Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2022.06.29, the SlowMist security team received the team's security audit application for HashKeyDID, 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.

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:

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

Serial Number	Audit Class	Audit Subclass
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
		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

Audit Version

contract file:

HashKeyDID-Contracts.zip

sha256:

0cb06be9119c6fc85c4ca105634ad3319640b4d432e70267eef3f9862d33e546

Fix Version

contract file:

HashKeyDID-Contracts.zip

sha256:

ad2bd5c5ed3f5b06f220d2bc2b91785d52769155ee5807c2a50167516c49e571

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	Medium	Ignored
N2	Risk of initial operation	Authority Control Vulnerability	Suggestion	Ignored
N3	Missing Zero-Address check	Others	Suggestion	Ignored
N4	Missing scope limit	Others	Suggestion	Ignored
N5	The authenticity of the tokenId is not verified	Others	Suggestion	Ignored
N6	Missing event record	Others	Suggestion	Ignored

NO	Title	Category	Level	Status
N7	Signature replay issue	Replay Vulnerability	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.

4.2 Visibility Description

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

DeedGrain			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC1155
mint	Public	Can Modify State	onlyControllers
burn	Public	Can Modify State	onlyControllers
reverseTransferable	Public	Can Modify State	onlyControllers
setSupply	Public	Can Modify State	onlyControllers
setBaseUri	Public	Can Modify State	-
uri	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-

DGIssuer			
Function Name	Visibility	Mutability	Modifiers
setDGMinterAddr	Public	Can Modify State	onlyOwner
issueDG	Public	Can Modify State	-
setSigner	Public	Can Modify State	onlyOwner
setTokenSupply	Public	Can Modify State	-
setTokenBaseUri	Public	Can Modify State	onlyOwner
mintDG	Public	Can Modify State	-
claimDG	Public	Can Modify State	-
_validate	Internal	-	-

DidV1			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
_setBaseURI	Internal	Can Modify State	-
_baseURI	Internal	-	-
setDidMinterAddr	Public	Can Modify State	onlyOwner
transferOwnership	Public	Can Modify State	onlyOwner
claim	Public	Can Modify State	-
mint	Public	Can Modify State	-

DidV1			
_mintDid	Internal	Can Modify State	-
verifyDIDFormat	Public	-	-
addAuth	Public	Can Modify State	-
removeAuth	Public	Can Modify State	-
getAuthorizedAddrs	Public	-	-
isAddrAuthorized	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-

DidV1Storage			
Function Name	Visibility	Mutability	Modifiers

EternalStorageProxy			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Payable	TransparentUpgradeableProxy

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The controller role has the right to mint any number of NFTs of any type to any address up to the total supply.

Code location:contracts/DeedGrain.sol #L33-38

```
    function mint(address to, uint256 tokenId, uint256 amount) public onlyControllers
{
        if (_supplies[tokenId] != 0) {
            require(totalSupply(tokenId) + amount <= _supplies[tokenId],
"insufficient supply");
        }
        _mint(to, tokenId, amount, "");
    }
```

The controller role has the right to burn any number of NFTs of any kind at any address.

Code location:contracts/DeedGrain.sol #L40-42

```
    function burn(address from, uint256 tokenId, uint256 amount) public
onlyControllers {
        _burn(from, tokenId, amount);
    }
```

The controller role has the right to modify the transferable state, which will affect whether the NFT can be circulated.

Code location:contracts/DeedGrain.sol #L44-46

```
    function reverseTransferable() public onlyControllers {
        transferable = !transferable;
    }
```

The controller role has the right to modify the maximum supply of any kind of NFT.

Code location:contracts/DeedGrain.sol #L48-50

```
    function setSupply(uint256 tokenId, uint256 supply) public onlyControllers {
        _supplies[tokenId] = supply;
    }
```

The Owner role has the right to modify the dgMinter address to any address, and dgMinter has the right to call mintDG to any address mintNFT.

Code location:contracts/Did.sol #L24-26

```
    function setDGMinterAddr(address minter) public onlyOwner {
        dgMinter = minter;
    }
```

The Owner role has the right to change the supply corresponding to any tokenID.

Code location:contracts/Did.sol #L51-55

```
    function setTokenSupply(address DGAddr, uint tokenId, uint supply) public {
        require(msg.sender == owner || msg.sender ==
deedGrainAddrToClientAddr[DeedGrain(DGAddr)], "caller are not allowed to set
supply");
        DeedGrain DG = DeedGrain(DGAddr);
        DG.setSupply(tokenId, supply);
    }
```

The Owner has the right to set didMinter to any address, which has the right to call the mint function.

Code location:contracts/Did.sol #L152-154

```
    function setDidMinterAddr(address minter) public onlyOwner {
        didMinter = minter;
    }
```

Owner has the right to call mint function to mint NFT for any address.

Code location:contracts/Did.sol #L176-179

```
    function mint(address to, string memory did) public {
        require(msg.sender == owner || msg.sender == didMinter, "caller is not
allowed to mint did");
        _mintDid(to, did);
    }
```

Solution

It is recommended to transfer the permissions of roles with excessive authorization risk to timelock management or

use multi-signature.

Status

Ignored

[N2] [Suggestion] Risk of initial operation

Category: Authority Control Vulnerability

Content

In the DidV1 contracts, by calling the initialize function to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize and there is no access control verification for the initialize functions.

