Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.06.03, the SlowMist security team received the HOTCROSS team's security audit application for Cross Mint, 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

Initial audit files:

https://github.com/hotcrosscom/proj-cross-mint-solidity

commit: 2919d966fb8e6051ec817aa1a17043cdca9f1640

Final audit files:

https://github.com/hotcrosscom/proj-cross-mint-solidity

commit: 902041055a8d193a013c4db83e0025a1f1a9c36a

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status

NO	Title	Category	Level	Status
N1	Code redundancy issue	Others	Suggestion	Fixed
N2	Missing event records	Others	Suggestion	Confirmed
N3	Permission control issue	Authority Control Vulnerability	Suggestion	Fixed
N4	Return value checking issue	Others	Suggestion	Fixed

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:

CrossMint721Factory			
Function Name	Visibility	Mutability	Modifiers
setCrossMint	Public	Can Modify State	-
deployCrossMint721	External	Can Modify State	onlyCrossMint

CrossMint1155Factory			
Function Name	Visibility	Mutability	Modifiers

CrossMint1155Factory			
setCrossMint	Public	Can Modify State	-
deployCrossMint1155	External	Can Modify State	onlyCrossMint

CrossMint721			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	MinterControl ERC721
supportsInterface	Public	-	-
_baseURI	Internal	-	-
baseURI	Public	-	-
totalSupply	Public	-	-
mint	Public	Can Modify State	minterOnly
mintWithUri	Public	Can Modify State	minterOnly

CrossMint1155			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	MinterControl ERC1155
assertIdExists	Internal	-	-
supportsInterface	Public	-	-
totalSupply	External	-	-
create	Public	Can Modify State	minterOnly
createWithUri	Public	Can Modify State	minterOnly

CrossMint1155			
mint	Public	Can Modify State	existsOnly minterOnly
mintBatch	Public	Can Modify State	minterOnly

CrossMint1155BaseURI			
Function Name	Visibility	Mutability	Modifiers
baseURI	Public	-	-
uri	Public	-	-
_tokenURI	Internal	-	-
_setTokenURI	Internal	Can Modify State	-
_setBaseURI	Internal	Can Modify State	-

FeeManager			
Function Name	Visibility	Mutability	Modifiers
initialize	External	Can Modify State	initializer
setupFees	Public	Can Modify State	onlyManager
setMintFee	Public	Can Modify State	onlyManager
_setMintFee	Private	Can Modify State	-
setCollectionFee	Public	Can Modify State	onlyManager
_setCollectionFee	Private	Can Modify State	-
setFeeCollector	Public	Can Modify State	onlyManager
_setFeeCollector	Private	Can Modify State	-

FeeManager			
distributeMintFee	Public	Payable	-
distributeCollectionFee	Public	Payable	-

Guard			
Function Name	Visibility	Mutability	Modifiers
__Guard_init	Internal	Can Modify State	initializer
pause	Public	Can Modify State	onlyOwner
unpause	Public	Can Modify State	onlyOwner

ManagerControl			
Function Name	Visibility	Mutability	Modifiers
__ManagerControl_init	Internal	Can Modify State	initializer

MinterControl			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

CrossMint			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setBaseURI	External	Can Modify State	onlyOwner

CrossMint			
_setCrossMint721Factory	Private	Can Modify State	-
_setCrossMint1155Factory	Private	Can Modify State	-
setFactories	External	Can Modify State	onlyOwner
_setFeeManager	Private	Can Modify State	-
setFeeManager	External	Can Modify State	onlyOwner
totalCrossMint721Deployed	Public	-	-
totalCrossMint1155Deployed	Public	-	-
mintCrossMint721WithUri	External	Payable	withPermissions whenNotPaused withMintFee
mintCrossMint721	External	Payable	withPermissions whenNotPaused withMintFee
mintCrossMint1155	External	Payable	withPermissions whenNotPaused withMintFee
createCrossMint1155WithUri	External	Payable	withPermissions whenNotPaused withMintFee
createCrossMint1155	External	Payable	withPermissions whenNotPaused withMintFee
deployCrossMint721	Public	Payable	whenNotPaused withCollectionFee
deployCrossMint1155	Public	Payable	whenNotPaused withCollectionFee

4.3 Vulnerability Summary

[N1] [Suggestion] Code redundancy issue

Category: Others

Content

- The _beforeTokenTransfer function exists in the CrossMint1155 contract, but this function does not implement any logic and belongs to redundant code.
- In the CrossMint1155BaseURI contract, there is a `_clearTokenURI` function to clear `tokenId`. The visibility of this function is internal, but no other function calls this internal visibility function.

Code loction:

```
function _beforeTokenTransfer(
  address operator,
  address from,
  address to,
  uint256[] memory ids,
  uint256[] memory amounts,
  bytes memory data
) internal virtual override {
  // noop
}
```

```
function _clearTokenURI(uint256 tokenId) internal {
  if (bytes(_tokenURIs[tokenId]).length != 0) {
    delete _tokenURIs[tokenId];
  }
}
```

Solution

It is recommended to remove redundant code.

Status

Fixed

[N2] [Suggestion] Missing event records

Category: Others

Content

- In the CrossMint1155BaseURI contract, the `_setBaseURI` function is used to set the `baseURI`, but no event recording is performed.
- In the CrossMint contract, the owner can set `baseURI` and `feeManager` through the setBaseURI function and setFeeManager function respectively, but no event recording is performed.

Code location:

```
function _setBaseURI(string memory baseURI_) internal virtual {
  _baseURI = baseURI_;
}
```

```
function setBaseURI(string memory baseURI_) external onlyOwner {
  baseURI = baseURI_;
}

function _setFeeManager(FeeManager feeManager_) private {
  require(
    Misc.isContract(address(feeManager_)),
    "CrosMint: feeManager_ cannot be a non-contract address"
  );

  feeManager = feeManager_;
}

function setFeeManager(FeeManager feeManager_) external onlyOwner {
  _setFeeManager(feeManager_);
}
```

Solution

It is recommended to record events when setting sensitive parameters.

Status

Confirmed

[N3] [Suggestion] Permission control issue

Category: Authority Control Vulnerability

Content

In the CrossMint721Factory library, any user can create a new CrossMint721 contract through the deployCrossMint721 function; in the CrossMint1155Factory library, any user can create a new deployCrossMint1155 contract through the deployCrossMint721 function;

Code location:

```
function deployCrossMint721(
  bytes32 salt,
  string memory name,
  string memory symbol,
  string memory baseURI,
  address owner
) external returns (address) {
  bytes memory bytecode =
    abi.encodePacked(
      type(CrossMint721).creationCode,
      abi.encode(name, symbol, baseURI, owner)
    );

  return Create2.deploy(0, salt, bytecode);
}
```

```
function deployCrossMint1155(
  bytes32 salt,
  string memory name,
  string memory symbol,
  string memory baseURI,
  address owner
) external returns (address) {
  bytes memory bytecode =
    abi.encodePacked(
      type(CrossMint1155).creationCode,
      abi.encode(name, symbol, baseURI, owner)
    );
```

```
    return Create2.deploy(0, salt, bytecode);
  }
```

Solution

It is recommended to control these two functions to avoid unknown risks.

Status

Fixed

[N4] [Suggestion] Return value checking issue

Category: Others

Content

There are the withMintFee modifier and withCollectionFee modifier in the CrossMint contract, which respectively call the distributeMintFee function and the distributeCollectionFee function to distribute fees, and check the results of their execution. However, the distributeMintFee function and the distributeCollectionFee function both have return values, so it may be better to check the return value while checking the execution result.

Code location:

```
  modifier withMintFee() {
    (bool success, ) =
      address(feeManager).call{ value: msg.value }(
        abi.encodeWithSignature("distributeMintFee(address)", msg.sender)
      );

    require(success, "CrossMint: Failed to send mint fees");

    _;
  }

  modifier withCollectionFee() {
    (bool success, ) =
      address(feeManager).call{ value: msg.value }(
        abi.encodeWithSignature("distributeCollectionFee(address)", msg.sender)
      );

    require(success, "CrossMint: Failed to send collection fees");
```

```
    _;
}
```

Solution

It is recommended to check the return value of the function.

Status

Fixed; The project team removed the check on the execution result of external calls in the new version.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002106090002	SlowMist Security Team	2021.06.03 - 2021.06.09	Passed

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 4 suggestion vulnerabilities. And 1 suggestion vulnerabilities were confirmed and being fixed; All other findings were 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist