Security Audit Report

Collar Protocol

SECBIT

June 15, 2021

1. Introduction

The Collar Protocol is a decentralized lending protocol for pegged crypto assets featuring no liquidations. SECBIT Labs conducted an audit from June 7th to June 15th, 2021, including an analysis of the contract in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The assessment shows that the Collar protocol 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 In practice, the impact of decimals on the loan amount needs to be considered.	Info	Discussed
Gas Optimization	4.3.2 Using `external` function instead of `public` function to save gas.	Info	Discussed
Coding Style	4.3.3 For `require` condition, a valid error message can be used to pinpoint the cause of the error.	Info	Discussed
Design & Implementation	4.3.4 The purpose and implementation of `CHCCollect` is unclear.	Medium	Discussed
Parameter Settings	4.3.5 Some parameter settings still seem to be for testing purposes only.	Info	Discussed

2. Contract Information

This part describes the basic contract information and code structure.

2.1 Basic Information

The basic information about the Collar protocol contract is shown below:

- Project website
    - https://collar.org
- Smart contract code
    - https://github.com/CollarFinanceAudit/contracts, commit faf4bb2.

2.2 Contract List

The following content shows the contracts included in the Collar Protocol project:

Name	Lines	Description
Collar.sol	118	An abstract contract that provides the core code of the collar protocol.
CollarERC20.sol	15	Base ERC20 contract.
CIPSwap.sol	137	The core code used to handle the swap between bond tokens and want tokens.
CIPStaking.sol	70	Contract for participants to claim their rewards.
CHSLite.sol	24	Contract that provides users with the ability to borrow and lend.

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 the protocol, namely Governance Account and Common Account.

- Governance Account
  - Description Contract administrator
  - Authority
    - Update basic parameters
    - Update owner
  - Method of Authorization The creator of the contract, or authorized by the transferring of governance account
- Common Account
  - Description Users participate in lending activities
  - Authority
    - Provide collateral assets and lend out want token
    - Deposit bond token and get rewarded
  - Method of Authorization No authorization required

3.2 Functional Analysis

The Collar Protocol is a DeFi protocol that establishes money markets optimized for pegged assets. We can divide the critical functions of the auction contract into several parts:

Collar

This contract is used to manage bond tokens.

The main functions in `Collar` are as below:

- `mint_dual()`

  Users deposit bond tokens and receive an equal number of call tokens and coll tokens before the expiry time.
- `mint_coll()`

  Users deposit want tokens and receive an equal number of coll tokens before the expiry time.
- `burn_dual()`

  Users burn an equal number of call tokens and coll tokens and obtain bond tokens before the expiry time.
- `burn_coll()`

  Users burn coll tokens and repay want tokens to get bond tokens back before the expiry time.
- `burn_coll()`

  After expiry time, users burn coll tokens to obtain bond tokens and want tokens in proportion.

CIPSwap

The contract has two important parts: one is to convert coll tokens into want tokens; the other is to deposit want tokens to receive CLP tokens as a reward.

The main functions in `CIPSwap` are as below:

- `swap_coll_to_min_want()`

  This function converts coll tokens to want tokens.
- `swap_want_to_min_coll()`

  This function converts want tokens to coll tokens.

- `mint()`

  This function allows the user to obtain CLP tokens. The address holding the CLP token will be rewarded.

- `burn()`

  Users who burn CLP tokens will receive coll tokens and want tokens.

CIPStaking

Holders of CLP tokens will be rewarded with collar tokens in this contract.

The main functions in `CIPStaking` are as below:

- `claim_reward()`

  Users claim their rewards by this function. Rewards are allocated in the form of collar tokens.

- `start_reward()`

  The administrator sets the rate parameter for the user to receive the award.

CHSLite

This contract provides a lending interface for users, and it is the core contract of the project.

The main functions in `CHSLite` are as below:

- `borrow_want()`

  Users mortgage bond tokens and lend want tokens.

- `repay_both()`

  The borrower withdraws the mortgaged assets.

4. Audit Detail

This part describes the process, and the detailed results of the audit 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 bug, 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 two 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	✓
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 In practice, the impact of decimals on the loan amount needs to be considered.

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

Description

This project is a stable coin lending protocol, and its collateral assets and lending assets are all in stable coin. When the decimals of the collateral assets and the lending assets are not the same, it may impact the lending assets.