Code location:contracts/Did.sol #L125-137

```
    function initialize(
        string memory _name,
        string memory _symbol,
        string memory _baseTokenURI,
        address _owner
    )
        public
        initializer
    {
        __ERC721_init(_name, _symbol);
        _setBaseURI(_baseTokenURI);
        owner = _owner;
    }
```

Solution

It is suggested that the initialize operation can be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

Status

Ignored; After communication, I learned that the initialize method will be called through the delegate after the project team deploys the proxy contract for the first time, and the proxy contract will not be initialized in the future.

[N3] [Suggestion] Missing Zero-Address check

Category: Others

Content

When the address is passed in the contract, it is not checked whether the incoming address is a Zero-Address.

Code location:contracts/Did.sol #L24-26

```
function setDGMinterAddr(address minter) public onlyOwner {
    dgMinter = minter;
}
```

Code location:contracts/Did.sol #L43-45

```
function setSigner(address _signer) public onlyOwner {
    signer = _signer;
}
```

Code location:contracts/Did.sol #L152-154

```
function setDidMinterAddr(address minter) public onlyOwner {
    didMinter = minter;
}
```

Solution

It is recommended to add a Zero-Address check.

Status

Ignored

[N4] [Suggestion] Missing scope limit

Category: Others

Content

Lack of limit on incoming amount when modifying NFT max supply.

Code location:contracts/DeedGrain.sol #L48-50

```
    function setSupply(uint256 tokenId, uint256 supply) public onlyControllers {
        _supplies[tokenId] = supply;
    }
```

Code location:contracts/Did.sol #L51-55

```
    function setTokenSupply(address DGAddr, uint tokenId, uint supply) public {
        require(msg.sender == owner || msg.sender ==
deedGrainAddrToClientAddr[DeedGrain(DGAddr)], "caller are not allowed to set
supply");
        DeedGrain DG = DeedGrain(DGAddr);
        DG.setSupply(tokenId, supply);
    }
```

Solution

It is recommended to limit the range of values passed in.

Status

Ignored; After communication, we learned that the project team does not limit the circulation in business.

[N5] [Suggestion] The authenticity of the tokenId is not verified

Category: Others

Content

The following functions do not verify that the tokenId passed in exists.

Code location:contracts/DeedGrain.sol #L40-42

```
    function burn(address from, uint256 tokenId, uint256 amount) public
onlyControllers {
        _burn(from, tokenId, amount);
    }
```

Code location:contracts/DeedGrain.sol #L48-50

```
function setSupply(uint256 tokenId, uint256 supply) public onlyControllers {
    _supplies[tokenId] = supply;
}
```

Code location:contracts/DeedGrain.sol #L57-59

```
function uri(uint256 tokenId) public override view returns(string memory) {
    return string(abi.encodePacked(_baseMetadataURI, tokenId));
}
```

Solution

It is recommended to check if the passed in tokenId exists.

Status

Ignored

[N6] [Suggestion] Missing event record

Category: Others

Content

The corresponding event record is missing when modifying sensitive parameters in the contract.

Code location:contracts/DeedGrain.sol #L44-55

```
function reverseTransferable() public onlyControllers {
    transferable = !transferable;
}

function setSupply(uint256 tokenId, uint256 supply) public onlyControllers {
    _supplies[tokenId] = supply;
}

function setBaseUri(string memory baseUri) public {
    require(msg.sender == controller);
```

```
        _baseMetadataURI = baseUri;
    }
```

Code location:contracts/Did.sol #L24-26

```
    function setDGMinterAddr(address minter) public onlyOwner {
        dgMinter = minter;
    }
```

Code location:contracts/Did.sol #L43-45

```
    function setSigner(address _signer) public onlyOwner {
        signer = _signer;
    }
```

Code location:contracts/Did.sol #L141-143

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

Code location:contracts/Did.sol #L152-154

```
    function setDidMinterAddr(address minter) public onlyOwner {
        didMinter = minter;
    }
```

Solution

It is recommended to add corresponding event records when modifying contract sensitive parameters.

Status

Ignored

[N7] [Medium] Signature replay issue

Category: Replay Vulnerability

Content

There is no anti-replay mechanism, and the authorized address and signature can repeatedly call the claimDG and issueDG function.

Code location:contracts/Did.sol #L34-39

```
    function issueDG(string memory _name, string memory _symbol, string memory _baseUri, bytes memory _evidence, bool _transferable) public {
        require(_validate(keccak256(abi.encodePacked(msg.sender)), _evidence), "invalid evidence");
        DeedGrain DG = new DeedGrain(_name, _symbol, _baseUri, _transferable);
        deedGrainAddrToClientAddr[DG] = msg.sender;
        emit IssueDG(msg.sender, address(DG));
    }
```

Code location:contracts/Did.sol #L81-85

```
    function claimDG(address DGAddr, uint tokenId, bytes memory evidence) public {
        require(_validate(keccak256(abi.encodePacked(msg.sender, DGAddr, tokenId)), evidence), "invalid evidence");
        DeedGrain DG = DeedGrain(DGAddr);
        DG.mint(msg.sender, tokenId, 1);
    }
```

Code location:contracts/Did.sol #L88-99

```
    function _validate(bytes32 message, bytes memory signature) internal view returns (bool) {
        require(signer != address(0) && signature.length == 65);

        bytes32 r;
        bytes32 s;
        uint8 v = uint8(signature[64]) + 27;
        assembly {
            r := mload(add(signature, 0x20))
            s := mload(add(signature, 0x40))
        }
```

```
        return ecrecover(message, v, r, s) == signer;
    }
```

Solution

It is recommended to add a nonce to prevent replay in the validate function.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002207040001	SlowMist Security Team	2022.06.29 - 2022.07.04	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risks and 5 suggestion vulnerabilities. And 1 medium risk was confirmed and fixed; All other findings were ignored. 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