Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2023.08.02, the SlowMist security team received the team's security audit application for GoldCollateralManager, 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 team 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.
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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

This is the GoldCollateralManager token contract that contains the NFT mortgage section and does not contain the dark coin functions. The total amount of contract tokens can be changed, PHYSICAL_GOLD_MINTER_ROLE can burn their tokens through the burnBackedByPhysicalGold function and PHYSICAL_GOLD_MINTER_ROLE can also mint tokens through the mintBackedByPhysicalGold function. The contract does not have the Overflow and the Race Conditions issue. Users can deposit their goldNFT through the createNewCollateral function and gain the Gold Pegged Coin(GPC). And by repaying with KLAY and burn the GPC tokens to withdraw their goldNFT.

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 Audit	High	Acknowledged
N2	Reentrancy reminder	Reentrancy Vulnerability	Suggestion	Fixed
N3	The business logic is unclear	Design Logic Audit	Suggestion	Fixed
N4	Token return values not checked	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

Audit version:

https://github.com/CrederLabs/Gold-Collateral-Manager/blob/main/contracts/GoldCollateralManager.sol

commit: c9998b048878ea273a710adb9a59bab01858e04c

FIxed version:

https://github.com/CrederLabs/Gold-Collateral-Manager/blob/audit-

20230808/contracts/GoldCollateralManager.sol

commit: c9cf972363e2b60f77b672d6ded62c8aef795a18

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:

GoldCollateralManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20

GoldCollateralManager			
addAdminRole	Public	Can Modify State	onlyOwner
deleteAdminRole	Public	Can Modify State	onlyOwner
_transfer	Internal	Can Modify State	permissionCheck
createNewCollateral	Public	Can Modify State	whenNotPaused
registerCollateralExchangeAmount	Public	Can Modify State	onlyOwner
deleteCollateralExchangeAmount	Public	Can Modify State	onlyOwner
registerRepaymentFeeAmount	Public	Can Modify State	onlyOwner
deleteRepaymentFeeAmount	Public	Can Modify State	onlyOwner
findCollateralsByAddress	Public	-	-
getCollateralsLengthByAddress	Public	-	-
findCollateralIndexByAddressAndTokenId	Public	-	-
removeForCollateralIndexByAddress	Private	Can Modify State	-
findCollateralIndexByTokenId	Public	-	-
removeForCollateralTokenIds	Private	Can Modify State	-
getCollateralHistoryByAddress	Public	-	-
repay	Public	Payable	whenNotPaused
getTotalGPCSupply	Public	-	-
addPhysicalGoldMinter	Public	Can Modify State	onlyRole
deletePhysicalGoldMinter	Public	Can Modify State	onlyRole
mintBackedByPhysicalGold	Public	Can Modify State	onlyRole
burnBackedByPhysicalGold	Public	Can Modify State	onlyRole
getPhysicalGoldTotalSupply	Public	-	-

GoldCollateralManager			
recoverERC20	Public	Can Modify State	onlyOwner
recoverERC721	Public	Can Modify State	onlyOwner
recoverKLAY	Public	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [High] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the contract, the owner role can add the DEFAULT_ADMIN_ROLE, the DEFAULT_ADMIN_ROLE can add the PHYSICAL_GOLD_MINTER_ROLE, and the PHYSICAL_GOLD_MINTER_ROLE can call the mintBackedByPhysicalGold to mint the Gold Pegged Coin(GPC). There is no upper limitation on the mint amount of the GPC.

Code location:

GoldCollateralManager.sol#165-171, 350-372

```
    function addAdminRole(address _account) public onlyOwner() {
        _grantRole(DEFAULT_ADMIN_ROLE, _account);
    }

    function deleteAdminRole(address _account) public onlyOwner() {
        _revokeRole(DEFAULT_ADMIN_ROLE, _account);
    }

    function addPhysicalGoldMinter(address _account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        _grantRole(PHYSICAL_GOLD_MINTER_ROLE, _account);
    }

    function deletePhysicalGoldMinter(address _account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        _revokeRole(PHYSICAL_GOLD_MINTER_ROLE, _account);
    }

    function mintBackedByPhysicalGold(uint256 _gpcAmount, address _recipient) public
onlyRole(PHYSICAL_GOLD_MINTER_ROLE) {
        require(_recipient != address(0), "Invalid _recipient address");
        require(_gpcAmount > 0, "Invalid _gpcAmount");

        _mint(_recipient, _gpcAmount);

        totalCreatedPhysicalGold += _gpcAmount;
        mintAllPhysicalGoldHistory[msg.sender].push(PhysicalGoldHistory(
            msg.sender,
            _recipient,
            _gpcAmount,
            block.timestamp
        ));
        emit MintBackedByPhysicalGold(msg.sender, _gpcAmount, block.timestamp);
    }
```

2.The owner role can add, delet and change the amount and type of the collateralExchangeAmount and repaymentFeeAmount. If the user created the collateral and have not repayed yet, the owner role can change the gpcRepaymentAmount and the repaymentFeeAmount to repay more than before or the owner can delet the gpcRepaymentAmount and the repaymentFeeAmount to make the repay function revert.

