This contract provides a service to exchange clevCVX tokens for CVX tokens. The user will have to wait if there are insufficient CVX tokens under this contract. The main functions in `Furnace` are as below:

- `deposit()`

  Deposit clevCVX token in this contract to change for CVX token.

- `withdraw()`

  The user retrieves the unexchanged clevCVX tokens to the specified address.

- `claim()`

  The user retrieves the exchanged CVX tokens and burns the corresponding clevCVX tokens.

- `harvest()`

  This function retrieves the contract's earnings and converts them into CVX 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 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 Undistributed reward could be neglected in extreme cases.

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

Description

The administrator can adjust the linear release period length with the `updatePeriodLength()` function. Consider the case where the administrator sets the `rewardInfo.periodLength` parameter to 0 during a revenue distribution period so that the remaining CVX tokens not distributed during that period will be neglected.

This issue exists in the `MetaFurnace.sol` and `Furnace.sol` contracts.

```
function updatePeriodLength(uint32 _length) external
onlyGovernorOrOwner {
   rewardInfo.periodLength = _length;

   emit UpdatePeriodLength(_length);
}

function _updatePendingDistribution() internal {
    LinearReward memory _info = rewardInfo;
    if (_info.periodLength > 0) {
      uint256 _currentTime = _info.finishAt;
      if (_currentTime > block.timestamp) {
        _currentTime = block.timestamp;
      }
```

```
        uint256 _duration = _currentTime >= _info.lastUpdate ?
_currentTime - _info.lastUpdate : 0;
        if (_duration > 0) {
          _info.lastUpdate = uint48(block.timestamp);
          rewardInfo = _info;

          _reduceGlobalDebt(_duration.mul(_info.ratePerSecond));
        }
      }
    }
```

Status

This issue has been discussed. This scenario related to administrator actions will not happen.

4.3.2 Harvest reward could be neglected in extreme cases.

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

Description

Any user can call the `harvest()` function to collect the reward, which eliminates the debt in the contract. Consider the following scenario: if the value of the parameter `rewardInfo.periodLength` is large, and the new reward added is small, then this reward amount will be neglected due to precision reasons. This effect is more severe in cases where the `harvest()` function is frequently called.

This issue exists in the `MetaFurnace.sol` and `Furnace.sol` contracts.

```
function _distribute(address _origin, uint256 _amount)
internal {
```

```
    // reduct pending debt
    _updatePendingDistribution();

    // distribute clevCVX rewards
    LinearReward memory _info = rewardInfo;
    if (_info.periodLength == 0) {
      _reduceGlobalDebt(_amount);
    } else {
      if (block.timestamp >= _info.finishAt) {
        _info.ratePerSecond = _toU128(_amount /
_info.periodLength);
      } else {
        uint256 _remaining = _info.finishAt - block.timestamp;
        uint256 _leftover = _remaining * _info.ratePerSecond;

        //@audit if the `amount` is too small, it may be
neglected for precision reasons.
        _info.ratePerSecond = _toU128((_amount + _leftover) /
_info.periodLength);
      }

      _info.lastUpdate = uint48(block.timestamp);
      _info.finishAt = uint48(block.timestamp +
_info.periodLength);

      rewardInfo = _info;
    }

   ......
  }
```

Status

This issue has been discussed. The above scenario is hard to happen in practice and has little impact.

5. Conclusion

After auditing and analyzing the Furnace update of the CLever 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