Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.11.07, the SlowMist security team received the Prosper team's security audit application for Prosper 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.
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
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

Serial Number	Audit Class	Audit Subclass
7	Security Design 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

This audit primarily focuses on the token cross-chain bridge of the Prosper protocol.

There are two contracts of interest. A TokenRelayer and a MessageRelayer. These work together, and the TokenRelayer handles all tokens, and the MessageRelayer all interaction with LZ endpoint/infra. This is done to separate token handling logic from message sending logic, so that the token handling logic may be upgradeable.

The intention is to have the tokens locked in the TokenRelayer in the source chain, and have the MessageRelayer relay the cross chain message via LZ infrastructure to the destination chain's TokenRelayer, where tokens will be released. A TokenRelayer and MessageRelayer pair is to be deployed on both ETH and BSC.

The TokenRelayers will be set up and injected with initial liquidity to allow for cross-chain transfers.

All management of the MessageRelayer is intended to happen via the TokenRelayer, however it is possible to transfer

rights to a different address for management. Additionally, the MessageRelayer may be re-deployed, and by appropriate procedures the TokenRelayer may be updated to use the new MessageRelayer. This is meant only as a last resort.

The contract functionality hinges on the correct management of liquidity on both chains so that cross-chain swapping may operate as designed.

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Missing update of the latestDstLiquidity value when cross-chain	Design Logic Audit	High	Fixed
N2	Incorrect check of the fee.nativeFee	Design Logic Audit	Low	Fixed
N3	Missing scope limit	Others	Suggestion	Fixed
N4	Token compatibility issues	Design Logic Audit	Suggestion	Acknowledged
N5	Missing non-zero address check	Others	Suggestion	Fixed
N6	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/hyplabs/animoca-bridge

commit: 2d9333b84f6ca2985cadcecde3166f29a1488e29

Fixed Version:

https://github.com/hyplabs/animoca-bridge

commit: 1b920bd950cacb48a9c9c7f240914622587f2508

Audit scope:

- ./contracts/MessageRelayer.sol
- ./contracts/TokenRelayer.sol
- ./contracts/interfaces/*.sol
- ./contracts/lib/*.sol
- ./contracts/storage/*.sol

The main network address of the contract is as follows:

Ethereum:

- TokenRelayer Proxy:

  https://etherscan.io/address/0xAC95f8b9f4B848E4A207b894e259cD4C82681179
- TokenRelayer Implementation:

  https://etherscan.io/address/0x5648003d9dca549Ff5Db0C76B3F1B0b58355f479
- MessageRelayer:

  https://etherscan.io/address/0x1412f2B39050cd05bc0B2f08acb5Cb8e41288fda

BSC:

- TokenRelayer Proxy:

  https://bscscan.com/address/0x717bCA1783361E7b12367d903e23cb2FEf1F1dFc
- TokenRelayer Implementation:

  https://bscscan.com/address/0x968Bc62856Ab19Bff3bcE143782225BC81487b10
- MessageRelayer:

  https://bscscan.com/address/0x8e1d9f15443eaF912412Ba7027564051E3d12995

4.2 Visibility Description

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

TokenRelayer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_authorizeUpgrade	Internal	Can Modify State	onlyRole
__TokenRelayer_init	External	Can Modify State	initializer
addLiquidity	External	Payable	onlyRole whenNotPaused
bridgeTokens	External	Payable	whenNotPaused
burnMessage	External	Can Modify State	onlyRole whenNotPaused
clearMessage	External	Can Modify State	onlyRole
unsafeCredit	External	Can Modify State	onlyRole
pause	External	Can Modify State	onlyRole whenNotPaused
receiveMessage	External	Can Modify State	onlyRole
removeLiquidity	External	Payable	onlyRole whenNotPaused
retireMessageRelayer	External	Can Modify State	onlyRole whenPaused
setDelegate	External	Can Modify State	onlyRole
setExpectedAddLiquidityGas	External	Can Modify State	onlyRole
setExpectedBridgeTokenGas	External	Can Modify State	onlyRole
setExpectedRemoveLiquidityGas	External	Can Modify State	onlyRole
setFeeBufferBP	External	Can Modify State	onlyRole
setMessageRelayer	External	Can Modify State	onlyRole
setPeer	External	Can Modify State	onlyRole
skipMessage	External	Can Modify State	onlyRole

TokenRelayer			
transferOwnership	External	Can Modify State	onlyRole
unpause	External	Can Modify State	onlyRole whenPaused
getAddLiquidityQuote	External	-	-
getBridgeTokenQuote	External	-	-
getExpectedAddLiquidityGas	External	-	-
getExpectedBridgeTokenGas	External	-	-
getExpectedRemoveLiquidityGas	External	-	-
getFeeBufferBP	External	-	-
getLatestDstLiquidity	External	-	-
getMessageRelayer	External	-	-
getNonce	External	-	-
getRemoveLiquidityQuote	External	-	-
_creditUser	Internal	Can Modify State	-
_buildOptions	Internal	-	-
_composeActionData	Internal	-	-
_addressToBytes32	Internal	-	-

MessageRelayer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	OApp Ownable
retire	External	Can Modify State	onlyOwner
sendMessage	External	Payable	onlyTokenRelayer onlyWhenActive
getQuote	External	-	-

MessageRelayer			
setPeer	Public	Can Modify State	onlyOwner onlyWhenActive
transferOwnership	Public	Can Modify State	onlyTokenRelayer onlyWhenActive
nextNonce	Public	-	-
_lzReceive	Internal	Can Modify State	onlyWhenActive
_relayMessage	Internal	Can Modify State	-

4.3 Vulnerability Summary

[N1] [High] Missing update of the latestDstLiquidity value when cross-chain

Category: Design Logic Audit

Content

In the TokenRelayer contract, any user can perform cross-chain operations by calling the bridgeTokens function, which requires the user to transfer tokens to the TokenRelayer contract on the current chain, after which a cross-chain message will be sent. Upon receiving the message on the target chain, tokens on the target chain will be transferred to the user, and the latestDstLiquidity value of the source chain will be updated in the TokenRelayer contract on the target chain. However, after the cross-chain operation, the latestDstLiquidity value for the target chain is not updated in the TokenRelayer contract on the source chain, which could lead to a situation where, during the next cross-chain operation, the bridgeTokens function can be successfully called on the source chain, but due to insufficient balance of the target chain, the tokens cannot be transferred to the user on the target chain, causing the tokens to be locked.

For example:

1.Suppose there are 100 tokens in the contract on both the ETH chain and the BSC chain, with both sides recording their latestDstLiquidity as 100.

2.At this point, 50 tokens are cross-chained from the ETH chain to the BSC chain, making the balance on the ETH chain 150, while the recorded latestDstLiquidity of the BSC chain remains at 100.

3.After receiving the message on the BSC chain and initiating the transfer, the balance on the BSC chain becomes

50, while the recorded latestDstLiquidity of the ETH chain remains at 150.

4.Since the received message does not send the latest token balance (liquidity) on the BSC chain back to the ETH chain for an update, the latestDstLiquidity of the BSC chain recorded on the ETH chain remains at 100, rather than the new balance after the transfer(50).

Code Location:

contracts/TokenRelayer.sol

```
function bridgeTokens(uint256 amount) external payable whenNotPaused {
    ...

    if (amount > $.latestDstLiquidity) {
        revert BridgeTokens__InsufficientDstLiquidity();
    }

    ...
}

...

function receiveMessage(
    uint64 nonce,
    address account,
    uint256 amount,
    uint256 srcLiquidity
) external onlyRole(MESSAGE_RELAYER_ROLE) {
    ...

    $.latestDstLiquidity = srcLiquidity;

    if (amount != 0) {
        _creditUser(account, amount);
    }
}
```

Solution

It is recommended that after receiving the message and transferring the tokens on the target chain, another message should be sent back to the source chain to update the latestDstLiquidity value.

Status

Fixed; Update: The new fix will timely update the optimisticDstLiquidity value during cross-chain operations and add

a rollback function controlled by the admin role to handle the rollback of liquidity in case of failed cross-chain message execution.

[N2] [Low] Incorrect check of the fee.nativeFee

Category: Design Logic Audit

Content

In the TokenRelayer contract, any cross-chain operation requires calculating the transaction fees based on the input parameters. In the TokenRelayer contract, it only checks that the incoming msg.value is not less than fee.nativeFee. If the fee estimate + buffer is too large, no additional gas is consumed and it is refunded according to LZ logic. However, the issue is that the fee buffer was incorrectly applied to the fee.nativeFee whereas it should have been applied to the gas units passed into the LZReceive option.

Code Location:

contracts/TokenRelayer.sol#L97&L141&L253

```
    function addLiquidity(
        uint256 amount
    ) external payable onlyRole(MANAGER_ROLE) whenNotPaused {
        ...

        if (msg.value < fee.nativeFee) {
            revert AddLiquidity__InsufficientNativeFee();
        }

        ...
    }

    ...

    function bridgeTokens(uint256 amount) external payable whenNotPaused {
        ...

        if (msg.value < fee.nativeFee) {
            revert BridgeTokens__InsufficientNativeFee();
        }

        ...
    }

    ...
```

```
function removeLiquidity(
    uint256 amount
) external payable onlyRole(MANAGER_ROLE) whenNotPaused {
    ...

    if (msg.value < fee.nativeFee) {
        revert RemoveLiquidity__InsufficientNativeFee();
    }

    ...
}
```

Solution

It is recommended that the cost of buffering be applied to the gas units passed into the LZReceive option instead of fee.nativeFee.

Status

Fixed

[N3] [Suggestion] Missing scope limit

Category: Others

Content

In the TokenRelayer contract, the manager role can set various fees for cross-chain operations by calling the setExpectedAddLiquidityGas, setExpectedBridgeTokenGas, setExpectedRemoveLiquidityGas, and setFeeBufferBP functions. However, there is a lack of range checks on the fees being set; if the values set are too large or too small, it could potentially lead to cross-chain failures.

Code Location:

contracts/TokenRelayer.sol#L299-334

```
function setExpectedAddLiquidityGas(
    uint128 expectedAddLiquidityGas
) external onlyRole(MANAGER_ROLE) {
    ...
}

function setExpectedBridgeTokenGas(
    uint128 expectedBridgeTokenGas
) external onlyRole(MANAGER_ROLE) {
```

```
        ...
    }

    function setExpectedRemoveLiquidityGas(
        uint128 expectedRemoveLiquidityGas
    ) external onlyRole(MANAGER_ROLE) {
        ...
    }

    function setFeeBufferBP(
        uint256 feeBufferBP
    ) external onlyRole(MANAGER_ROLE) {
        ...
    }
```

Solution

It is recommended to perform the range checks when setting key parameters.

Status

Fixed

[N4] [Suggestion] Token compatibility issues

Category: Design Logic Audit

Content

In the TokenRelayer contract, when adding liquidity by calling the addLiquidity function or sending tokens across chains by calling the bridgeTokens function, tokens need to be transferred into this contract first. Then, the amount of liquidity added or tokens transferred should be sent across chains in the form of a message. However, these two functions do not check whether the difference in the contract's token balance before and after the transfer equals the value of the amount parameter. If the token is a deflationary token, it might result in the actual amount transferred being less than the value of amount, ultimately leading to unexpected errors.

Code Location:

contracts/TokenRelayer.sol#L77-160

```
    function addLiquidity(
        uint256 amount
    ) external payable onlyRole(MANAGER_ROLE) whenNotPaused {
        ...
```

```
        (
            bytes memory message,
            bytes memory options,
            MessagingFee memory fee
        ) = _composeActionData(
                _msgSender(),
                0,
                IERC20(BRIDGE_TOKEN).balanceOf(address(this)) + amount,
                $.expectedAddLiquidityGas
            );

        if (msg.value < fee.nativeFee) {
            revert AddLiquidity__InsufficientNativeFee();
        }

        IERC20(BRIDGE_TOKEN).safeTransferFrom(
            _msgSender(),
            address(this),
            amount
        );

        ...
    }

    function bridgeTokens(uint256 amount) external payable whenNotPaused {
        ...

        (
            bytes memory message,
            bytes memory options,
            MessagingFee memory fee
        ) = _composeActionData(
                _msgSender(),
                amount,
                IERC20(BRIDGE_TOKEN).balanceOf(address(this)) + amount,
                $.expectedBridgeTokenGas
            );

        if (msg.value < fee.nativeFee) {
            revert BridgeTokens__InsufficientNativeFee();
        }

        IERC20(BRIDGE_TOKEN).safeTransferFrom(
            _msgSender(),
            address(this),
            amount
        );
```

```
        ...
    }
```

Solution

It is recommended to use the difference in the token balance in the contract before and after the user's transfer as the actual amount, instead of directly using the amount parameter passed in.

Status

Acknowledged; Project team response: Only token which will be bridged is PROS, which is non-deflationary.

[N5] [Suggestion] Missing non-zero address check

Category: Others

Content

The admin role can set critical address variables in several functions, but there is a lack of non-zero address validation checks for the parameters passed in.

Code Location:

contracts/TokenRelayer.sol#L68-74

```
    function __TokenRelayer_init(address admin) external initializer {
        __UUPSUpgradeable_init();
        __AccessControl_init();

        _grantRole(DEFAULT_ADMIN_ROLE, admin);
        _pause();
    }
```

contracts/TokenRelayer.sol#L199-204

```
    function unsafeCredit(
        address account,
        uint256 amount
    ) external onlyRole(MANAGER_ROLE) {
        _creditUser(account, amount);
    }
```

contracts/TokenRelayer.sol#L375-381

```
function transferOwnership(
    address newOwner
) external onlyRole(MANAGER_ROLE) {
    IMessageRelayer(Storage.layout().messageRelayer).transferOwnership(
        newOwner
    );
}
```

Solution

It is recommended that the address non-zero checks should be added.

Status

Fixed

[N6] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

In the TokenRelayer contract, the MANAGER_ROLE can call the unsafeCredit function to transfer any amount of tokens from the contract to any address, and the destruction, clearing, or skipping of messages is also controlled by this role through calling the burnMessage, clearMessage, and skipMessage functions. In addition, the MANAGER_ROLE can also set the address of the MessageRelayer and related configurations of the MessageRelayer. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

contracts/TokenRelayer.sol

```
function burnMessage(
    bytes32 guid,
    address sender,
    uint64 nonce,
    bytes32 payloadHash
) external onlyRole(MANAGER_ROLE) whenNotPaused {
    ...
}

function clearMessage(
    Origin calldata origin,
    bytes32 guid,
    bytes calldata message
```

```
    ) external onlyRole(MANAGER_ROLE) {
        ...
    }

    ...

    function unsafeCredit(
        address account,
        uint256 amount
    ) external onlyRole(MANAGER_ROLE) {
        _creditUser(account, amount);
    }

    ...

    function setMessageRelayer(
        address messageRelayer
    ) external onlyRole(MANAGER_ROLE) {
        Storage.layout().messageRelayer = messageRelayer;

        emit MessageRelayerSet(messageRelayer);
    }

    ...


    function skipMessage(
        uint64 nonce,
        address sender
    ) external onlyRole(MANAGER_ROLE) {
        ...
    }

    ...

    function transferOwnership(
        address newOwner
    ) external onlyRole(MANAGER_ROLE) {
        IMessageRelayer(Storage.layout().messageRelayer).transferOwnership(
            newOwner
        );
    }
```

Solution

In the short term, during the early stages of the project, the protocol may need to frequently set various parameters to ensure the stable operation of the protocol. Therefore, transferring the ownership of core roles to a multisig

management can effectively solve the single-point risk, but it cannot mitigate the excessive privilege risk. In the long run, after the protocol stabilizes, transferring the owner ownership to community governance or executing through a timelock can effectively mitigate the excessive privilege risk and increase the community users' trust in the protocol.

Status

Acknowledged; Project team response: Permissions will be transferred to a multi-signature address for management.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002411110001	SlowMist Security Team	2024.11.07 - 2024.11.11	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 1 high risk, 1 medium risk, 1 low risk and 3 suggestion. All the findings were fixed or acknowledged. The code has been deployed to the mainnet. Since the manager role's permission will be managed by a multi-signature wallet, the risk level of the report is set to medium, considering the potential future risk of centralization.

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