Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.12.06, the SlowMist security team received the Flurry Finance team's security audit application for Flurry Bridge, 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:

- 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 Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit Version:

https://github.com/FlurryFinance/flurry-bridge/

commit: 4b58c809014e2d6a851962724e65abd6e4055da8

Fixed Version

https://github.com/FlurryFinance/flurry-bridge/

commit: f37de2076995ad14c0144ecf98e8ba5ceae8f91b

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	DoS issue	Denial of Service Vulnerability	Low	Confirmed
N2	Safety Reminders	Others	Suggestion	Confirmed
N3	Limit of value range	Others	Suggestion	Fixed
N4	Useless code	Others	Suggestion	Confirmed

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:

Bridge			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
setFederation	External	Can Modify State	onlyOwner
isTransferProcessed	External	-	-
acceptTransfer	External	Can Modify State	onlyFederation

Bridge			
bridgeTokenAt	External	Can Modify State	whenNotPaused nonReentrant
bridgeToken	External	Can Modify State	whenNotPaused nonReentrant
_bridgeToken	Internal	Can Modify State	-
_crossTokens	Internal	Can Modify State	-
_calculateFee	Internal	-	-

Federation			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
processed	External	-	-
setBridge	External	Can Modify State	onlyOwner
_setBridge	Internal	Can Modify State	-
addMember	External	Can Modify State	onlyOwner
removeMember	External	Can Modify State	onlyOwner
voteTransfer	External	Can Modify State	onlyMember
hasVotedTransfer	External	-	-
isTransferProcessed	External	-	-
getVoteCount	Public	-	-
setRequired	External	Can Modify State	onlyOwner
_setRequired	Internal	Can Modify State	-

Registry			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	initializer
registerToken	External	Can Modify State	onlyOwner
unregisterToken	External	Can Modify State	onlyOwner
setFee	External	Can Modify State	onlyOwner

4.3 Vulnerability Summary

[N1] [Low] DoS issue

Category: Denial of Service Vulnerability

Content

The getVoteCount function uses a for loop to count members' votes. When the number of members is large, it will cause DoS due to the increased number of for loops.

- bridge/contracts/contracts/Federation.sol#L242-L249

```
function getVoteCount(bytes32 processId) public view override returns(uint) {
    uint count = 0;
    for (uint i = 0; i < members.length; i++) {
        if (votes[processId][members[i]])
            count += 1;
    }
    return count;
}
```

Solution

It is recommended to use global variables like "VoteCount" to accumulate votes when members are voting.

Status

Confirmed; The project team response: They acknowledged the risk, and that the members can only be added by theirs.

[N2] [Suggestion] Safety Reminders

Category: Others

Content

To capture events in the cross-chain bridge, the implementation of subscribing to the events of the specified contract should be adopted to avoid the attacks of fake contract events.

Solution

It is recommended to determine the attribution contract of the event when obtaining the event.

Status

Confirmed; The project team response: events captured from frontend and validators nodes are filtered by the contract address.

[N3] [Suggestion] Limit of value range

Category: Others

Content

Owner can set fee arbitrarily, and there is no restriction on the value range. and the fee variable is not used in the contract code.

- bridge/contracts/contracts/Registry.sol#L87-L91

```
    function setFee(address localaddr_, uint256 fee_) external override onlyOwner {
        require(fee_ > 0, "Registry: Fee Should be> 0");
        fee[localaddr_] = fee_;
        emit FeeChanged(localaddr_, fee_);
    }
```

Solution

It is recommended to add a restriction on the fee range.

Status

Fixed; The issue has been fixed in commit: f37de2076995ad14c0144ecf98e8ba5ceae8f91b

[N4] [Suggestion] Useless code

Category: Others

Content

There are a lot of comment codes in the contract. It is necessary to confirm whether the comment codes are redundant codes.

- bridge/contracts/contracts/Registry.sol#L43-L85

```
// function registerCall(
    //     uint256 alienChainId_,
    //     address alienChainContractAddr_,
    //     address localChainContractAddr_,
    //     bytes4 callSig_
    // ) external onlyOwner {
    //     bytes32 callRegistryID = Utils.getCallRegistryId(
    //         alienChainId_,
    //         alienChainContractAddr_,
    //         localChainContractAddr_,
    //         callSig_
    //     );
    //     require(!callRegistry[callRegistryID], "Registry: Call already exists in
callRegistry");
    //     callRegistry[callRegistryID] = true;
    //     emit CallRegistered(
    //         alienChainId_,
    //         alienChainContractAddr_,
    //         localChainContractAddr_,
    //         callSig_
    //     );
    // }

    // function unregisterCall(
    //     uint256 alienChainId_,
    //     address alienChainContractAddr_,
    //     address localChainContractAddr_,
    //     bytes4 callSig_
    // ) external onlyOwner {
    //     bytes32 callRegistryID = Utils.getCallRegistryId(
    //         alienChainId_,
    //         alienChainContractAddr_,
    //         localChainContractAddr_,
    //         callSig_
    //     );
    //     require(callRegistry[callRegistryID], "Registry: Call not registered");
    //     delete callRegistry[callRegistryID];
    //     emit CallUnregistered(
    //         alienChainId_,
    //         alienChainContractAddr_,
    //         localChainContractAddr_,
    //         callSig_
    //     );
    // }
```

- bridge/contracts/contracts/Federation.sol#L159-L240

```
// function voteCall(
    //     uint256 srcChainID_,
    //     address srcChainContractAddress_,
    //     address dstChainContractAddress_,
    //     bytes32 transactionHash_,
    //     uint32 logIndex_,
    //     bytes calldata payload
    // ) external override onlyMember {
    //     if (bridge.isCallProcessed(
    //         srcChainID_,
    //         srcChainContractAddress_,
    //         dstChainContractAddress_,
    //         transactionHash_,
    //         logIndex_,
    //         payload
    //     )) {
    //        return;
    //     }
    //     bytes32 callId = Utils.getCallId(
//          srcChainID_,
//          srcChainContractAddress_,
//          dstChainContractAddress_,
//          transactionHash_,
//          logIndex_,
//          payload
//      );
//      if (votes[callId][_msgSender()])
//          return;

//      votes[callId][_msgSender()] = true;
//      emit VotedCall(
//          srcChainID_,
//          srcChainContractAddress_,
//          dstChainContractAddress_,
//          transactionHash_,
//          logIndex_,
//          _msgSender(),
//          callId,
//          payload
//      );
//      uint voteCount = getVoteCount(callId);
//      if ((voteCount >= required) && (voteCount >= members.length / 2 + 1)) {
//          bridge.acceptCall(
//              srcChainID_,
//              srcChainContractAddress_,
//              dstChainContractAddress_,
//              transactionHash_,
//              logIndex_,
//              payload
//          );
//          emit ExecutedCall(callId);
//      }

// }
// function hasVotedCall(
//      uint256 srcChainID_,
//      address srcChainContractAddress_,
//      address dstChainContractAddress_,
//      bytes32 transactionHash_,
//      uint32 logIndex_,
//      bytes calldata payload
// ) external view override returns(bool) {
//      bytes32 callId = Utils.getCallId(
    //          srcChainID_,
    //          srcChainContractAddress_,
    //          dstChainContractAddress_,
    //          transactionHash_,
    //          logIndex_,
    //          payload
    //      );
    //      return votes[callId][_msgSender()];
    // }
    // function isCallProcessed(
    //      uint256 srcChainID_,
    //      address srcChainContractAddress_,
    //      address dstChainContractAddress_,
    //      bytes32 transactionHash_,
    //      uint32 logIndex_,
    //      bytes calldata payload
    // ) external view override returns(bool) {
    //      return bridge.isCallProcessed(srcChainID_, srcChainContractAddress_,
dstChainContractAddress_, transactionHash_, logIndex_, payload);
    // }
```

- bridge/contracts/contracts/Bridge.sol#L114-L154

```
// function acceptCall(
    //      uint256 srcChainID_,
    //      address srcChainTokenAddress_,
    //      address dstChainTokenAddress_,
    //      bytes32 transactionHash_,
    //      uint32 logIndex_,
    //      bytes calldata payload
    // ) external override onlyFederation nonReentrant {
    //      require(dstChainTokenAddress_ != address(0), "Bridge: destination chain
token address is null");
    //      require(srcChainTokenAddress_ != address(0), "Bridge: src chain token
address is null");
    //      require(transactionHash_ != bytes32(0), "Bridge: Transaction is null");
    //      require(srcChainTokenAddress_ != address(0), "src token address is null");
    //      bytes4 sig =
    //          payload[0] |
    //          (bytes4(payload[1]) >> 8) |
    //          (bytes4(payload[2]) >> 16) |
    //          (bytes4(payload[3]) >> 24);
    //     bytes32 callRegistryID = Utils.getCallRegistryId(
    //         srcChainID_,
    //         srcChainTokenAddress_,
    //         dstChainTokenAddress_,
    //         sig
    //     );
    //     require(tokenRegistry.callRegistry(callRegistryID), "Call Not
Registered");
    //     bytes32 callId = Utils.getCallId(
    //         srcChainID_,
    //         srcChainTokenAddress_,
    //         dstChainTokenAddress_,
    //         transactionHash_,
    //         logIndex_,
    //         payload
    //     );

    //     require(processed[callId] == 0, "Bridge: Already processed");
    //     processed[callId] = block.number;

    //     // call the function
    //     (bool success, ) = dstChainTokenAddress_.call(payload);
    //     require(success, "call fail");
    // }
```

- bridge/contracts/libraries/Utils.sol#L6-L18

```
    // function getCallRegistryId(
    //     uint256 alienChainId_,
    //     address alienChainContractAddr_,
    //     address localChainContractAddr_,
    //     bytes4 callSig_
    // ) internal pure returns(bytes32) {
    //     return keccak256(abi.encodePacked(
    //         alienChainId_,
    //         alienChainContractAddr_,
    //         localChainContractAddr_,
    //         callSig_
    //     ));
    // }
```

- bridge/contracts/libraries/Utils.sol#L50-L67

```
// function getCallId(
//     uint256 srcChainID_,
  //     address srcChainTokenAddress_,
  //     address dstChainTokenAddress_,
  //     bytes32 transactionHash_,
  //     uint32 logIndex_,
  //     bytes calldata payload
  // ) internal pure returns (bytes32) {
  //     return keccak256(abi.encodePacked(
  //         "Call",
  //         srcChainID_,
  //         srcChainTokenAddress_,
  //         dstChainTokenAddress_,
  //         transactionHash_,
  //         logIndex_,
  //         payload
  //     ));
  // }
```

Solution

Need to confirm with the developer, if the code is useless, it is recommended to delete them.

Status

Confirmed; The project team response: the commented codes are deliberately left there, they are for later use.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002112140002	SlowMist Security Team	2021.12.06 - 2021.12.14	Low 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 1 low risk, 3 suggestion vulnerabilities. And 1 low risk, 2 suggestion vulnerabilities were confirmed;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.