OpenZeppelin | security

BridgeV2 Audit

October 24, 2025

Table of Contents

Summary

Type	Bridge
Timeline	From 29-09-2025 To 07-10-2025
Languages	Solidity

Total Issues	16 (16 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	4 (4 resolved)
Low Severity Issues	5 (5 resolved)
Notes & Additional Information	7 (7 resolved)

Scope

OpenZeppelin audited the lombard-finance/smart-contracts repository at commit 63d4076.

In scope were the following files:

```
contracts
├── LBTC
│   └── BridgeTokenAdapter.sol
│
└── bridge
    ├── providers
    │   ├── BridgeTokenPool.sol
    │   └── LombardTokenPoolV2.sol
    │
    └── BridgeV2.sol
```

System Overview

In this audit, four smart contracts of the Lombard Protocol were reviewed. These contracts facilitate cross-chain token bridging and integration with Chainlink's CCIP (Cross-Chain Interoperability Protocol). Specifically, the audit scope included the following contracts:

- `BridgeV2`: This contract handles token deposits—either directly from users or indirectly through CCIP relayers. Upon receiving a deposit, the contract burns the tokens and emits a message for the Mailbox (as described in our previous report). Relayers monitor these messages and trigger the corresponding mint function on the destination chain to complete the bridging process.
- `LombardTokenPoolV2`: This contract enables integration with the CCIP system. Instead of interacting directly with the bridge, users interact through CCIP. Their tokens are transferred to the Pool contract, which invokes the `lockOrBurn` function. After performing a series of validations, this function calls the deposit function on the bridge contract. Conversely, on the destination chain, the `releaseOrMint` function is executed to validate inputs, communicate with the `Mailbox`, and instruct the bridge to mint tokens to the intended recipient by calling `handlePayload` function.
- `BridgeTokenPool`: This contract is a variation of `LombardTokenPoolV2`, designed to operate with token adapters instead of directly handling the underlying tokens.
- `BridgeTokenAdapter`: Since the `BridgeV2` contract requires tokens to implement a specific interface, the tokens that do not conform to this interface will have to use this adapter. The adapter acts as a wrapper token, with permission to mint and burn the underlying asset, thereby ensuring compatibility with the bridge. This contract has been developed to accommodate the BTC.b token to support the Lombard bridge, and will be granted minting privileges for the BTC.b token on Avalanche. The minting and burning operations for BTC.b must be routed through the adapter contract. If a user attempts to burn their BTC.b tokens to get back BTC directly calling the `unwrap` function of BTC.b contract, the Lombard team will simply mint these tokens back to the user, who should then follow the intended process.

Security Model and Trust Assumptions

The contracts reviewed in this audit interact with multiple external components, and their overall security depends on the correct and expected behavior of these dependencies.

In particular, the system is designed to integrate with Chainlink's CCIP. As such, it is assumed that CCIP functions as intended, and that tokens have been successfully transferred to the appropriate Pool contract (`LombardTokenPoolV2` or `BridgeTokenPool`) before invoking the `lockOrBurn` function. Since CCIP does not support forwarding additional value via `msg.value`, it is expected that the `BridgeV2` contract will configure the maximum fee discount for the Pool contracts, effectively ensuring that CCIP-based transactions incur zero fees.

The latest Version of CCIP supports tokens with different decimal configurations on each side of the bridge. In a burn-and-mint bridge, this can lead to a loss of precision when tokens are transferred from a chain with more decimals to one with fewer. Since tokens on the source side are burned, any loss of precision results in a permanent loss of those tokens. The Lombard team is expected to configure the tokens on both sides with the same number of decimals.

Furthermore, CCIP imposes gas limits on the operations it executes. It is assumed that the Lombard team has reviewed these limits and configured them appropriately to guarantee that all required actions can be completed without exceeding the available gas. The bridge relies on relayers to transmit messages between chains. There is no built-in mechanism to cancel a deposit that has not been finalized. Therefore, it is assumed that all legitimate deposits will be finalized by the relayers in a timely manner.

The bridge implements a rate limit on token minting on the destination side, but not on deposits at the source. Consequently, a large deposit may succeed on the source chain but remain pending until the rate limit on the destination chain is raised. It is expected that the bridge owner will manually increase the rate limit when such a deposit is identified to allow its successful finalization.