For example, when the collateral asset is USDT token (decimals of 6), and the lending asset is DAI token (decimals of 18), consider the following scenario. Suppose Alice mortgages 10 000 USDT token. Then she ends up lending 0.00 000 001 DAI token. Obviously, this does not follow the normal logic.

```
 // Located on CHSLite.sol
function borrow_want(uint256 nb, uint256 minnw) public {
    mint_dual(nb);
    swap_coll_to_min_want(nb, minnw);
}

// Located on Collar.sol
function mint_dual(uint256 n) public before_expiry {
    send_call(n);
    send_coll(n);
    recv_bond(n);
}
```

```
// Located on CIPSwap.sol
function swap_coll_to_min_want(uint256 dx, uint256 mindy)
public {
    uint256 fee = swap_fee();
    uint256 dy = calc_dy(sx, sy, dx);
    dy = (dy * fee) / 1e18;
    require(dy >= mindy, "insufficient fund");
    sx += dx;
    sy -= dy;
    recv_coll(dx);
    send_want(dy);
}
```

Status

The issue has been discussed. The developer team will set up appropriate functions for the specified stable coin pairs to deal with the problems caused by the different decimals.

4.3.2 Using `external` function instead of `public` function to save gas.

Risk Type	Risk Level	Impact	Status
Gas Optimization	Info	More gas consumption	Discussed

Description

The declaration of `public` functions that are never called by the contract should be declared `external` to save gas.

```
// Located on CHSLite.sol
function (uint256 nb, uint256 minnw) public {
    ......borrow_want
}

function repay_both(uint256 nw, uint256 no) public {
    ......
}

function withdraw_both(uint256 dk) public {
    ......
}

function get_dx(uint256 dy) public view returns (uint256) {
    ......
}

function get_dy(uint256 dx) public view returns (uint256) {
    ......
}
```

Suggestion

Use the `external` attribute for functions never called from the contract.

4.3.3 For `require` condition, a valid error message can be used to pinpoint the cause of the error.

Risk Type	Risk Level	Impact	Status
Coding Style	Info	Code readability	Discussed

Description

The `require()` function can be used to check for conditions and throw an exception if the condition is not met. Providing a concise description makes it easier to pinpoint the cause of the error.

```
// Located on Collar.sol
function expire() public {
    require(rate > 1e18);
    ......
    require(block.timestamp > time);
    ......
}

// Located on Collar.sol
modifier before_expiry() {
    ......
    require(block.timestamp < time);
    _;
}
```

Suggestion

Provide a concise description. One recommended modification is as follows.

```
// Located on Collar.sol
function expire() public {
    require(rate > 1e18, "prohibit recall");
    ......
    require(block.timestamp > time, "!expiry time");
    ......
}

// Located on Collar.sol
modifier before_expiry() {
    ......
    require(block.timestamp < time, "!expiry time");
    _;
}
```

4.3.4 The purpose and implementation of `CHCCollect` is unclear

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

Description

The CHCCollect contract does not import any header files. The intent of this contract is unclear. It seems to be used for the same purpose as the Collar::burn_coll function. However, the implementation of the burn_coll_want_only function seems to be faulty.

```
// Located on CHCCollect.sol
function burn_coll_want_only(uint256 n) public {
    require(rate == 1e18);
    recv_coll(n);
    send_coll(n); // @audit: send_want?
}

function burn_coll_bond_only(uint256 n) public {
    require(rate == 0);
    recv_coll(n);
    send_bond(n);
}
```

Suggestion

Import the appropriate header files. Revisit the functionality of this contract. Fix the implementation of the burn_coll_want_only function as below.

```
function burn_coll_want_only(uint256 n) public {
    require(rate == 1e18);
    recv_coll(n);
    send_want(n);
}
```

4.3.5 Some parameter settings still seem to be for testing purposes only

Risk Type	Risk Level	Impact	Status
Parameter Settings	Info	Formal deployment	Discussed

Description

Some of the parameter settings in the current code implementation appear to be for testing purposes only. It is not clear what the final setup and deployment process will be.

```
// Located on TestLocal.sol
function expiry_time() public pure override returns (uint256)
{
    return 4000000000; // @audit: for real?
}
```

```
// Located on CIPStaking.sol
function reward_end() public pure virtual returns (uint256) {
    return 2008000000; // @audit: year 2033
}
```

Suggestion

Confirm all undetermined parameters in the code.

5. Conclusion

After auditing and analyzing the Collar Protocol contract, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. The collar Protocol is a creative, liquidity-risk-free, stable coin lending protocol with high capital efficiency. SECBIT Labs holds the view that the Collar protocol smart contract has high code quality, concise implementation, and detailed documentation.

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 ethers 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 construct a common-consensus, reliable and ordered blockchain economic entity.

http://www.secbit.io

audit@secbit.io

@secbit_io