Code location:

GoldCollateralManager.sol#220-240

```
    function registerCollateralExchangeAmount(uint16 _goldType, uint256 _gpcAmount)
public onlyOwner {
        require(_gpcAmount > 0, "Invalid _gpcAmount");
        collateralExchangeAmount[_goldType] = _gpcAmount;
        emit RegisterCollateralExchangeAmount(_goldType, _gpcAmount);
    }

    function deleteCollateralExchangeAmount(uint16 _goldType) public onlyOwner {
        delete collateralExchangeAmount[_goldType];
        emit DeleteCollateralExchangeAmount(_goldType);
    }

    function registerRepaymentFeeAmount(uint16 _goldType, uint256 _klayAmount) public
onlyOwner {
        require(_klayAmount > 0, "Invalid _klayAmount");
        repaymentFeeAmount[_goldType] = _klayAmount;
        emit RegisterRepaymentFeeAmount(_goldType, _klayAmount);
    }

    function deleteRepaymentFeeAmount(uint16 _goldType) public onlyOwner {
        delete repaymentFeeAmount[_goldType];
        emit DeleteRepaymentFeeAmount(_goldType);
    }
```

3.In the contract, the owner role can call the recoverERC721 function to transfer the NFT tokens in the contract to his address, and it also can transfer the goldNFTContract NFT.

Code location:

GoldCollateralManager.sol#398-401

```
    function recoverERC721(address _tokenAddress, uint256 _tokenId) public onlyOwner
 {
        IERC721(_tokenAddress).transferFrom(address(this), msg.sender, _tokenId);
        emit RecoverERC721(_tokenAddress, _tokenId);
    }
```

4.In the contract, the DEFAULT_ADMIN_ROLE can change the `maxMintingAmount` to change the upper limit of the restriction of the mint amout.

Code location:

GoldCollateralManager.sol#348-352

```
    function setMaxMintingAmount(uint256 _maxMintingAmount) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(_maxMintingAmount > 0, "Invalid _maxMintingAmount");
        maxMintingAmount = _maxMintingAmount;
        emit SetMaxMintingAmount(maxMintingAmount);
    }
```

Solution

1. It's recommended to transfer the owner role to the community governance or add a limitation on the amount of the token.
2. It's recommended to transfer the owner role to the community governance.
3. It's recommended to add the `require(_tokenAddress != goldNFTContract)` check.

Status

Acknowledged; After communication with the project team, they fixed the issue point 3 and added an upper limit of the mint amount called `maxMintingAmount` to the mintBackedByPhysicalGold function (Point 4). But the `maxMintingAmount` can be changed in the setMaxMintingAmount function and the mint amount is not restricted in the createNewCollateral function of the gpcSupplyAmount. The project team expressed that they control the issuance of GPC with NFT and they also expressed that they will not transfer the ownership to the community governance.

[N2] [Suggestion] Reentrancy reminder

Category: Reentrancy Vulnerability

Content

In the contract, the collateralIndexByAddress, token mint, totalCreatedGold increase, collateralTokenIds push and userAllCollateralHistory push are written after the call of the transferFrom the goldNFTContract ERC721 NFT. There is an external call hook onERC721Received exited in the goldNFTContract. And the goldNFTContract is not in the audit scope. It is the same as the repay function, the increase of the totalBurnedGold and the userAllCollateralHistory push after the transferFrom. And the goldNFTContract is not in the audit scope, it may have the reentrancy issue.

Code location:

