Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.05.21, the SlowMist security team received the MintSwap Finance team's security audit application for MintSwap Finance - NFT Marketplace, 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 an NFT market contract.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Listing Confusion Vulnerability	Design Logic Audit	Low	Acknowledged
N2	Cancel listing validation missing	Design Logic Audit	Low	Acknowledged

NO	Title	Category	Level	Status
	vulnerability			
N3	Cancel Bids validation missing vulnerability	Design Logic Audit	Low	Acknowledged
N4	Redundant code	Others	Suggestion	Fixed
N5	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

https://github.com/MintSwapFinance/mintswap-nft-marketplace

Initial audit commit: d08b45afa7b99e9ac0ec8e2994f341d68d7a39b4

Final audit commit: 26e06e65d0a127333c7738cd8376ee08a1d44805

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:

MintSwapNFTMarketplaceV1			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
initialize	External	Can Modify State	initializer
createOrUpdateListing	External	Can Modify State	nonReentrant whenNotPaused

MintSwapNFTMarketplaceV1			
_createListingWithoutEvent	Internal	Can Modify State	-
cancelListing	External	Can Modify State	nonReentrant
cancelBids	External	Can Modify State	nonReentrant
createOrUpdateTokenBid	External	Can Modify State	nonReentrant whenNotPaused whenBiddingActive
createOrUpdateCollectionBid	External	Can Modify State	nonReentrant whenNotPaused whenBiddingActive
_createBidWithoutEvent	Private	Can Modify State	-
acceptCollectionBid	External	Can Modify State	nonReentrant whenNotPaused whenBiddingActive
acceptTokenBid	External	Can Modify State	nonReentrant whenNotPaused whenBiddingActive
_acceptBid	Private	Can Modify State	-
buyItems	External	Payable	nonReentrant whenNotPaused
_buyItem	Private	Can Modify State	-
_payFees	Private	Can Modify State	-
_transferAmount	Private	Can Modify State	-
setSupportPaymentToken	Public	Can Modify State	onlyRole
setFee	Public	Can Modify State	onlyRole
setCollectionOwnerFee	External	Can Modify State	-
setFeeRecipient	Public	Can Modify State	onlyRole
setWeth	External	Can Modify State	onlyRole

MintSwapNFTMarketplaceV1			
toggleAreBidsActive	External	Can Modify State	onlyRole
pause	External	Can Modify State	onlyRole
unpause	External	Can Modify State	onlyRole

4.3 Vulnerability Summary

[N1] [Low] Listing Confusion Vulnerability

Category: Design Logic Audit

Content

In the MintSwapNFTMarketplaceV1 contract, if the user transfers the ownership of the token during the token listing period and does not delete the corresponding `listings` mapping, the new token owner will list the token again to generate a new `listings` mapping, which will cause the new and old `listings` mappings to coexist, making it impossible for buyers to distinguish which `listings` mapping is valid.

Solution

It is recommended to modify the recording method of listing information to ensure that when the ownership of the token changes, the corresponding listing information can be deleted in time.

Status

Acknowledged; The project team said that the off-chain process is to query the listing list from the database, and then on the one hand, the query will be filtered by expiration time, and on the other hand, the generation of new orders will change the status of old order data.

[N2] [Low] Cancel listing validation missing vulnerability

Category: Design Logic Audit

Content

In the MintSwapNFTMarketplaceV1 contract, the `cancelListing` function did not check whether the `listings` mapping to be canceled existed, causing the `ItemCanceled` event to be triggered arbitrarily.

- contracts/MintSwapNFTMarketplaceV1.sol#L344-L360

```
    function cancelListing(
        CancelListParams[] calldata _cancelListParams
    ) external nonReentrant {
        for (uint256 i = 0; i < _cancelListParams.length; i++) {
            CancelListParams calldata _cancelListParam = _cancelListParams[i];
            delete (
                listings[_cancelListParam.nftAddress][_cancelListParam.tokenId][
                    _msgSender()
                ]
            );
            emit ItemCanceled(
                _msgSender(),
                _cancelListParam.nftAddress,
                _cancelListParam.tokenId
            );
        }
    }
```

Solution

It is recommended to check whether the listing mapping to be deleted exists.

