Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.08.17, the SlowMist security team received the UMB team's security audit application for Spear, 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

This is a cross-chain bridge contract that includes ERC721 Token and ERC20 Token parts.

Audit version file information

Initial audit files:

https://github.com/umbrella-network/spear-contracts

commit: 1f3d7f48db2869f9ee3cc5e58c37a5f942ff9e47

Final audit files:

https://github.com/umbrella-network/spear-contracts

branch: addressed-audit-issues

commit: 89357bad45acc7ce54f49e1038af26415c7d4de7

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Illegal signature verification	Design Logic Audit	High	Fixed
N2	The contract owner of token can mint unlimited tokens	Authority Control Vulnerability	Low	Ignored
N3	Unfinished code	Others	Suggestion	Confirming
N4	Unauthenticated function	Authority Control Vulnerability	Low	Ignored
N5	The called target contract and the called target function are not checked	Design Logic Audit	Low	Confirming
N6	Redundant check	Design Logic Audit	Suggestion	Fixed
N7	Single signature risk	Design Logic Audit	Medium	Confirming
N8	Transaction malleability attack risk	Design Logic Audit	Low	Ignored
N9	Unused modifier	Others	Suggestion	Confirming

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:

SampleERC20			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

ERC721StringFaucet			
Function Name	Visibility	Mutability	Modifiers
releaseToken	Public	Can Modify State	onlyOwner
mint	External	Can Modify State	-
getAllTokens	Public	-	-
parseTokenId	Public	-	-

ERC721Mintable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Can Modify State	onlyOwner
mintTo	External	Can Modify State	onlyOwner
transfer	External	Can Modify State	-

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

HomeGate			
withdraw	External	Can Modify State	-
withdrawAndCall	External	Can Modify State	-

OperatorHub			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
addOperator	Public	Can Modify State	onlyOwner
removeOperator	Public	Can Modify State	onlyOwner
updateLocation	Public	Can Modify State	onlyOwner
setRequiredOperators	Public	Can Modify State	onlyOwner
isOperator	Public	-	-
operatorCount	Public	-	-
operatorAddresses	Public	-	-
operatorLocations	Public	-	-
stringToBytes32	Internal	-	-
checkSignatures	Public	-	-
prefixed	Internal	-	-

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

ForeignGate			
mint	External	Can Modify State	-
mintAndCall	External	Can Modify State	-
transferTokenOwnership	External	Can Modify State	onlyOwner

ERC20Mintable			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Can Modify State	onlyOwner
mintTo	External	Can Modify State	onlyOwner

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

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

SampleERC721			
Function Name	Visibility	Mutability	Modifiers

SampleERC721			
<Constructor>	Public	Can Modify State	-

4.3 Vulnerability Summary

[N1] [High] Illegal signature verification

Category: Design Logic Audit

Content

One operator can do multiple signatures to generate multiple sets of v, r, s, and the code doesn't verify that the operator has had signature verification. So one operator can bypass signature verification by generating multiple sets of v, r, s.

Code location: `evm/contracts/OperatorHub.sol`

```
function checkSignatures(
    bytes32 hash,
    uint256 length,
    uint8[] memory v,
    bytes32[] memory r,
    bytes32[] memory s
  ) public view returns(uint8) {
    uint8 approvals = 0;

    for (uint i = 0; i < length; ++i) {
      address operator = ecrecover(hash, v[i], r[i], s[i]);
      require(isOperator(operator), "should be an operator");
      approvals ++;
    }

    return approvals;
  }
```

So the attack scenario:

1.The operator signs a nonexistent record(transactionHash, tokenContract, recipient, value) to generate multiple sets of v, r, s.

2.The operator calls functions `mint` , `mintAndCall` , `withdraw` , `withdrawAndCall` in `HomeGate.sol` and `ForeignGate.sol` , parameters for which are the nonexistent record and the generated multiple sets of v, r, s.

3. The operator can bypass signature verification, and can `mint` and `withdraw` from the bridge.

Solution

Add conditional code: A verified operator can't be checked repeatedly.

Status

Fixed

[N2] [Low] The contract owner of token can mint unlimited tokens

Category: Authority Control Vulnerability

Content

The ERC20 & ERC721 token contract owner can mint unlimited tokens via the `mint` function and the `mintTo` fuction.

