Beam Bridge Security Review

Pashov Audit Group

Conducted by: Hals, btk, Shaka

March 28th 2025 - March 29th 2025

Contents

1. About Pashov Audit Group 2
2. Disclaimer 2
3. Introduction 2
4. About Beam Bridge 2
5. Risk Classification 3
   5.1. Impact 3
   5.2. Likelihood 3
   5.3. Action required for severity levels 4
6. Security Assessment Summary 4
7. Executive Summary 5
8. Findings 7
   8.1. High Findings 7
      [H-01] No guaranteed minimum received amount when fees are zero 7
   8.2. Medium Findings 10
      [M-01] Incorrect amount pulled from sender in BeamOFTAdapter 10
      [M-02] _debit() Fee logic results in unintended higher fee deduction 10
   8.3. Low Findings 13
      [L-01] Incorrect fee percentage precision 13
      [L-02] Approval requirement for fee transfer is unnecessary 13
      [L-03] Transfer might revert if customFee is zero 14
      [L-04] Deployment failure when owner is not the sender 14
      [L-05] Loss of funds if bridged token supply is too high 15

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 **BuildOnBeam/layerzero-v2-bridge-builder** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Beam Bridge

Beam Bridge utilizes LayerZero's OFT (Omnichain Fungible Token) and ONFT (Omnichain Non-Fungible Token) standards, along with oftadapter and onftadapter, to enable seamless ERC20 and ERC721 token bridging between Beam Network and other blockchains.

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 - 816f18cc694f30cd2a009469410efadf514a5d33

fixes review commit hash - ddc85b81b92c7775bcb8090a1e463bfa8813db4f

Scope

The following smart contracts were in scope of the audit:

- `BeamOFT`
- `BeamOFTAdapter`
- `BaseBeamBridge`

7. Executive Summary

Over the course of the security review, Hals, btk, Shaka engaged with Beam to review Beam Bridge. In this period of time a total of **8** issues were uncovered.

Protocol Summary

Protocol Name	Beam Bridge
Repository	https://github.com/BuildOnBeam/layerzero-v2-bridge-builder
Date	March 28th 2025 - March 29th 2025
Protocol Type	Bridge

Findings Count

Severity	**Amount**
High	1
Medium	2
Low	5
Total Findings	**8**

Summary of Findings

ID	Title	Severity	Status
[H-01]	No guaranteed minimum received amount when fees are zero	High	Resolved
[M-01]	Incorrect amount pulled from sender in BeamOFTAdapter	Medium	Resolved
[M-02]	_debit() Fee logic results in unintended higher fee deduction	Medium	Resolved
[L-01]	Incorrect fee percentage precision	Low	Resolved
[L-02]	Approval requirement for fee transfer is unnecessary	Low	Resolved
[L-03]	Transfer might revert if customFee is zero	Low	Resolved
[L-04]	Deployment failure when owner is not the sender	Low	Resolved
[L-05]	Loss of funds if bridged token supply is too high	Low	Resolved

8. Findings

8.1. High Findings

[H-01] No guaranteed minimum received amount when fees are zero

Severity

Impact: Medium

Likelihood: High

Description

Both `BeamOFT` and `BeamOFTAdapter` contracts override the `_debitView()` function from `OFTCore` to support custom fees.

For the case when the fee is set to zero, `amountReceivedLD` is set to `_amountLD`. This is in contrast with the default implementation, which sets `amountReceivedLD` to the amount sent after removing the dust.

```
File: OFTCore.sol

        // @dev Remove the dust so nothing is lost on the conversion between chains
 // with different decimals for the token.
        amountSentLD = _removeDust(_amountLD);
        // @dev The amount to send is the same as amount received in the default
 // implementation.
        amountReceivedLD = amountSentLD;
```

```
File: BeamOFT.sol / BeamOFTAdapter.sol

        } else {
                amountReceivedLD = _amountLD;
                amountSentLD = _removeDust(_amountLD);
        }
```

This means that the dust amount is not removed from `amountReceivedLD`, so `amountReceivedLD` can be greater than `amountSentLD`.

Fortunately, in `OFTCore` the amount added to the cross-chain message is divided by `decimalConversionRate`, which prevents that more tokens than the amount sent are received on the other side. That is, both the amount sent and the amount received discard the dust amount.

However, the amount received can still be less than the minimum amount required by the user, which might force the user to submit a new transaction.

Imagine the following scenario:

- Alice needs 1.9e12 tokens in the target chain to perform an operation, so she sets the amount sent and minimum amount received to 1.9e12.
- The transaction is processed and 1e12 tokens are sent to the target chain.
- Alice does not have the required amount in the target chain, so she is forced to submit a new transaction.

Proof of concept

Add the following code to the file `BeamBridge.t.sol` and run `forge test --mt test_receiveLessThanMinAmount`.

```
function test_receiveLessThanMinAmount() public {
        bOFT.setFeePercentage(0);
        oftAdapterToOft();
        uint256 tokensToSend = 1.9e12;
        uint256 minTokensToReceive = tokensToSend;
        uint256 initBalTarget = aToken.balanceOf(userD);

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption
    (200000, 0);
        SendParam memory sendParam = SendParam(
                aEid,
                addressToBytes32(userD),
                tokensToSend,
                minTokensToReceive,
                options,
                "",
                ""
        );
        MessagingFee memory fee = bOFT.quoteSend(sendParam, false);

        vm.prank(userD);
        bOFT.approve(address(bOFT), tokensToSend);

        vm.prank(userD);
        (, OFTReceipt memory oftReceipt) = bOFT.send{ value: fee.nativeFee }
    (sendParam, fee, payable(address(this)));

        // The receipt confirms that we receive the expected amount
        assertEq(oftReceipt.amountReceivedLD, minTokensToReceive);

        verifyPackets(aEid, addressToBytes32(address(aOFTAdapter)));

        // However, the actual amount received is lower than expected
        uint256 received = aToken.balanceOf(userD) - initBalTarget;
        assertEq(received, 1e12);
}
```

Recommendations

Apply the following changes in the `BeamOFT` and `BeamOFTAdapter` contracts:

```
-       amountReceivedLD = _amountLD;
        amountSentLD = _removeDust(_amountLD);
+       amountReceivedLD = amountSentLD;
```

8.2. Medium Findings

[M-01] Incorrect amount pulled from sender in `BeamOFTAdapter`

Severity

Impact: High

Likelihood: Low

Description

In the `_debit()` function of the `BeamOFTAdapter` contract, when the fee percentage is 0, `amountReceivedLD` tokens are pulled from the sender. This is incorrect, as the amount should be `amountSentLD` instead.

This will not be a problem when both amounts are equal, which would be the expected behavior when no fee is applied. However, if the contract was to be inherited or modified, making it possible that `amountReceivedLD` is lower than `amountSentLD`, this would break the token accounting, as fewer tokens than expected would be pulled from the sender.

Additionally, there is another issue that reports that `amountReceivedLD` can be greater than `amountSentLD`, due to the dust not being removed from the `amountReceivedLD` variable. That issue is aggrieved by this one, as more tokens would be pulled from the sender than expected.

Recommendations

```
} else {
-           innerToken.safeTransferFrom(_from, address(this), amountReceivedLD);
+           innerToken.safeTransferFrom(_from, address(this), amountSentLD);
        }
```

[M-02] `_debit()` Fee logic results in

unintended higher fee deduction

Severity

Impact: Medium

Likelihood: Medium

Description

In the `_debit()` function, the fee is calculated as follows:

```
function _debit(/**/) internal view virtual override returns
     (uint256 amountSentLD, uint256 amountReceivedLD) {

        (amountSentLD, amountReceivedLD) = _debitView(
            _amountLD,
            _minAmountLD,
            _dstEid
        );

        // @dev Burn OFT and send custom fees to fee receiver if custom fees are
        // enabled
        if (s_feePercentage > 0) {
            uint256 customFee = _amountLD - amountReceivedLD;
            innerToken.safeTransferFrom(_from, s_feeReceiver, customFee);
           //...
        } else {
            //...
        }
    }
```

However, this calculation can result in the user being charged more fees than intended. The issue arises because `amountReceivedLD` is calculated after applying the fee and removing the dust by `_debitView()`:

```
function _debitView(
        uint256 _amountLD,
        uint256 _minAmountLD,
        uint32 /*_dstEid*/
    ) internal view virtual override returns
      (uint256 amountSentLD, uint256 amountReceivedLD) {
        amountSentLD = _amountLD;
        if (s_feePercentage > 0) {
            uint256 calculatedFees = (_amountLD * s_feePercentage) / PRECISION;

            amountReceivedLD = _removeDust(_amountLD - calculatedFees);
        } else {
           //...
        }

           //...
    }
```

Since the operation of removing dust may result in a lower value for `amountReceivedLD`, the `customFee` calculated as the difference between `_amountLD` and `amountReceivedLD` could be higher than the actual fee that was intended.

This discrepancy occurs because the dust removal process is applied to the fee-inclusive amount, which could cause the `customFee` to be greater than the actual intended fee.

Recommendations

the `customFee` should be the original calculated fee amount based on the `_amountLD`.

8.3. Low Findings

[L-01] Incorrect fee percentage precision

The `BaseBeamBridge` contract specifies that the initial `_feePercentage` should be expressed with `1e18` precision. However, this is inconsistent with the implementation in both `BeamPFTAdapter` and `BeamOFT`, which divide by `1e6` in the `debitView()` function. This discrepancy will lead to a DoS if the fee percentage is set incorrectly during deployment.

```
*
   @param _feePercentage Initial fee percentage of transactions, should be expressed w
```

To resolve this issue, the initial `_feePercentage` should be expressed in `1e6` precision during initialization.

[L-02] Approval requirement for fee transfer is unnecessary

When fees are set, in the `BeamOFT._debit()` function `customFee` is sent to the fee receiver. This requires the sender to approve `BeamOFT` to spend `BeamOFT` tokens. This is an unnecessary inconvenience for the user and also inconsistent with the burning of tokens, which is done using the internal `_burn()` function.

Given that the tokens sent are from the `BeamOFT` contract itself, it is recommended to use the internal `_transfer()` function, avoiding the need for extra calls from the user to approve the contract.

```
File: BeamOFT.sol

        uint256 customFee = _amountLD - amountReceivedLD;
-       IERC20(address(this)).safeTransferFrom(_from, s_feeReceiver, customFee);
+       _transfer(_from, s_feeReceiver, customFee);
        _burn(_from, amountSentLD - customFee);
```

[L-03] Transfer might revert if `customFee` is zero

When fees are set, in the `BeamOFTAdapter._debit()` function `customFee` is sent to the fee receiver. For small amounts, `customFee` can be truncated to zero. Certain tokens, like BNB revert when transferring zero tokens.

To avoid this, it is recommended to check if `customFee` is greater than zero before transferring it.

```
File: BeamOFTAdapter.sol

    uint256 customFee = _amountLD - amountReceivedLD;
+   if (customFee > 0) {
        innerToken.safeTransferFrom(_from, s_feeReceiver, customFee);
+   }
    innerToken.safeTransferFrom(_from, address(this), amountSentLD - customFee);
```

[L-04] Deployment failure when owner is not the sender

The `BaseBeamBridge` contract is designed to manage cross-chain token transfers and includes functions for setting a fee percentage and a fee receiver. The constructor of this contract calls `setFeePercentage()` and `setFeeReceiver()` to initialize these values.

However, if the `_delegate` address provided during deployment is not the same as the `msg.sender`, the contract cannot be deployed successfully. This is due to the fact that the functions being called in the constructor are restricted to the owner, which in this case would be the `_delegate`.

```
function testDifferentOwner() public {
        BeamOFTAdapter _aOFTAdapter;

        // Revert with OwnableUnauthorizedAccount
        _aOFTAdapter = BeamOFTAdapter(
            _deployOApp(
                type(BeamOFTAdapter).creationCode,
                abi.encode(address(aToken), address
                  (endpoints[aEid]), userA, feePercentage)
            )
        );
    }
```

Recommended The logic for setting the fee percentage and fee receiver should be moved to internal functions that can be called from the constructor. Additionally, implement public functions that invoke these internal functions.

[L-05] Loss of funds if bridged token supply is too high

By default, the OFT maximum supply is ~18.45T. If the supply of the token in its native chain exceeds this value (for example, Shiba Inu has a total supply of ~589.5T), a potential loss of funds may occur when this limit is reached.

Add the following code to the file `BeamBridge.t.sol` and run `forge test --mt test_high_supply_token`.

```
function test_high_supply_token() public {
        uint256 highAmount = 1.85e13 * 1e18;
        aToken.mint(userA, highAmount);
        aOFTAdapter.setFeePercentage(0);

        uint256 initBalSource = aToken.balanceOf(userA);
        uint256 initBalTarget = bOFT.balanceOf(userA);

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption
    (200000, 0);
        SendParam memory sendParam = SendParam(
                bEid,
                addressToBytes32(userA),
                highAmount,
                highAmount,
                options,
                "",
                ""
        );
        MessagingFee memory fee = aOFTAdapter.quoteSend(sendParam, false);

        vm.prank(userA);
        aToken.approve(address(aOFTAdapter), highAmount);

        vm.prank(userA);
        aOFTAdapter.send{ value: fee.nativeFee }(sendParam, fee, payable(address
    (this)));
        verifyPackets(bEid, addressToBytes32(address(bOFT)));

        uint256 sent = initBalSource - aToken.balanceOf(userA);
        uint256 received = bOFT.balanceOf(userA) - initBalTarget;
        assertEq(sent, highAmount);
        assert(received < highAmount / 100);
}
```

Provide a mechanism to override the `shareDecimals` value at deployment time for the `BeamOFT` and `BeamOFTAdapter` contracts and document this limitation properly.