LaPoste Security Review

Pashov Audit Group

Conducted by: T1MOH, btk, peanuts

October 22th - October 24th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **stake-dao/laposte** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About LaPoste

LaPoste is a utility contract that streamlines cross-chain communication and token transfers between blockchains. It uses Chainlink’s CCIP for interactions across networks.

5. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 0bf0be29ea9d78aa147cc7a924a4c10dba262669

fixes review commit hash - 0c1065d364692b396aa4bc49dddf2d18d3975f98

Scope

The following smart contracts were in scope of the audit:

- `TokenFactory`
- `Token`
- `LaPoste`
- `Adapter`
- `Client`
- `Chains`

7. Executive Summary

Over the course of the security review, T1MOH, btk, peanuts engaged with LaPoste to review LaPoste. In this period of time a total of **9** issues were uncovered.

Protocol Summary

Protocol Name	LaPoste
Repository	https://github.com/stake-dao/laposte
Date	October 22th - October 24th
Protocol Type	Crosschain messages

Findings Count

Severity	**Amount**
Medium	3
Low	6
Total Findings	**9**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Protocol will not work on ZkSync and Celo	Medium	Resolved
[M-02]	Manual Execution will not work with the current nonce implementation	Medium	Resolved
[M-03]	receiveMessage() will revert when it runs out of gas	Medium	Resolved
[L-01]	Adapter.sol does not handle some chainIds	Low	Resolved
[L-02]	Missing a fallback function	Low	Acknowledged
[L-03]	MessageReceived event can be confusing if there is no payload	Low	Resolved
[L-04]	EVMExtraArgsV2 is not used	Low	Resolved
[L-05]	Excess msg.value should be returned to the user	Low	Resolved
[L-06]	Fee-on-Transfer and Rebasing token will not work properly	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] Protocol will not work on ZkSync and Celo

Severity

Impact: Medium

Likelihood: Medium

Description

Every chain contains a single version of LaPoste.sol, Adapter.sol, TokenFactory.sol.

When a message is received, it assumes that LaPoste.sol shares the same address across all the chains:

```
/// @notice Receives a message from another chain
    /// @param message The received message
    function ccipReceive
      (Client.Any2EVMMessage calldata message) external override onlyRouter {
        address source = abi.decode(message.sender, (address));
@>      if (source == address(0) || source != laPoste) revert InvalidSender();

        ILaPoste(laPoste).receiveMessage({chainId: getChainId
          (message.sourceChainSelector), payload: message.data});
    }
```

However, ZkSync has different formula derivation, so it's impossible to deploy to the same address.

Similar issues with Celo: it has different address derivation from seed phrase, it means create3 will result in a different address because the same tx sender can't be reproduced on Celo.

The same goes vice versa: in the current design, LaPoste can't call Adapter on such a chain, because the adapter has a different address.

Recommendations

Use specific addresses when sending messages to/from ZkSync and Celo.

[M-02] Manual Execution will not work with the current nonce implementation

Severity

Impact: High

Likelihood: Low

Description

When sending messages / tokens through CCIP, if the transaction fails on the destination chain, CCIP offers an option to manually execute (retry) the transaction again.

One case where the transaction may fail is when the receiver contract on the destination blockchain reverted due to the gas limit being insufficient to execute the triggered function (Note: The gas limit value is set in the extraArgs param of the message) (Source: Chainlink Docs)

In LaPoste.sol, when `receiveMessage` is successful, `receivedNonce[chainId]` is increased by `message.nonce`. The function also checks `message.nonce <= receivedNonces[chainId]`.

```
function receiveMessage
      (uint256 chainId, bytes calldata payload) external onlyAdapter {
        ILaPoste.Message memory message = abi.decode(payload,
          (ILaPoste.Message));

        // Check if the message has already been processed
        if
          (message.nonce <= receivedNonces[chainId]) revert MessageAlreadyProcessed();
```

This nonce value is set when sending message, and it is incremented by one every time. Consider this scenario:

- Message 1 (with tokens locked on the source chain) is sent to the destination chain, `message.nonce = 1`, and `receivedNonces[chainId]= 0`. The check passes, the sender gets his minted tokens, and `receivedNonces[chainId]` is 1.
- Message 2 (tokens locked on source chain) is sent to destination chain. `message.nonce = 2`, and `receivedNonces[chainId]= 1`. Something happened to the gas limit and this transaction reverts.
- Message 3 (tokens locked on source chain) is sent to destination chain. `message.nonce = 3`, and `receivedNonces[chainId]= 1`. Transaction passes and `receivedNonces[chainId]= 3`. Sender gets his minted tokens.
- Message 2 sender wants to retry his message through manual execution, but since `message.nonce = 2`, and `receivedNonces[chainId]= 3`, the function will always revert. The sender's token is locked on the source chain.

Recommendations

One recommendation from the docs is to use `EVMExtraArgsV2` with `allowOutOfOrderExecution` set to false so that the messaging order is sequential.

[M-03] `receiveMessage()` will revert when it runs out of gas

Severity

Impact: Medium

Likelihood: Medium

Description

Protocol uses CCIP messaging in a strict way. It means that further messages per lane can't be processed until the execution of previous messages succeeds. As soon as any action can revert in `LaPoste.receiveMessage()`, further messages can't be processed.

How an attacker can make his message revert. It uses `try catch` to execute a call to the attacker's contract:

```
/// 2. Execute the message.
        bool success;
        if (message.payload.length > 0) {
@>          try IMessageReceiver(message.to).receiveMessage
  (chainId, message.sender, message.payload) {
                success = true;
            } catch {
                success = false;
            }
        }
```

The problem is that `try catch` doesn't handle the out-of-gas error. So basically attacker can waste all gas in this external call, so that `LaPoste.receiveMessage()` always reverts.

An attacker can perform such an attack on every CCIP lane so that any communication in LaPoste protocol will be DOSed. As a result, locked tokens on Mainnet will be locked forever, which means users lose money.

Recommendations

Add `executionGasLimit` field to ILaposte.Message struct. And specify this gasLimit in external call here:

```
/// 2. Execute the message.
        bool success;
        if (message.payload.length > 0) {
-           try IMessageReceiver(message.to).receiveMessage
- (chainId, message.sender, message.payload) {
+           try IMessageReceiver
+ (message.to).{gas: message.executionGasLimit}receiveMessage(chainId, message.sender,
                success = true;
            } catch {
                success = false;
            }
        }
```

8.2. Low Findings

[L-01] Adapter.sol does not handle some chainIds

Library `Chains` contains 17 chains. However, Adapter.sol handles only 13 chains in functions `getChainId()` and `getBridgeChainId()`. As a result, chains WEMIX, METIS, MODE, KROMA can't be used in the protocol.

[L-02] Missing a fallback function

According to Chainlink's documentation, it is considered best practice to implement fallback mechanisms in CCIP receiver contracts to handle unexpected exceptions smoothly. However, the LaPoste contract does not follow this guideline, which could potentially require the protocol to deploy new contracts in certain scenarios. Consider implementing a fallback function.

[L-03] MessageReceived event can be confusing if there is no payload

The receiveMessage() function in LaPoste.sol creates a success variable and changes its value if `message.payload.length > 0`, then emits the `MessageReceived` event.

```
bool success;
        if (message.payload.length > 0) {
            try IMessageReceiver(message.to).receiveMessage
              (chainId, message.sender, message.payload) {
                success = true;
            } catch {
                success = false;
            }
        }
        emit MessageReceived(
          chainId,
          message.nonce,
          message.sender,
          message.to,
          message,
          success
        );
```

The confusion arises when `message.payload.length` is zero, and `success` is automatically set to false but the transaction is successful.

In this transaction on the destination chain, on line 20, the payload is empty and the success value is set to false, but the transaction is a success.

Have two types of events, and emit the event accordingly if `message.payload.length` is more than zero or zero.

```
if (message.payload.length > 0) {
        emit MessageReceived(
          chainId,
          message.nonce,
          message.sender,
          message.to,
          message,
          success
        );
} else {
        success = true;
        emit MessageReceivedNoPayload(
          chainId,
          message.nonce,
          message.sender,
          message.to,
          message,
          success
        );
}
```

[L-04] EVMExtraArgsV2 is not used

It seems like only `EVMExtraArgsV1` is used when sending messages using CCIP.

```
Client.EVMExtraArgsV1 memory evmExtraArgs = Client.EVMExtraArgsV1
     ({gasLimit: totalGasLimit});
        bytes memory extraArgs = Client._argsToBytes(evmExtraArgs);

        Client.EVM2AnyMessage memory ccipMessage = Client.EVM2AnyMessage({
            receiver: abi.encode(to),
            data: message,
            tokenAmounts: new Client.EVMTokenAmount[](0),
            feeToken: address(0),
            extraArgs: extraArgs
        });
```

The Client contract has `EVMExtraArgsV2` but it is unused.

```
struct EVMExtraArgsV2 {
        uint256 gasLimit;
        bool allowOutOfOrderExecution;
    }

    function _argsToBytes
      (EVMExtraArgsV2 memory extraArgs) internal pure returns (bytes memory bts) {
        return abi.encodeWithSelector(EVM_EXTRA_ARGS_V2_TAG, extraArgs);
    }
```

Consider removing unused code.

[L-05] Excess msg.value should be returned to the user

The protocol uses native tokens instead of LINK tokens to pay for CCIP fees. The Adapter.sendMessage() function checks `msg.value < fee`, but in the case where the user accidentally sends an extra `msg.value`, it will be stuck in the contract.

```
uint256 fee = router.getFee(chainSelector, ccipMessage);

        if (fee == 0) revert InvalidMessage();
        if (msg.value < fee) revert NotEnoughFee();
        return router.ccipSend{value: fee}(chainSelector, ccipMessage);
```

Return excess `msg.value` to the caller.

[L-06] Fee-on-Transfer and Rebasing token will not work properly

The protocol provides a single entry point for cross-chain interactions and leverages Chainlink's CCIP for secure and reliable message and token transfers.

There is no mention of what tokens are allowed, so it is assumed that FoT/Rebasing tokens can be used as well.

```
>   ITokenFactory(tokenFactory).burn
  (messageParams.token.tokenAddress, msg.sender, messageParams.token.amount);

            (
              message.tokenMetadata.name,
              message.tokenMetadata.symbol,
              message.tokenMetadata.decimals
            ) =
                ITokenFactory(tokenFactory).getTokenMetadata
                  (messageParams.token.tokenAddress);
```

The user will face issues when using those tokens with the protocol.

For Rebasing tokens:

The rebase rewards will be stuck in the contract since the user can only get the same amount from the source chain to the destination chain and back.

For Fee-on-Transfer tokens:

The fee will not be taken into account when minting on the destination chain.

A recommendation will be to whitelist tokens, but If the protocol does intend to allow any type of token to be accepted, ensure that the user knows the issues faced when using uncommon ERC20 tokens.