Code location: `evm/contracts/ERC20Mintable.sol`

```
abstract contract ERC20Mintable is ERC20Burnable, Ownable {
  constructor(string memory _name, string memory _symbol)
    ERC20(_name, _symbol) {
  }

  function mint(uint256 amount) onlyOwner external {
    _mint(msg.sender, amount);
  }

  function mintTo(address to, uint256 amount) onlyOwner external {
    _mint(to, amount);
  }
}
```

Code location: `evm/contracts/ERC721Mintable.sol`

```
abstract contract ERC721Mintable is ERC721Burnable, Ownable {
  constructor(string memory _name, string memory _symbol)
    ERC721(_name, _symbol) {
  }

  function mint(uint256 tokenId) onlyOwner external {
    _mint(msg.sender, tokenId);
  }

  function mintTo(address to, uint256 tokenId) onlyOwner external {
    _mint(to, tokenId);
  }

  // TODO: not all third-party contracts will be compatible with this method
  function transfer(address to, uint256 tokenId) external {
    _transfer(msg.sender, to, tokenId);
  }
}
```

Solution

Suggest setting up a limited amount of token management plan.

Status

Ignored; This is not a token that will be released with bridge, this just is a test token.

[N3] [Suggestion] Unfinished code

Category: Others

Content

The `TODO` code is found in the audit code.

Code location: `evm/contracts/ERC721Mintable.sol`

```
abstract contract ERC721Mintable is ERC721Burnable, Ownable {
  ...

  // TODO: not all third-party contracts will be compatible with this method
  function transfer(address to, uint256 tokenId) external {
    _transfer(msg.sender, to, tokenId);
    }
}
```

Solution

Complete the `TODO` code.

Status

Confirming

[N4] [Low] Unauthenticated function

Category: Authority Control Vulnerability

Content

The contracts in the sample directory and the `ERC721StringFaucet.sol` contract contain some unauthenticated functions. The visibility of these unauthenticated functions is either public or external.

Code location: `evm/contracts/sample/SampleForeignContract.sol`

```
function submit(uint256 _root) public {
    require(relayToken.ownerOf(_root) != address(0x0), "A relay token should exist");
    require(submissions[_root] == false, "This submission has already been made");

    submissions[_root] = true;
  }
```

Code location: `evm/contracts/sample/SampleHomeContract.sol`

```
function submit(uint256 _root) public {
    require(relayToken.ownerOf(_root) != address(0x0), "A relay token should exist");
    require(submissions[_root] == false, "This submission has already been made");

    submissions[_root] = true;

    // mint an NFT token
    relayToken.mint(_root);

    // transfer an NFT token through the bridge
    relayToken.transferFrom(address(this), address(homeGate), _root);
  }
```

Code location: `evm/contracts/ERC721StringFaucet.sol`

```
function mint(ERC721Mintable token, string memory text) external {
    nonce += 1;

    uint256 valueBytes;
    bytes memory stringBytes = bytes(text);

    uint256 length = stringBytes.length;

    require(length <= 27, "text is too long");

    if (stringBytes.length == 0) {
      valueBytes = 0x0;
    } else {
      assembly {
        valueBytes := mload(add(text, 32))
      }
    }

    // 0x[text bytes][1 byte text length][4 byte nonce], e.g. 0x3132330300000001 =
"210"
    uint256 tokenId = uint256(nonce) | (length << 32) | (uint256(valueBytes) >> (216
- length * 8));

    // mint an NFT token
    token.mint(tokenId);

    // transfer an NFT token
    token.transferFrom(address(this), address(msg.sender), tokenId);
  }
```

Solution

Confirm these unauthenticated functions are test contracts.

Status

Ignored; They are just test contracts.

[N5] [Low] The called target contract and the called target function are not checked

Category: Design Logic Audit

Content

The `mintAndCall` function exists in the `ForeignGate` contract, and the `withdrawAndCall` function exists in the `HomeGate` contract. These functions don't check the target contract address and the target function before the `call` operation.

Code location: `evm/contracts/HomeGate.sol`

```
function withdrawAndCall(
    bytes32 transactionHash,
    address tokenContract,
    address recipient,
    uint256 value,
    uint8[] memory v,
    bytes32[] memory r,
    bytes32[] memory s,
    address target,
    bytes memory _calldata
  ) external {
    this.withdraw(transactionHash, tokenContract, recipient, value, v, r, s);

    assembly {
      let succeeded := call(gas(), target, 0, add(_calldata, 0x20), mload(_calldata),
0, 0)

      switch iszero(succeeded)
      case 1 {
      // throw if delegatecall failed
        let size := returndatasize()
        returndatacopy(0x00, 0x00, size)
        revert(0x00, size)
      }
    }
  }
```

Code location: `evm/contracts/ForeignGate.sol`

```
function mintAndCall(
    bytes32 transactionHash,
    address tokenContract,
    address recipient,
    uint256 value,
    uint8[] memory v,
    bytes32[] memory r,
    bytes32[] memory s,
    address target,
    bytes memory calldata_
  ) external {
    this.mint(transactionHash, tokenContract, recipient, value, v, r, s);

    assembly {
      let succeeded := call(gas(), target, 0, add(calldata_, 0x20), mload(calldata_),
0, 0)

      switch iszero(succeeded)
        case 1 {
          // throw if delegatecall failed
          let size := returndatasize()
          returndatacopy(0x00, 0x00, size)
          revert(0x00, size)
        }
    }
  }
```

Solution

Check the target contract address and the `calldata` to ensure the expected `call` operation is executed correctly.

Status

Confirming

[N6] [Suggestion] Redundant check

Category: Design Logic Audit

Content

In the `OperatorHub` contract, it is unnecessary to check if the address in the parameter `initialOperators` (an address list) is an operator.

Code location: `evm/contracts/OperatorHub.sol`

```
constructor(uint8 requiredOperators_, address[] memory initialOperators) {
    require(requiredOperators_ != 0, "should provide the number of required
operators");
    require(initialOperators.length >= requiredOperators_, "should provide more
operators");

    for (uint i = 0; i < initialOperators.length; i++) {
      require(!isOperator(initialOperators[i]) && initialOperators[i] != address(0));
      addOperator(initialOperators[i]);
    }

    setRequiredOperators(requiredOperators_);
  }
```

Solution

Remove conditional code `!isOperator(initialOperators[i])`.

Status

Fixed

[N7] [Medium] Single signature risk

Category: Design Logic Audit

Content

If `requiredOperators` is set as 1, the procedure of verifying the signature has a single signature risk. And the function `setRequiredOperators` doesn't require `requiredOperators` > 1.

Code location: `evm/contracts/OperatorHub.sol`

```
function setRequiredOperators(uint8 requiredOperators_) public onlyOwner {
    require(operatorList.length >= requiredOperators_, "cannot be more than the
number of added operators");
    requiredOperators = requiredOperators_;
  }
```

Solution

The function `setRequiredOperators` require `requiredOperators` > 1.

Status

Confirming

[N8] [Low] Transaction malleability attack risk

Category: Design Logic Audit

Content

The process of verifying signatures via `ecrecover` occurs through the function `checkSignatures`. The process of verifying signatures has a transaction malleability attack risk.

Code location: `evm/contracts/OperatorHub.sol`

```
function checkSignatures(
    bytes32 hash,
    uint256 length,
    uint8[] memory v,
    bytes32[] memory r,
    bytes32[] memory s
  ) public view returns(uint8) {
    uint8 approvals = 0;

    for (uint i = 0; i < length; ++i) {
      address operator = ecrecover(hash, v[i], r[i], s[i]);
      require(isOperator(operator), "should be an operator");
      approvals ++;
    }

    return approvals;
  }
```

Solution

Use the ECDSA library of openzeppelin contracts.

Status

Ignored; Transaction malleability attack risk is not included in this scenario. The transaction data from the original chain is verified with the `withdraw` function.

```
bytes32 hash = prefixed(keccak256(abi.encodePacked(transactionHash, tokenContract,
recipient, value)));
require(usedHashes[hash] == false, "already withdrawn");
usedHashes[hash] = true;
```

[N9] [Suggestion] Unused modifier

Category: Others

Content

`onlyOperator` is an unused modifier.

Code location: `evm/contracts/OperatorHub.sol`

```
modifier onlyOperator(address operator) {
    assert(operators[operator] == true);
    _;
  }
```

Solution

If a modifier is not used, remove it.

Status

Confirming

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X02108210001	SlowMist Security Team	2021.08.17 - 2021.08.20	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and the SlowMist team's analysis tool to audit the project. During the audit work, we found 1 critical risk, 1 medium risk, 4 low risk, 3 suggestion vulnerabilities. 2 findings were fixed, 3 findings were ignored, and 4 findings were confirming. 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.

SLOWMIST

Official Website
www.slowmist.com

✉
E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist