Smart Contract
Security Audit Report

Table Of Contents

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

File name: HashKeyDID-V2.zip

SHA256: 14786eb722485a60e13c579640fda93284e93178ea400797023b11b8c16c273d

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
N7	Signature replay issue	Replay Vulnerability	Medium	Ignored

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	-
_uint2str	Internal	-	-

DeedGrainFactory			
Function Name	Visibility	Mutability	Modifiers
issueDG	Public	Can Modify State	-
issueNFT	Public	Can Modify State	-

DeedGrainNFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC721
mint	Public	Can Modify State	onlyControllers
mintBatch	External	Can Modify State	onlyControllers
setBaseUri	External	Can Modify State	-
setSupply	External	Can Modify State	onlyControllers
tokenURI	Public	-	-
name	Public	-	-
symbol	Public	-	-
_uint2str	Internal	-	-

DGIssuer			
Function Name	Visibility	Mutability	Modifiers
setDGMinterAddr	Public	Can Modify State	onlyOwner
issueDG	Public	Can Modify State	-
issueNFT	Public	Can Modify State	-
setSigner	Public	Can Modify State	onlyOwner
setDGFactory	Public	Can Modify State	onlyOwner
setTokenSupply	Public	Can Modify State	-
setTokenBaseUri	Public	Can Modify State	onlyOwner

DGIssuer			
mintDG	Public	Can Modify State	-
claimDG	Public	Can Modify State	-
setNFTSupply	Public	Can Modify State	-
setNFTBaseUri	Public	Can Modify State	onlyOwner
mintDGNFT	Public	Can Modify State	-
claimDGNFT	Public	Can Modify State	-
_validate	Internal	-	-

DidV2			
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	-
_mintDid	Internal	Can Modify State	-
verifyDIDFormat	Public	-	-
addAuth	Public	Can Modify State	-

DidV2			
removeAuth	Public	Can Modify State	-
getAuthorizedAddrs	Public	-	-
isAddrAuthorized	Public	-	-
addKYC	Public	Can Modify State	-
getKYCInfo	Public	-	-
_beforeTokenTransfer	Internal	Can Modify State	-

DidV2Storage			
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 and issuer roles have the risk of excessive authority. These two roles have the right to mint coins for any address, 1 mint at a time.

Code location:contracts/DeedGrain.sol #L38-47

```
function mint(address to, uint256 tokenId) public onlyControllers {
    if (supplies[tokenId] != 0) {
        require(
            totalSupply(tokenId) + 1 <= supplies[tokenId],
            "insufficient supply"
        );
    }
    indexMap[tokenId][to] = totalSupply(tokenId) + 1;
    _mint(to, tokenId, 1, "");
}
```

There is a risk of excessive authority for the Controller and Issuer roles, which have the power to burn any number of tokens for any user.

Code location:contracts/DeedGrain.sol #L49-55

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

The controller and issuer roles have the risk of excessive authority. These two roles have the right to change the transferable parameter, which will affect whether users can transfer their own tokens.

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

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

The controller and issuer roles have the risk of excessive authority, and these two roles have the right to modify the maximum supply of tokens to any value.

Code location:contracts/DeedGrain.sol #L61-63

```
    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 #L27-29

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

Code location:contracts/Did.sol #L151-165

```
    function mintDG(
        address DGAddr,
        uint256 tokenId,
        address[] memory addrs
    ) public {
        require(
            msg.sender == dgMinter ||
                msg.sender == deedGrainAddrToIssuer[DGAddr],
            "caller are not allowed to mint"
        );
        IDeedGrain DG = IDeedGrain(DGAddr);
        for (uint256 i = 0; i < addrs.length; i++) {
            DG.mint(addrs[i], tokenId);
        }
    }
```js
The Owner role has the right to change the factory contract address.
```js
function setDGFactory(address factory) public onlyOwner {
        dgFactory = factory;
    }
```

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

Code location:contracts/Did.sol #L123-134

```
    function setTokenSupply(
        address DGAddr,
        uint256 tokenId,
        uint256 supply
    ) public {
        require(
            msg.sender == owner || msg.sender == deedGrainAddrToIssuer[DGAddr],
            "caller are not allowed to set supply"
        );
        IDeedGrain DG = IDeedGrain(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 #L329-331

```
    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 #L353-356

```
    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);
    }
```

There is a risk of excessive authority for the roles of Controller and issuer, which have the right to mint NFT for any address.

Code location: DeedGrainNFT.sol #L39-69

```
    function mint(address to, uint256 sid)
        public
        onlyControllers
        returns (uint256)
    {
        if (supply != 0) {
            require(totalSupply + 1 <= supply, "insufficient supply");
        }
        totalSupply++;
        uint256 tokenId = totalSupply;

        _mint(to, tokenId);
        sids[tokenId] = sid;
        return tokenId;
    }

    function mintBatch(
        address[] calldata addrs,
        uint256[] calldata seriesIds
    ) external onlyControllers {
        require(addrs.length == seriesIds.length, "accounts length not equal to
sidArr length");
        if (supply != 0) {
            require(totalSupply + addrs.length <= supply, "insufficient supply");
        }
        for (uint256 i = 0; i < addrs.length; i++) {
            totalSupply++;
            uint256 tokenId = totalSupply;
            _mint(addrs[i], tokenId);
            sids[tokenId] = seriesIds[i];
        }
    }
```

The controller and issuer roles have the risk of excessive authority, and these two roles have the right to change the maximum supply of tokens to any value.

Code location: DeedGrainNFT.sol #L82-84

```
    function setSupply(uint256 supply_) external onlyControllers {
        supply = supply_;
    }
```

The owner role has the risk of excessive authority. This role has the right to modify the maximum supply of tokens to any value.

Code location:contracts/Did.sol #L193-200

```
function setNFTSupply(address NFTAddr, uint256 supply) public {
    require(
        msg.sender == owner || msg.sender == deedGrainAddrToIssuer[NFTAddr],
        "caller are not allowed to set supply"
    );
    IDeedGrainNFT NFT = IDeedGrainNFT(NFTAddr);
    NFT.setSupply(supply);
}
```

There is a risk of excessive authority with the dgMinter role, which has the power to mint NFT to arbitrary addresses.

Code location:contracts/Did.sol #L217-231

```
function mintDGNFT(
    address NFTAddr,
    uint256 sid,
    address[] memory addrs
) public {
    require(
        msg.sender == dgMinter ||
            msg.sender == deedGrainAddrToIssuer[NFTAddr],
        "caller are not allowed to mint"
    );
    IDeedGrainNFT NFT = IDeedGrainNFT(NFTAddr);
    for (uint256 i = 0; i < addrs.length; i++) {
        NFT.mint(addrs[i], sid);
    }
}
```

Solution

1. It is recommended to transfer the authority of roles with excessive authorization risk to the governance contract, at least using multi-signature wallets.

2. It is recommended to attach an airdrop cap when creating a contract to limit the number of airdrops available.
3. It is recommended to limit the maximum value that can be set for the maximum supply of tokens.

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 #L300-314

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

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 #L27-29

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

Code location:contracts/Did.sol #L109-117

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

/// @dev Set dgFactory address
/// @param factory DeedGrainFactory contract address
function setDGFactory(address factory) public onlyOwner {
    dgFactory = factory;
}
```

Code location:contracts/Did.sol #329-331

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

Solution

It is recommended to check that the incoming address is not a zero address.

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 #L61-63

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

Code location:contracts/Did.sol #L123-134

```
function setTokenSupply(
    address DGAddr,
    uint256 tokenId,
    uint256 supply
) public {
    require(
        msg.sender == owner || msg.sender == deedGrainAddrToIssuer[DGAddr],
        "caller are not allowed to set supply"
    );
    IDeedGrain DG = IDeedGrain(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 #L49-55

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

Code location:contracts/DeedGrain.sol #L61-63

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

Code location:contracts/DeedGrain.sol #L70-72

```
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 #L57-68

```
    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 #L27-29

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

Code location:contracts/Did.sol #L109-145

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

    /// @dev Set dgFactory address
    /// @param factory DeedGrainFactory contract address
```

```
function setDGFactory(address factory) public onlyOwner {
    dgFactory = factory;
}

/// @dev Only issuer can set every kind of token's supply
/// @param DGAddr DG contract address
/// @param tokenId TokenId
/// @param supply Token's supply number
function setTokenSupply(
    address DGAddr,
    uint256 tokenId,
    uint256 supply
) public {
    require(
        msg.sender == owner || msg.sender == deedGrainAddrToIssuer[DGAddr],
        "caller are not allowed to set supply"
    );
    IDeedGrain DG = IDeedGrain(DGAddr);
    DG.setSupply(tokenId, supply);
}

/// @dev Only issuer can set token's baseuri
/// @param DGAddr DG contract address
/// @param baseUri All of the token's baseuri
function setTokenBaseUri(address DGAddr, string memory baseUri)
    public
    onlyOwner
{
    IDeedGrain DG = IDeedGrain(DGAddr);
    DG.setBaseUri(baseUri);
}
```

Code location:contracts/Did.sol #L318-320

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

Code location:contracts/Did.sol #L329-331

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

Code location: DeedGrainNFT.sol #L74-84

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

/**
 * Sets totalSupply for NFT.
 */
function setSupply(uint256 supply_) external onlyControllers {
    supply = supply_;
}
```

Code location:contracts/Did.sol #L193-211

```
function setNFTSupply(address NFTAddr, uint256 supply) public {
    require(
        msg.sender == owner || msg.sender == deedGrainAddrToIssuer[NFTAddr],
        "caller are not allowed to set supply"
    );
    IDeedGrainNFT NFT = IDeedGrainNFT(NFTAddr);
    NFT.setSupply(supply);
}

/// @dev Only issuer can set NFT's baseuri
/// @param NFTAddr DG NFT contract address
/// @param baseUri All of the NFT's baseuri
function setNFTBaseUri(address NFTAddr, string memory baseUri)
    public
    onlyOwner
{
    IDeedGrainNFT NFT = IDeedGrainNFT(NFTAddr);
    NFT.setBaseUri(baseUri);
}
```

Solution

It is recommended to add event records when modifying important parameters of the contract, which is conducive to the supervision of users and the community.

Status

Ignored

[N7] [Medium] Signature replay issue

Category: Replay Vulnerability

Content

In issueDG and issueNFT, single evidence can be reused, which will result in an unlimited issuance of DG and NFT by evidence holders.

Code location: DGIssuer.sol #L37-105

```
    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,
                signer
            ),
            "invalid evidence"
        );
        bool success;
        bytes memory data;
        (success, data) = dgFactory.delegatecall(
            abi.encodeWithSignature(
                "issueDG(string,string,string,bool)",
                _name,
                _symbol,
                _baseUri,
```

```
            _transferable
        )
    );
    require(success, "issueDG failed");
    address DGAddr = abi.decode(data, (address));
    deedGrainAddrToIssur[DGAddr] = msg.sender;
    emit IssueDG(msg.sender, DGAddr);
}

/// @dev Issue DG NFT
/// @param _name ERC721 NFT name
/// @param _symbol ERC721 NFT symbol
/// @param _baseUri ERC721 NFT baseUri
/// @param _evidence Signature by HashKeyDID
/// @param _supply DG NFT supply
function issueNFT(
    string memory _name,
    string memory _symbol,
    string memory _baseUri,
    bytes memory _evidence,
    uint256 _supply
) public {
    require(
        _validate(
            keccak256(abi.encodePacked(msg.sender)),
            _evidence,
            signer
        ),
        "invalid evidence"
    );
    bool success;
    bytes memory data;
    (success, data) = dgFactory.delegatecall(
        abi.encodeWithSignature(
            "issueNFT(string,string,string,uint256)",
            _name,
            _symbol,
            _baseUri,
            _supply
        )
    );
    require(success, "issueDGNFT failed");
    address DGNFTAddr = abi.decode(data, (address));
    deedGrainAddrToIssur[DGNFTAddr] = msg.sender;
```

```
        emit IssueNFT(msg.sender, DGNFTAddr);
    }
```

Solution

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

Status

Ignored; After communication, we learned that signatures are allowed to be reused in normal business design. We recommend adding a state to the signature that needs to be determined when the issuer uses the signature to see if the signature is available so that the project team can respond in time to withdraw the permission in extreme cases like the issuer's private key being stolen. Of course, replacing the signer can also avoid the above situation, but this will affect other average issuers, so adding a switch to the signature is undoubtedly a better choice.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002207110003	SlowMist Security Team	2022.06.29 - 2022.07.11	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool for auditing the project, during the audit work we found 2 medium risks and 5 suggestion vulnerabilities. All the findings have been 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