GoldCollateralManager.sol#179-217, 296-342

```
    function createNewCollateral(uint256 _tokenId) public whenNotPaused {
        require(goldNFTContract.ownerOf(_tokenId) == msg.sender, "You don't own!");

        uint16 goldType =
TheMiningClubInterface(address(goldNFTContract)).getGoldTypeOfTokenId(_tokenId);
//SlowMist// @audit-info tokenid小于65535
        require(goldType > 0, "Invalid goldType");

        goldNFTContract.transferFrom(msg.sender, address(this), _tokenId);

        collaterals[_tokenId] = CollateralData(
            msg.sender,
            _tokenId,
            goldType,
            CollateralStatus.RECEIVED,
            block.timestamp
        );

        collateralIndexByAddress[msg.sender].push(_tokenId);

        uint256 gpcSupplyAmount = collateralExchangeAmount[goldType];
        require(gpcSupplyAmount > 0, "Invalid gpcSupplyAmount");

        // mint KIP-7(ERC-20)
        _mint(msg.sender, gpcSupplyAmount);

        // Info record (overflow check function added since 0.8.x or later. No need
to use SafeMath)
        totalCreatedGold += gpcSupplyAmount;
        collateralTokenIds.push(_tokenId);

        // Leave a record for history inquiry
        userAllCollateralHistory[msg.sender].push(CollateralHistory(
            msg.sender,
            _tokenId,
            goldType,
            CollateralStatus.RECEIVED,
            block.timestamp
        ));

        emit CreateNewCollateral(msg.sender, _tokenId, goldType, gpcSupplyAmount,
CollateralStatus.RECEIVED, block.timestamp);
    }

    function repay(uint256 _tokenId) payable public whenNotPaused {
        require(collaterals[_tokenId].userAccount == msg.sender, "Not matched
userAccount");
        require(collaterals[_tokenId].collateralStatus == CollateralStatus.RECEIVED,
"No received collateral");

        uint16 goldType =
TheMiningClubInterface(address(goldNFTContract)).getGoldTypeOfTokenId(_tokenId);
        require(goldType > 0, "Invalid goldType");

        uint256 gpcRepaymentAmount = collateralExchangeAmount[goldType];
        require(gpcRepaymentAmount > 0, "Invalid gpcRepaymentAmount");

        // 0.05g: 1 KLAY
        // 1g: 5 KLAY
        // 5g: 10 KLAY
        // 10g: 15 KLAY
        // 50g: 20 KLAY
        // 100g: 25 KLAY
        // 200g: 30 KLAY
        require(msg.value == repaymentFeeAmount[goldType], "Insufficient KLAY Fee");

        _burn(msg.sender, gpcRepaymentAmount);

        collaterals[_tokenId].collateralStatus = CollateralStatus.RETURNED;
        // Delete the collateral information from the user's address
        uint256 indexOfTokenId = findCollateralIndexByAddressAndTokenId(_tokenId);
        removeForCollateralIndexByAddress(msg.sender, indexOfTokenId);

        uint256 indexOfTokenId2 = findCollateralIndexByTokenId(_tokenId);
        removeForCollateralTokenIds(indexOfTokenId2);

        // Give back NFTs
        goldNFTContract.transferFrom(address(this), msg.sender, _tokenId);

        // Info record (overflow check function added since 0.8.x or later. No need
to use SafeMath)
        totalBurnedGold += gpcRepaymentAmount;

        // Leave a record for history inquiry
        userAllCollateralHistory[msg.sender].push(CollateralHistory(
            msg.sender,
            _tokenId,
            goldType,
            CollateralStatus.RETURNED,
            block.timestamp
        ));

        emit Repay(msg.sender, _tokenId, goldType, gpcRepaymentAmount,
CollateralStatus.RETURNED, block.timestamp);
    }
```

Solution

It is recommended to add the nonReentrant lock to prevent the unexpected callback reentrancy of the onERC721Received function or follow the Checks-Effects-Interactions principle.

Status

Fixed

[N3] [Suggestion] The business logic is unclear

Category: Design Logic Audit

Content

In the contract, the design of the GoldCollateralManager contract, two additional bookkeeping methods are designed in the current contract, balanceOf will record the total amount of mint and burn, while nft mortgage and repay is a way of bookkeeping data, the record is totalCreatedGold; then minter mint and burn tokens It is another way of accounting data, which is recorded as totalCreatedPhysicalGold. And when the owner has enought gpc coin and he can call the burnBackedByPhysicalGold function to burn off tokens that exceed totalCreatedPhysicalGold will cause revert in getPhysicalGoldTotalSupply function.

Solution

It is recommended to clarify the logic implementation.

Status

Fixed; After communication with the project team, they expressed that there is a method of issuing GPC with gold nft as collateral, and there is a process of exchanging real gold for GPC at about 100 of their offline branches. Therefore, the role was managed separately for the issuance of real gold.

[N4] [Suggestion] Token return values not checked

Category: Others

Content

Some tokens (like USDT) don’t correctly implement the EIP20 standard and their transfer/transferFrom function return void instead of a successful boolean. Calling these functions with the correct EIP20 function signatures will always revert. It is the same as the ERC721 transfer.

Code location:

GoldCollateralManager.sol#393-396

```
    function recoverERC20(address _tokenAddress, uint256 _amount) public onlyOwner {
        IERC20(_tokenAddress).transfer(msg.sender, _amount);
        emit RecoverERC20(_tokenAddress, _amount);
    }
```

Solution

It is recommended to use OpenZeppelin’s SafeERC20 versions with the safeTransfer and safeTransferFrom functions that handle the return value check as well as non-standard-compliant tokens.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002308070001	SlowMist Security Team	2023.08.02 - 2023.08.07	Medium 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 medium risk, 1 low risk, 6 suggestion vulnerabilities. All the findings were fixed or acknowledged. 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.