The `BridgeV2` contract exposes two `deposit` functions:

- The first is intended to be called by whitelisted relayers, such as the CCIP Token Pool contracts. Any user can interact with these relayers to bridge tokens. It is expected that

the relayer has already received and holds the tokens before invoking the bridge, as these tokens will be burned from the relayer's address.
- The second deposit function is designed for direct interaction by whitelisted users.

The `BridgeTokenAdapter` contract is expected to grant the `MINTER` role exclusively to the Bridge contract (there is also a `batchMint` function, but this is not used by the `BridgeV2` and is included solely for interface compatibility), and must have both minting and burning permissions for the underlying BTC.b Token. The bridge is in turn expected to grant appropriate token allowances to the adapter contract. The BTC.b token will grant the `BridgeTokenAdaptor` minting privileges through `migrateBridgeRole` function. The `unwrap` function in the BTC.b contract only allows EOAs to unwrap the token. Hence, special care will be taken if `BrdigeTokenAdaptor` is upgraded to support the `unwrap` functionality of the BTC.b token.

Finally, the `spendDeposit` function in the adapter contract is expected to be called only once, at the time the underlying token grants the adapter its minter role. This call is solely for registering the total balance of the underlying token in the internal ledger.

Privileged Roles

Throughout the in-scope codebase, the following privileged roles were identified:

- **Owner of the `BridgeV2` Contract** : The owner has the authority to add or remove chains where bridging is permitted, define allowed pairs of source and destination tokens, configure fee discounts for specific addresses, and set rate limits for tokens minted on the destination chain. It is expected that the owner will ensure consistency across chains by configuring the same token pairs on both sides of each bridge. For example, if a token is allowed to be bridged from Chain A to Chain B, the reverse pairing (from Chain B to Chain A) should also be enabled. Moreover, the owner is responsible for adjusting rate limits when necessary, such as to accommodate larger deposits.

- **Owner of the `LombardTokenPoolV2` and `BridgeTokenPool` Contracts**: The owner is responsible for configuring pool-related token details and maintaining up-to-date information about tokens that can be bridged. It is expected that the configuration of these Pool contracts remains consistent with that of the `BridgeV2` contract to ensure coherent bridging behavior across the system.

- `BridgeTokenAdapter` Roles:
  - **Default Admin Role**: The holder of this role can assign other roles, update key protocol parameters (such as consortium configurations), and unpause the protocol if it has been paused.
  - **Pauser Role**: The holder of this role can pause the protocol in case of emergencies or detected anomalies. It is expected to be used only when necessary to prevent potential harm to users or the system.
  - **Minter Role**: The bridge adapter must be authorized by the underlying token contract to mint and burn tokens on arbitrary addresses. The holder of the `MINTER` role can perform these actions. It is expected that this role is exclusively assigned to the bridge contract, which should invoke mint and burn functions only in response to legitimate deposit and bridging events.

Medium Severity

M-01 Token Pool Deployment May Fail for Some Tokens

The constructor of the `LombardTokenPoolV2` contract calls `token_.decimals()` function to fetch the decimals of the token. However, according to the ERC-20 spec, the `decimals` function is optional. Hence, there is a possibility that, for a given token, the `decimals` function may not exist at all. The inherited `TokenPool` contract of the CCIP acknowledges this possibility and resolves using `try-catch` blocks. This could lead to the `LombardTokenPoolV2` contract never be deployed for certain tokens.

Since the inherited `TokenPool` contract already calls `decimals` function on the token and compares against the given constructor parameter, instead of relying on an on-chain call that may revert, consider modifying the constructor so that it accepts the decimals value as an explicit parameter that is supplied off-chain or through the configuration.

Update: *Resolved at commit 3c71001. The Lombard team stated:*

> Implemented fallback ratio for `LombardTokenPoolV2`. `BridgeTokenPool` always sets the fallback ratio to 0 because the implementation of the contract is known before deployment.

M-02 Lack of Cross-Verification for Destination Token

The `lockOrBurn` function of the `LombardTokenPoolV2` and `BridgeTokenPool` contracts is designed to be invoked by the CCIP and calls the `deposit` function of the `BridgeV2` contract, which in turn burns the local token referenced by the `i_token` variable. Within the `deposit` function, a verification step ensures that `i_token` is an approved token for bridging and the target chain is permitted. The `destinationToken` address is also retrieved from the `allowedDestinationToken` mapping of the `BridgeV2` contract.