Status

Acknowledged; The project team stated that this problem can be avoided by users calling functions through the front end.

[N3] [Low] Cancel Bids validation missing vulnerability

Category: Design Logic Audit

Content

In the MintSwapNFTMarketplaceV1 contract, the `cancelBids` function did not check whether the `collectionBids` or `tokenBids` mappings to be canceled existed, causing the `TokenBidCancelled` event to be triggered arbitrarily.

- contracts/MintSwapNFTMarketplaceV1.sol#L362-L387

```
    function cancelBids(
        CancelBidParams[] calldata _cancelBidParams
    ) external nonReentrant {
        for (uint256 i = 0; i < _cancelBidParams.length; i++) {
```

```
            CancelBidParams calldata _cancelBidParam = _cancelBidParams[i];
            if (_cancelBidParam.bidType == BidType.COLLECTION) {
                collectionBids[_cancelBidParam.nftAddress][_msgSender()]
                    .quantity = 0;

                emit CollectionBidCancelled(
                    _msgSender(),
                    _cancelBidParam.nftAddress
                );
            } else {
                tokenBids[_cancelBidParam.nftAddress][_cancelBidParam.tokenId][
                    _msgSender()
                ].quantity = 0;

                emit TokenBidCancelled(
                    _msgSender(),
                    _cancelBidParam.nftAddress,
                    _cancelBidParam.tokenId
                );
            }
        }
    }
```

Solution

It is recommended to check whether the collectionBids or tokenBids mappings to be deleted exists.

Status

Acknowledged; The project team stated that this problem can be avoided by users calling functions through the front end.

[N4] [Suggestion] Redundant code

Category: Others

Content

In the MintSwapNFTMarketplaceV1 contract, there are no external calls to the `cancelListing` function and the `cancelBids` function, so there is no need to add the `nonReentrant` modifier.

- contracts/MintSwapNFTMarketplaceV1.sol#L344-L360, L362-L387

```
    function cancelListing(
        CancelListParams[] calldata _cancelListParams
    ) external nonReentrant {
        ```
```
```

```
    }

    function cancelBids(
        CancelBidParams[] calldata _cancelBidParams
    ) external nonReentrant {
        ```
    }
```

**Solution**

It is recommended to delete the redundant code.

**Status**

Fixed

### [N5] [Medium] Risk of excessive authority

**Category: Authority Control Vulnerability Audit**

**Content**

1.In the MintSwapNFTMarketplaceV1 contract, the `MARKETPLACE_ADMIN_ROLE` role can set important parameters in the contract through the following functions.

- contracts/MintSwapNFTMarketplaceV1.sol

```
    function setSupportPaymentToken
    function setFee
    function setFeeRecipient
    function setWeth
    function toggleAreBidsActive
```

2.In the MintSwapNFTMarketplaceV1 contract, the `MARKETPLACE_ADMIN_ROLE` role and collection owners can set the `collectionToCollectionOwnerFee` mapping via the `setCollectionOwnerFee` function.

- contracts/MintSwapNFTMarketplaceV1.sol#L838-L863

```
    function setCollectionOwnerFee(
        address _collectionAddress,
        CollectionOwnerFee calldata _collectionOwnerFee
    ) external {
        OwnableUpgradeable collection = OwnableUpgradeable(_collectionAddress);
        require(
            collection.owner() == _msgSender() ||
```
```

```
                hasRole(MARKETPLACE_ADMIN_ROLE, _msgSender()),
            "No permission"
        );
        require(
            _collectionOwnerFee.fee <= MAX_COLLECTION_FEE,
            "Collection fee too high"
        );

        // The collection recipient can be the 0 address, meaning we will treat this
as a collection with no collection owner fee.
        collectionToCollectionOwnerFee[
            _collectionAddress
        ] = _collectionOwnerFee;

        emit UpdateCollectionOwnerFee(
            _collectionAddress,
            _collectionOwnerFee.recipient,
            _collectionOwnerFee.fee
        );
    }
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. The authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002405240002	SlowMist Security Team	2024.05.21 - 2024.05.24	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 medium risk, 3 low risk, 1 suggestion.

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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist