SLOWMIST

Smart Contract

Security Audit Report

Table Of Contents

1 Executive Summary

On 2025.04.09, the SlowMist security team received the CATProtocol team's security audit application for cat-token-box, 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.
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
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Unsafe External Call Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	The `unlockArgs.isUserSpend` lacks verification checks	Design Logic Audit	Low	Fixed
N2	constructor parameter lacks validation check	Design Logic Audit	Medium	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.

Audit version:

https://github.com/CATProtocol/cat-token-box/tree/v2/packages/sdk/src/contracts

Initial audit commit: 2c79705d3546bc28bff936339236afb96f0aa3dd

Final audit commit: be9691cd8a569e1748b3a8664fcc1ed62229b2ec

4.2 Visibility Description

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

CAT20		
@method	Parameter	Security Constraints
unlock	4/4	Ownership Verification
checkGuardState	4/4	State Integrity

CAT20Guard		
@method	Parameter	Security Constraints

CAT20Guard		
unlock	6/6	Token Balance Validation

CAT20StateLib		
@method	Parameter	Security Constraints
checkState	1/1	Token Validity

CAT20GuardStateLib		
@method	Parameter	Security Constraints
formalCheckState	1/1	Format Validation
checkTokenScriptsUniq	1/1	Uniqueness Validation

CAT20ClosedMinter		
@method	Parameter	Security Constraints
mint	7/7	Minting Authorization

CAT20OpenMinter		
@method	Parameter	Security Constraints
mint	8/8	Supply Control

CAT721		
@method	Parameter	Security Constraints
unlock	4/4	NFT Ownership
checkGuardState	4/4	NFT State Integrity

CAT721Guard		
@method	Parameter	Security Constraints

CAT721Guard		
unlock	6/6	NFT Collection Integrity

CAT721StateLib		
@method	Parameter	Security Constraints
checkState	1/1	NFT Validity

CAT721GuardStateLib		
@method	Parameter	Security Constraints
formalCheckState	1/1	State Format Validation
checkNftScriptsUniq	1/1	Script Uniqueness

CAT721ClosedMinter		
@method	Parameter	Security Constraints
mint	7/7	NFT Issuance Control

CAT721OpenMinter		
@method	Parameter	Security Constraints
mint	9/9	Merkle Tree Validation

CAT721OpenMinterMerkleTree		
@method	Parameter	Security Constraints
updateLeaf	5/5	Merkle Proof Verification
leafStateHash	1/1	Hash Integrity
checkLeaf	1/1	Leaf Validation

CAT721OpenMinterMerkleTree		
leafPropHashes	1/1	Property Hashing

OwnerUtils		
@method	Parameter	Security Constraints
toLockingScript	2/2	Script Generation
checkUserOwner	3/3	User Authentication
checkPubKey	2/2	Key Format Validation
checkOwnerAddr	1/1	Address Validation

4.3 Vulnerability Summary

[N1] [Low] The `unlockArgs.isUserSpend` lacks verification checks

Category: Design Logic Audit

Content

When handling unlock logic, the contract only relies on the length of `this.state.ownerAddr` to determine whether the token is owned by the contract or by the user, neglecting to check the `unlockArgs.isUserSpend` parameter.

- packages/sdk/src/contracts/cat20/cat20.ts

```
        if (len(this.state.ownerAddr) == OWNER_ADDR_CONTRACT_HASH_BYTE_LEN) {
            // unlock token owned by contract script
            assert(unlockArgs.contractInputIndexVal >= 0n &&
unlockArgs.contractInputIndexVal < this.ctx.inputCount);
            assert(this.state.ownerAddr ==
hash160(this.ctx.spentScripts[Number(unlockArgs.contractInputIndexVal)]));
        } else {
            // unlock token owned by user key
            OwnerUtils.checkUserOwner(unlockArgs.userPubKeyPrefix,
unlockArgs.userXOnlyPubKey, this.state.ownerAddr);
            assert(this.checkSig(unlockArgs.userSig, unlockArgs.userXOnlyPubKey));
        }
```

- packages/sdk/src/contracts/cat721/cat721.ts

```
       if (len(this.state.ownerAddr) == OWNER_ADDR_CONTRACT_HASH_BYTE_LEN) {
           // unlock token owned by contract script
           assert(unlockArgs.contractInputIndexVal >= 0n &&
unlockArgs.contractInputIndexVal < this.ctx.inputCount);
           assert(this.state.ownerAddr ==
hash160(this.ctx.spentScripts[Number(unlockArgs.contractInputIndexVal)]));
       } else {
           // unlock token owned by user key
           OwnerUtils.checkUserOwner(unlockArgs.userPubKeyPrefix,
unlockArgs.userXOnlyPubKey, this.state.ownerAddr);
           assert(this.checkSig(unlockArgs.userSig, unlockArgs.userXOnlyPubKey));
       }
```

Solution

Add verification for `unlockArgs.isUserSpend`.

Status

Fixed; Removed `isUserSpend`.

[N2] [Medium] constructor parameter lacks validation check

Category: Design Logic Audit

Content

In the constructor of the CAT20OpenMinter contract, there is an important mathematical relationship constraint: premineCount * limit == premine. According to the code comments, this relationship should be enforced, but the actual code does not implement this check.

Additionally, all Int32 types are not verified to be greater than 0.

- packages/sdk/src/contracts/cat20/minters/cat20OpenMinter.ts

```
    constructor(
        genesisOutpoint: ByteString,
        maxCount: Int32,
        premine: Int32,
        premineCount: Int32,
        limit: Int32,
        premineAddr: ByteString,
    ) {
        super(...arguments);
        this.genesisOutpoint = genesisOutpoint;
        this.maxCount = maxCount;
```

```
       // this assumes this.premineCount * this.limit == this.premine,
       // which can be trivially validated by anyone after the token is deployed
       this.premine = premine;
       this.premineCount = premineCount;
       this.limit = limit;
       this.preminerAddr = premineAddr;
   }
```

In the CAT721OpenMinter contract, maxCount and premine need to be greater than 0.

- packages/sdk/src/contracts/cat721/minters/cat721OpenMinter.ts

```
   constructor(genesisOutpoint: ByteString, maxCount: Int32, premine: Int32,
premineAddr: ByteString) {
       super(...arguments);
       this.genesisOutpoint = genesisOutpoint;
       this.max = maxCount;
       this.premine = premine;
       this.preminerAddr = premineAddr;
   }
```

Solution

Add necessary mathematical relationship validation in the constructor.

Status

Fixed; The construction parameter is a static contract script for construction, and verification logic cannot be added.

The mathematical relationship is verified by calling checkProps in CAT20OpenMinterCovenant.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002504110002	SlowMist Security Team	2025.04.09 - 2025.04.11	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 1 medium risk, 1 low risk vulnerabilities.

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