The Pool contracts maintain their own record of the destination token, which can be obtained via the `getRemoteToken` function. These two values are expected to be the same. However, this consistency is not verified either during the setup phase, when the values are assigned, or

during the execution of the `lockOrBurn` function. A mismatch between these two addresses could cause bridge malfunction, as CCIP may attempt to interact with a destination token pool corresponding to a different token than the one expected by the bridge.

Consider either adding a consistency check in `lockOrBurn` ensuring that these two values are equal or, when the owner invokes `applyChainUpdates` in the Pool contract, validating that the destination token aligns with the bridge's value for the destination token.

Update: *Resolved at commits b736e59, ef5e24f and 987a4af.*

M-03 Unnecessary Infinite Approval to Bridge by the `BridgeTokenPool` Contract

In the `lockOrBurn` function of the `BridgeTokenPool` contract, a call is made to the `deposit` function of the `BridgeV2` contract. This function internally invokes `_burnToken`, which in turn calls `transferFrom` of the `BridgeTokenAdapter` contract. The `transferFrom` operation first moves the underlying `BridgeToken` to the `BridgeTokenAdapter` from the caller which is the `BridgeTokenPool` contract and then transfers the tokens to the `BridgeV2` contract.

For this operation to succeed, the `BridgeTokenPool` contract must have granted approval to the `BridgeTokenAdapter` contract. This approval is granted in the constructor of the `BridgeTokenPool` contract. However, the `BridgeTokenPool` contract inherits `LombardTokenPoolV2`, and in the constructor of the latter, infinite approval is granted to the `BridgeV2` contract. This approval is necessary for tokens without an adapter, as for them, the `_burnToken` function transfers the tokens directly from the pool to the Bridge contract. In contrast, the additional infinite approval to the `BridgeTokenAdapter` in `BridgeTokenPool` is redundant. Although the `BridgeTokenPool` contract is not expected to hold tokens, maintaining this unnecessary infinite approval is risky, especially if changes are introduced in future upgrades.

Consider removing the unnecessary infinite approval given to the `BridgeV2` contract in the `BridgeTokenPool`'s constructor.

Update: *Resolved at commit 4b2cf8b.*

M-04 Insufficient Input Validation in `deposit` May Lead to Stuck Bridging Transactions

In the `BridgeV2` contract, the `deposit` function accepts the `recipient` as a `bytes32` argument, since the destination chain is not necessarily EVM compatible. Currently, the function only checks that the `recipient` is non-zero.

However, when bridging to an EVM-compatible destination chain, the `recipient` is converted to an `address` in the destination by taking the lowest 20 bytes and verifying that the remaining are zero. If this condition is not met, the transaction on the destination chain will revert, preventing the bridging process from finalizing and potentially causing permanently locked funds. This issue could occur either accidentally or intentionally (e.g., to force the relayers to attempt to finalize a transaction that will always revert). A similar issue exists with the `sender` argument: on the destination side, a non-zero sender is required, but `deposit` does not verify this.

Consider adding these extra checks which will guarantee that the cross-chain transactions can always be finalized.

Update: *Resolved at commit c1c6528, at commit e8d9daf, and at commit 9fca809.*

Low Severity

L-01 Missing Validation in `addDestinationToken` May Cause Unusable Bridge Paths

In the `BridgeV2` contract, the `addDestinationToken` function verifies that `destinationChain` and `sourceToken` are non-zero, but it never checks that the supplied `destinationToken` is also non-zero. As a result, the owner could accidentally store a zero address as the destination token. Such a mistake would force any deposit involving that `sourceToken` and `destinationChain` to revert.

Except for the check that the `destinationToken` is non-zero, extra validation is needed to ensure that the `destinationToken` is compatible with the expected address format in the destination chain. For example, for EVM chains, all higher-order bytes, except for the lowest 20 bytes, should be 0. Otherwise, the transaction will revert on the destination chain.

Consider adding additional input validation for `destinationToken` to prevent accidental misconfiguration.

Update: *Resolved at commit c083e0f and commit 4050f9a.*

L-02 Missing Input Validation

Several functions in the codebase lack proper input validation to ensure that critical parameters are non-zero or correctly configured. Such missing checks can result in unintended behavior.

Throughout the codebase, multiple instances of missing input validation were identified:

- The `constructor` of `BridgeTokenPool` does not validate that the `tokenAdapter` address is non-zero.
- The `BridgeV2::setTokenRateLimits` function does not verify that the `Config.window` is non-zero. If `window` is set to zero, the `else` branch in `RateLimits::availableAmountToSend` will always revert due to a division by zero.
- The `BridgeV2::rescueERC20` function does not check that the `to` address is non-zero. While many ERC-20 tokens revert on transfers to the zero address, this behavior is not guaranteed for all implementations.

Consider adding explicit input validation to avoid any problems.

Update: *Resolved at commit 58561fc.*

L-03 Inconsistent Allowed Destination Token Logic May Cause Confusion

The `BridgeV2::getAllowedDestinationToken` function, given a destination chain and a source token, returns the corresponding destination token address. When this function returns a non-zero address, it implies that the source token can be bridged to the destination chain and receive an equivalent amount of the destination token.

However, this assumption may not always hold true. A destination chain could be removed from the list of allowed chains without first cleaning its associated destination tokens in the `allowedDestinationToken` mapping. As a result, `getAllowedDestinationToken()` may still return a non-zero address for a chain that is no longer a valid destination. This

inconsistency can lead to confusion or integration errors for external protocols interacting with the `BridgeV2` contract, as they may incorrectly assume that the bridge route is still active.

Consider either removing all the associated destination tokens before removing a destination chain or modifying `getAllowedDestinationToken` function to return a non-zero address if both the destination chain and token are allowed.

Update: *Resolved at commit 6331c79.*

L-04 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `BridgeTokenAdapter.sol`, the `BridgeTokenChanged` event, `initialize` function, `getConsortium` function, `getAssetRouter` function, `isNative` function, `isRedeemsEnabled` function, `getTreasury` function, `getRedeemFee` function, `getFeeDigest` function.
- In `BridgeV2.sol`, the `MSG_VERSION` state variable, `initialize` function, `addDestinationToken` function, `getAllowedDestinationToken` function, `removeDestinationToken` function, `setTokenRateLimits` function, `getTokenRateLimit` function, `setSenderConfig` function, `getSenderConfig` function, `getFee` function, `decodeMsgBody` function, `destinationBridge` function, `mailbox` function.
- In `BridgeTokenPool.sol`, the `getTokenAdapter` state variable
- In `LombardTokenPoolV2.sol`, the `PathSet` event, `PathRemoved` event, `typeAndVersion` state variable, `bridge` state variable, `removePath` function.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved at commit 5f3c625.*

L-05 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In `BridgeTokenAdapter.sol`:
  - In the `changeConsortium` function, the `newVal` parameter is not documented.
  - In the `changeTreasuryAddress` function, the `newValue` parameter is not documented.
  - In the `changeBridgeToken` function, the `newVal` parameter is not documented.
  - In the `getBascule` function, not all return values are documented.
  - In the `spendDeposit` function, the `payload` and `proof` parameters are not documented.
  - In the `transferFrom` function, the `from`, `to`, and `amount` parameters are not documented.
  - In the `burn` function, the `amount` parameter is not documented.
- In `BridgeTokenAdapter.sol`, in the `burn` function, the `from` and `amount` parameters are not documented.
- In `BridgeV2.sol`:
  - In the `setDestinationBridge` function, the `destinationChain` and `destinationBridge_` parameters are not documented.
  - In the `setAllowance` function, the `token`, `tokenAdapter`, and `allow` parameters are not documented.
  - In the `deposit` function, the `destinationChain` and `sender` parameters are not documented.
  - In the `deposit` function, the `destinationChain` parameter is not documented.
- In `BridgeTokenPool.sol`, in the `lockOrBurn` function, the `lockOrBurnIn` parameter and some return values are not documented.
- In `LombardTokenPoolV2.sol`:
  - In the `lockOrBurn` function, the `lockOrBurnIn` parameter and some return values are not documented.
  - In the `releaseOrMint` function, the `releaseOrMintIn` parameter and some return values are not documented.
  - In the `setPath` function: The `remoteChainSelector`, `lChainId`, and `allowedCaller` parameters are not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved at commit 7c2824c and commit c9e55c6.*

Notes & Additional Information

N-01 Redundant Mapping in `BridgeV2`

The `BridgeV2` contract maintains information about which tokens can be bridged and their corresponding destination chains and tokens. To manage this, it uses two mappings that essenctially store the same information:

- `allowedDestinationToken`: It has the destination chain and source token address as the key and the destination token as the value.
- `allowedSourceToken`: It has the destination chain and the destination token address as the key and the source token as the value.

The entries of these mappings are always being set and removed together. However, the `removeDestinationToken` function requires both the source and the destination tokens as inputs, without verifying that the provided pair matches the stored mapping entries. This can lead to incorrect removals or inconsistencies in the contract's state. In addition, the `allowedSourceToken` mapping appears to not be used anywhere in the contracts, making it redundant.

Consider using a single mapping to simplify the logic and make the code clearer and less error-prone. If the contract is already deployed, avoid removing the unused mapping from storage to prevent collisions and not updating it in the functions.

Update: *Resolved at commit 64bb444. The Lombard team stated:*

> *The variable used to have more strict validation.*

N-02 `require` Statement Does Not Check for Any Conditions

In Solidity, using `revert()` is recommended when no conditions are being checked. In `BridgeTokenAdapter.sol`, in line `183`, `require(False)` is used. However, `revert()` would be more appropriate.

Consider replacing all instances of `require(False)` with `revert()` for improved clarity and maintainability of the codebase.

Update: *Resolved at commit 3dd79ca.*

N-03 Misleading Comments

Throughout the codebase, multiple instances of misleading comments were identified:

- In `BridgeV2::setTokenRateLimits`, there is a comment which states that the chain ID is not used anywhere. However, this is contradicted in the very next line, where the rate limit ID is computed by hashing the chain ID.
- The comment in line 301, above the `deposit` function, states that "Deposits and burns tokens from tx sender to be minted on `destinationChain`". However, this is incorrect as the function deposits on behalf of the spender but burns from the transaction sender.

Consider addressing the aforementioned instances misleading comments to avoid confusion during future upgrades or audits.

Update: *Resolved at commit 1086236.*

N-04 Unnecessary Cast

The `address(getTokenAdapter)` cast in the `BridgeTokenPool` contract is unnecessary.

To improve the overall clarity and intent of the codebase, consider removing any unnecessary casts.

Update: *Resolved at commit 39bd27d.*

N-05 Interface `IBridgeV2` Not Advertised in `supportsInterface`

The `BridgeV2` contract inherits the `IBridgeV2` interface. However, the implementation of `supportsInterface` currently only recognizes `IHandler` and `IERC165`. As a result, calls such as `supportInterface(IBridgeV2)` will erroneously return `false`.

Consider updating the `supportsInterface` function to include `IBridgeV2`.

Update: *Resolved at commit 9b12a4f.*

N-06 Unused Imports

Throughout the codebase, multiple instances of unused imports were identified:

- `import {BaseLBTC} from "./BaseLBTC.sol";` in `BridgeTokenAdapter.sol`.
- `import {FeeUtils} from "../libs/FeeUtils.sol";` in `BridgeV2.sol`
- `import {IAdapter} from "./adapters/IAdapter.sol";` in `BridgeV2.sol`
- `import {RateLimits} from "../libs/RateLimits.sol";` in `IBridgeV2.sol`
- `import {IBridge} from "../IBridge.sol";` in `TokenPool.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved at commit c1e10be.*

N-07 Missing Security Contact

Providing a specific security contact (such as an email address or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `BridgeTokenAdapter` contract
- The `BridgeV2` contract
- The `IBridgeV2` interface
- The `LombardTokenPool` contract
- The `BridgeTokenPool` contract
- The `LombardTokenPoolV2` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved at commit 86409f1.*

Conclusion

This audit reviewed the `BridgeV2` contract of the Lombard Protocol, which is responsible for burning tokens upon deposit on the source chain and minting them on the destination chain. The Pool contracts designed for integration with the Chainlink CCIP protocol were also examined, along with the Token Adapter contract that enables compatibility with non-standard token interfaces such as that of the BTC.b token.

During the assessment, several medium- and low-severity issues were identified, primarily related to insufficient input validation. Addressing these findings will further strengthen the protocol's overall security posture.

Overall, the codebase demonstrated high quality and sound design practices. The Lombard team was responsive, collaborative, and professional throughout the engagement, which contributed to a smooth and efficient audit process.