Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.07.19, the SlowMist security team received the Collar team's security audit application for Collar Finance, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit version：

FIle name: Collar Finance.zip

File hash: 650cda38f163d0604179f47da9122c4126866dca2621aad8a4bbb1cba94f1284

Code address: https://github.com/CollarFinanceAudit/contracts

commit: f2f4bb2b4c3aeefbaf51db0fbecd6c99150bd2eb

Fixed verison：

File name：contracts-20210604.zip

File hash：f55fcdc7402afbcc1ecc3ceda85b8d1c2c939f16b37bb036d3a9f7d5f40595f2

Code address：https://github.com/CollarFinanceAudit/contracts

commit：549a56cc71d465af51bf09a411983aa3c416478f

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	High	Confirming
N2	Business logic bug	Design Logic Audit	Suggestion	Confirming
N3	Potential compatibility issue	Others	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

Collar			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
mint_dual	Public	Can Modify State	before_expiry
mint_coll	Public	Can Modify State	before_expiry
burn_dual	Public	Can Modify State	before_expiry

Collar			
burn_call	Public	Can Modify State	before_expiry
burn_coll	Public	Can Modify State	-
expire	Public	Can Modify State	-
send_call	Internal	Can Modify State	-
send_coll	Internal	Can Modify State	-
recv_call	Internal	Can Modify State	-
recv_coll	Internal	Can Modify State	-
send_want	Internal	Can Modify State	-
send_bond	Internal	Can Modify State	-
recv_want	Internal	Can Modify State	-
recv_bond	Internal	Can Modify State	-
norm_bond_max	Public	-	-
norm_bond_min	Public	-	-
norm_want_max	Public	-	-
norm_want_min	Public	-	-
get_coll_norm	Internal	-	-
get_want_norm	Internal	-	-
expiry_time	Public	-	-
address_bond	Public	-	-
address_want	Public	-	-

Collar			
address_call	Public	-	-
address_coll	Public	-	-
address_collar	Public	-	-

TestLocal			
Function Name	Visibility	Mutability	Modifiers
address_bond	Public	-	-
address_want	Public	-	-
address_call	Public	-	-
address_coll	Public	-	-
address_collar	Public	-	-
expiry_time	Public	-	-

CIPStaking			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
claim_reward	Public	Can Modify State	-
start_reward	External	Can Modify State	onlyOwner
reward_per_token	Public	-	-
earned	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-

CIPStaking			
sync_reward	Internal	Can Modify State	-
sync_account	Internal	Can Modify State	-
reward_end	Public	-	-

CIPSwap			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
swap_coll_to_min_want	Public	Can Modify State	-
swap_want_to_min_coll	Public	Can Modify State	-
mint	Public	Can Modify State	-
burn	Public	Can Modify State	-
get_coll_norm	Internal	-	-
get_want_norm	Internal	-	-
sk	Public	-	-
calc_dx	Public	-	-
calc_dy	Public	-	-
calc_x	Public	-	-
calc_y	Public	-	-
calc_k	Public	-	-
sqrt	Internal	-	-

CIPSwap			
swap_sqp	Public	-	-
swap_fee	Public	-	-

CollarERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
mint	Public	Can Modify State	onlyOwner
burn	Public	Can Modify State	onlyOwner
drop	Public	Can Modify State	-

CHCCollect			
Function Name	Visibility	Mutability	Modifiers
burn_coll_want_only	Public	Can Modify State	-
burn_coll_bond_only	Public	Can Modify State	-

CHSLite			
Function Name	Visibility	Mutability	Modifiers
borrow_want	Public	Can Modify State	-
repay_both	Public	Can Modify State	-
withdraw_both	Public	Can Modify State	-
get_dx	Public	-	-

CHSLite			
get_dy	Public	-	-

4.3 Vulnerability Summary

[N1] [High] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The Owner role can mint arbitrarily through the mint function, and there is no upper limit on the number of minted tokens.

- implements/0x00/CollarERC20.sol#L10

```
function mint(address account, uint256 amount) public onlyOwner {
        _mint(account, amount);
    }
```

The Owner role can burn user's tokens arbitrarily through the burn function.

- implements/0x00/CollarERC20.sol#L14

```
function burn(address account, uint256 amount) public onlyOwner {
        _burn(account, amount);
    }
```

The Owner role can arbitrarily set the value of `reward_rate` through the start_reward function to change the reward rate at the beginning.

- implements/0x02/CIPStaking.sol#L36

```
function start_reward(uint256 rate) external onlyOwner {
        sync_reward();
        reward_rate = rate;
    }
```

Solution

1. CollarERC20.sol contract Owner has the risk of excessive authority. It is recommended to use a time lock mechanism or community governance to restrict.
2. It is recommended to add event records for the start_reward function and restriction on the range of `reward_rate` value.

Status

Confirming; After communicating with the project team, the project team will mint a certain amount of Collar tokens after the deployment of the CollarERC20.sol contract, and give the Owner permission to the zero address.

[N2] [Suggestion] Business logic bug

Category: Design Logic Audit

Content

In the burn_coll_want_only function, recv_coll will burn coll tokens, but send_coll mint the tokens back, which is a logical error.

- inheritances/code/CHCCollect.sol#L12

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

In the CIPStaking contract, calling the sync_account function alone will cause the updated amount of the `paid[account]` value to be incorrect. Although the contract is used in conjunction with the sync_reward function, there is still a possibility of errors in the use of subsequent inherited contracts.

- implements/0x02/CIPStaking.sol#L73

```
function sync_reward() internal {
        reward_per_token_stored = reward_per_token();
        last_time_updated = reward_end().min(block.timestamp);
    }

function sync_account(address account) internal {
        rewards[account] = earned(account);
        paid[account] = reward_per_token_stored;
    }
```

Solution

1. According to the project logic, if coll tokens are burned, want tokens should be minted back. It is recommended to change send_coll to send_want.
2. According to the project logic, it is recommended to change the visibility of the sync_account and sync_reward functions to private so that the inherited contract cannot be used.

Status

Confirming; The CHCollect.sol contract was caused by a mistake by the project team and has been repaired. After communicating with the project team, the project team decided to change the visibility of the sync_account and sync_reward functions to private, making the inherited contract unusable.

[N3] [Suggestion] Potential compatibility issue

Category: Others

Content

Notice: rebase stablecoins are not compatible with contract codes. When adding currencies, project team need to pay attention to the compatibility of such tokens and contracts.

Solution

It is recommended to audit the trading pair before adding liquidity to ensure that the trading pair and the item are compatible with each other.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0x002107270001	SlowMist Security Team	2021.07.19 - 2021.07.27	High Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 2 suggestion vulnerabilities. And 1 suggestion vulnerabilities were confirmed and being fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist