OpenZeppelin | security

Lombard GMP Contracts Audit

LOMBARD

July 17, 2025

Table of Contents

Table of Contents 2

Summary 3

Scope 4

Overview 5

Transport Layer Overview 5

Security Model and Trust Assumptions 6

Privileged Roles 6

Medium Severity 8

M-01 Missing Lower Bound on User-Specified Fees in Minting 8

M-02 AssetRouter setRoute function doesn't use fromChainId parameter 8

M-03 Insufficient Input Validation When Updating the Ratio 9

Low Severity 10

L-01 Incorrect Selector Value 10

L-02 Missing check that Redemption Fee is Less than Amount 11

L-03 Incorrect Enforcement of Total Gas Limit in batchStakeAndBake 11

L-04 Insufficient Error Handling 12

L-05 Potential Conflict with isNative Routes When Changing the Native Token in AssetRouter 12

Notes & Additional Information 13

N-01 Explicit Pause Check Missing 13

N-02 Redundant Conditional Branch 13

N-03 Inaccurate Comments 14

N-04 Gas Optimization Opportunities 14

N-05 Unrestricted Empty reinitialize() Function 15

Conclusion 16

Summary

Type	DeFi
Timeline	From 2025-06-23 To 2025-07-11
Languages	Solidity

Total Issues	13 (9 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	3 (2 resolved)
Low Severity Issues	5 (4 resolved, 1 partially resolved)
Notes & Additional Information	5 (3 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited `lombard-finance/smart-contracts` repository at commit `9e5c6b1` In scope were the following files:

```
contracts
├── LBTC
│   ├── AssetRouter.sol
│   ├── BaseLBTC.sol
│   ├── NativeLBTC.sol
│   ├── StakedLBTC.sol
│   ├── StakedLBTCOracle.sol
│   └── libraries
│       ├── Assert.sol
│       ├── Assets.sol
│       ├── Redeem.sol
│       └── Validation.sol
├── gmp
│   ├── Mailbox.sol
│   └── libs
│       ├── GMPUtils.sol
│       └── MessagePath.sol
├── libs
│   ├── Actions.sol
│   └── LChainId.sol
└── stakeAndBake
    └── StakeAndBake.sol
```

After completion of the audit, the Lombard team introduced several changes to the contracts. The main changes were as follows:

- the previously shared data across all tokens, such as the commision for redeeming for BTC, was made specific to each individual token.
- the `NativeLBTC` storage structure was fixed to prevent collisions after upgrade, since a previous version has been already deployed in a chain.

These changes, included in commit `2356372`, were reviewed as part of the fix review process and no additional issues were identified.

Overview

Lombard General Message Passing (GMP) is the cross-chain transport layer that allows LBTC and other Lombard assets to move between different blockchains while preserving 1 to 1 backing and preventing liquidity fragmentation.

On EVM chains, the audited contracts implement the on-chain half of this transport layer. In particular:

- **Mailbox.sol**: Emits `MessageSent` events for outbound payloads, verifies notary proofs for inbound payloads, and dispatches them to the receiving contract.
- **GMPUtils.sol & MessagePath.sol**: Libraries that provide deterministic path hashing and other helpers so that off-chain notaries can reference the same paths.
- **AssetRouter.sol**: Connect LBTC family tokens and GMP. Handles deposits, redemptions, fee accounting and forwarding payloads to the Mailbox.
- **NativeLBTC.sol & StakedLBTC.sol & StakeAndBake.sol**: ERC-20 wrappers around native BTC or staked LBTC that use the AssetRouter & Mailbox to be able to leverage GMP.

Transport Layer Overview

A message starts when a user calls `Mailbox.sendMessage`, specifying the destination chain and an encoded payload. The Mailbox will check that the path is enabled and the payload is under the max size. It then collects the fees from the user and emits `MessageSent` with the canonical payload hash.

Off-chain notaries then observe the event, verify the chain specific metadata, and sign the payload hash once a quorum is reached.

Then, on the destination chain a relayer submits the payload and its proof to the `Mailbox.deliverAndHandle`. The Mailbox verifies that the proof passes threshold of notary signatures. It then dispatches the payload to the recipient address. In the `LBTC` flow the recipient is the `AssetRouter` which mints or burns `LBTC` as its required.

The entire flow is stateless on the sending side and its prevents relay attacks on the receiving side thanks to the `payloadHash` mapping.

Security Model and Trust Assumptions

- **Honest notaries**: It is assumed that a majority of the notaries from the network behave honestly. A dishonest quorum could forge messages or halt messages by refusing to sign them.
- **Privileged Roles Management**: It is assumed that all privileged roles are held in governance controlled multisigs. A compromised role could take away all the security guarantees on the contracts side. (See Privileged Roles)
- **Reasonable parameters**: `feePerByte` must stay aligned with real relayer costs. Mispricing can either block users (if too high) or expose operators to spam DoS (if too low). Also `dustFeeRate` should be set to a value small enough that doesn't prevent redemptions.

Privileged Roles

Mailbox

- **DEFAULT_ADMIN_ROLE**: Allows granting and revoking roles, managing message paths, pausing the contract and setting the default maximum payload size and fee per byte.
- **PAUSER_ROLE**: Can pause the mailbox, effectively stopping all message sending and receiving.
- **TREASURER_ROLE**: Can withdraw accumulated fees via `withdrawFee()` and rescue stranded ERC-20 tokens.

AssetRouter

- **DEFAULT_ADMIN_ROLE**: Allows managing routes, changing Bascule, Oracle & Mailbox addresses, adjusting token configurations, updating dust fee rate, and toggling global redeem switch.
- **OPERATOR_ROLE**: Can set the maximum mint commission via `setMaxMintCommission()`.

- **CALLER_ROLE**: Authorised caller for the staking token, can call `changeRedeemFee()` and `toggleRedeem()` for token.

NativeLBTC/StakedLBTC

- **DEFAULT_ADMIN_ROLE**: Allows toggling the BTC reedem flag, changing the name, symbol, consortium, treasury, Bascule, AssetRouter, and unpausing the token.
- **PAUSER_ROLE**: Can pause the token, preventing transfers and mints.
- **MINTER_ROLE**: Can call `mint()`, `batchMint()`, `burn()` and `transfer()` functions.
- **CLAIMER_ROLE**: Can execute `mintV1WithFee()/mintWithFee()` and `batchMintV1WithFee()/batchMintWithFee()` functions.
- **OPERATOR_ROLE**: Defined but not referenced in current code (reserved for future use).

StakedLBTCOracle

- **OWNER**: Allows changing the `consortium` address and setting the `maxAheadInterval` parameter,

Medium Severity

M-01 Missing Lower Bound on User-Specified Fees in Minting

The address holding the `CLAIMER_ROLE` can mint `NativeLBTC` by calling `mintV1WithFee` or `batchMintV1WithFee`, providing a `DepositBtcActionV1` payload signed by the validators and a `feePayload`, signed by the recipient. The actual fee applied is the minimum of the `maximumFee`, set by the contract owner, and the `fee` specified by the user in the signed `feePayload`.

While the contract ensures that the fee signed by the user is non-zero, it can be set to arbitrarily small values (e.g. `1 wei`). In such cases, this minimal fee would be applied (even if `maximumFee` is non-zero), rendering the fee mechanism ineffective in preventing denial-of-service (DoS) attacks, its primary intended purpose.

The same issue arises also in the `StakedLBTC` contract.

Consider enforcing a minimum fee threshold (at least when `maximumFee` is not zero) to ensure the minting fee remains an effective mechanism against DoS attacks.

Update: *Acknowledged, not resolved. The Lombard team stated:*

> *The nature of both functions (`mintWithFee` and `batchMintWithFee`) presumes that a certain address with CLAIMER role will trigger mint on behalf of the user and pay gas for this transaction. The fee is meant to compensate claimer's expenses. So it is claimer's responsibility to choose mint payloads with the fee level it considers as acceptable. Also depending on situation claimer might decide to subsidize the mint and let it through despite fee accepted by the user is too low. So I do not think any changes are necessary.*

M-02 AssetRouter setRoute function doesn't use fromChainId parameter

In the `AssetRouter` contract, the `setRoute` function utilizes the `fromToken` and `toChainId` parameters to generate a unique route key.

However the `fromChainId` function parameter is not used at all. There are two possible scenarios, where either the `fromChainId` it is always expected to be the same one, in which then it should not be a function parameter, or where it is expected to support multiple chains.

In the case that it should support different values `fromChainId`, there is a risk in which a identical token address could exist on two different chains, which would cause that the second `setRoute` call would overwrite the first in storage. This is because the `CREATE2` opcode makes creating identical addresses across EVM chains trivial. So if for some reason a token has decided to have the same address across different chains, new routes would overwrite the previous ones causing the router to malfunction.

Consider either removing the `fromChainId` parameter or it to generate the key alongside `fromToken` and `toChainId.

Update: *Resolved in pull request #248.*

M-03 Insufficient Input Validation When Updating the Ratio

In the `StakedLBTCOracle` contract, the `publishNewRatio` function allows setting a new ratio using a valid signed payload. However, the current implementation lacks sufficient input validation, which can lead to incorrect behavior in contracts relying on the reported ratio:

- At initialization the `$.switchTime` and `$.currentRatio` are zero, therefore immediately after initialization `$.prevRatio` will be zero. Since the `ratio()` function returns the `prevRatio` until the `switchTime` is reached, the ratio returned immediately after initialization is zero. This can cause problems to functions like `getRate()`, that will revert due to a division by zero.

- The `_publishNewRatio` function does not enforce that the provided `switchTime` is not in the past.

- The implementation permits multiple signed payloads with the same `switchTime` but different ratios. Because `switchTime` is not required to be strictly increasing, this opens the door to race conditions where conflicting updates are repeatedly submitted and overwrite one another, resulting in fluctuating ratio values.

- The `prevRatio` is not necessarily updated when a new `ratio` and `switchTime` are being submitted. This can result in the system using an outdated prevRatio beyond the intended period, as can be seen in the following scenario:

- Suppose that `prevRatio = r0`, `currentRatio = r1`, `switchTime = s1` and the current `block.timestamp` is `t1 < s1`. The `ratio()` returns `r0`.
- Someone calls `publishNewRatio()` with `ratio = r2` and `switchTime = s2`. After this call we have `prevRatio` is still `r0` as long as the time is `< s2`, even if it is `> s1` the previous switch time, `currentRatio = r2` and `ratio()` returns `r0`.
- If `publishNewRatio` is called again at `t2 < s2`, after the call we have `prevRatio = r0` again, `currentRatio = r3` etc.

Consider initializing the `prevRatio` and apply stricter input validation in `publishNewRatio` to fix the issues described above.

Update: *Resolved in pull request #250 and pull request #253. The Lombard team implemented a fix that initializes the `prevRatio` to 1 and added extra checks that prevent competing ratio updates. It is still possible to submit a `switchRatio` that is in the past, but this is intentional. The Lombard team also stated:*

> *The possibility to submit switch times that are in the past is intentional. The reason for it is the fast the ratio update payload is to be generated by validator consensus (Ledger consensus), that means we have an consensus agreement about what is correct new ratio and correct switch time. At the same time anyone can submit this payload to the oracle smart-contract. In some chains there might be a delay before transaction is submitted due to different reasons. As a result such transaction might be executed after switch time set in payload. This is not something that we expect to happen every time, but still possible and the payload is still legitimate. Consensus is not supposed to issue payload on demand and individually for each chain.*

Low Severity

L-01 Incorrect Selector Value

In the `Assets` library, the `REDEEM_FROM_NATIVE_TOKEN_SELECTOR` has been computed as `bytes4(keccak256("redeemForBTC(bytes32,bytes,uint64)"))`. However, the `amount` argument of `encodeRedeemNativeRequest` is of type `uint256`, not `uint64`. This mismatch can result in incorrect selector computation and encoding.

Consider computing and setting the correct selector value.

Update: *Resolved in pull request #238.*

L-02 Missing check that Redemption Fee is Less than Amount

In `AssetRouter::calcUnstakeRequestAmount`, the redemption fee for the provided token is computed, and then the difference `amount - redeemFee` is passed to `Validation::calcFeeAndDustLimit` without verifying that `redeemFee` is indeed less than `amount`.

Although the latest versions of Solidity do not allow overflows and will revert if `redeemFee` exceeds `amount`, `calcUnstakeRequestAmount` is a `view` function intended to inform the user of the expected amount after fees. Therefore, it would be better to emit an informative error message if the amount cannot cover the fees.

Consider explicitly checking that `redeemFee` is less than `amount` and revert gracefully with a clear error message if the condition is not met.

Update: *Resolved in pull request #238.*

L-03 Incorrect Enforcement of Total Gas Limit in `batchStakeAndBake`

The comment above the `setGasLimit` function suggests that the `gasLimit` represents the total maximum gas that it is allowed to be spent in a `batchStakeAndBake` operation. However, in practice, this limit is applied separately to each individual call of `stakeAndBakeInternal` within the batch, rather than the cumulative gas used by the entire batch.

Consider enforcing the `gasLimit` as a cap to the total gas consumed by the batch. Alternatively, if the per-call application of the gas limit is intended, apply this limit also to the `stakeAndBake` function and update the comment on `setGasLimit` to avoid confusion and better represent the implementation.

Update: *Resolved in pull request #248.*

L-04 Insufficient Error Handling

In the `StakeAndBake` contract, the `batchStakeAndBake` function utilizes a `try/catch` pattern for each individual `stakeAndBakeInternal` call to ensure that a failure in one operation does not revert the entire batch. However, the current implementation catches only error messages emitted by failing revert or require statements, but it will not catch `Panic` or custom errors (e.g. those emitted by the `ERC20PermitUpgradeable` contract during a failed signature verification). This limitation can lead to unhandled exceptions, causing full batch reversion.

Consider including a general catch case in the `try/catch` block to ensure that all error types are appropriately handled.

Update: *Resolved in pull request #248.*

L-05 Potential Conflict with `isNative` Routes When Changing the Native Token in `AssetRouter`

In the `AssetRouter` contract, the owner is responsible for setting allowed routes using the `setRoute` function. The logic ensures that if either the source or the destination chain is the local chain and the corresponding token of the route is marked as `isNative`, then it must match the contract's stored nativeToken. Therefore, it prevents defining multiple routes with different `isNative` tokens for the local chain.

However, if the owner updates the contract's `nativeToken` and then defines a new route for this updated token, the previous route involving the old `nativeToken` will still exist, resulting in a temporary state where two routes for the local chain involve different local `isNative` tokens. This violates the intended uniqueness constraint until the old route is manually removed.

Consider automatically removing the old route when calling `changeNativeToken` function to ensure consistency and prevent possible conflicts.

Update: *Partially resolved in pull request #250. The Lombard team stated:*

> *We added code that removes what can be removed easily. But changing native token itself is a sort of emergency situation that should not happen even once unless we did a mistake when we configure the contracts. So if it happens we will remove all related routes manually. Again, hope this will never happen even once.*

Notes & Additional Information

N-01 Explicit Pause Check Missing

The `StakedLBTC` token is `Pausable`, but the `mint` and `mintWithFee` functions, that can be invoked by anyone providing a valid staking proof, can currently be called even when the contract is paused. This behavior is inconsistent with the `batchMint` and `batchMintWithFee` functions, which correctly block execution when the contract is paused.

Although calls to `mint` and `mintWithFee` will eventually fail if the contract is paused, since the `AssetRouter` will call back the `StakedLBTC`, the `_update` function will be invoked through `_mint`, which revert when the contract is paused. However it is better to include an explicit pause check, similar to that in `batchMint` and `batchMintWithFee`, to improve code clarity and ensure an early revert saving gas.

Consider adding an explicit check in both `mint` and `mintWithFee` to ensure that they can be called only when the contract is not paused.

Update: *Acknowledged, not resolved. The Lombard team stated:*

> *Caller should access AssetRouter directly for all mint-related actions. mint* functions are left in token contracts for backward compatibility and might be removed in future versions. So we prefer to leave these the way they are.*

N-02 Redundant Conditional Branch

In the `AssetRouter:calcUnstakeRequestAmount` function, there is an `if` statement that checks whether the token is native or not and calculates the unstake amount accordingly. However, in both the native and non-native case, the unstake amount calculation logic is identical, rendering the `if` statement redundant.

Consider simplifying the code by removing the unnecessary `if` condition. This will reduce code complexity and slightly decrease the gas consumption.

Update: *Acknowledged, not resolved. The Lombard team stated:*

> *We might need to have different fee for native and non-native tokens since the nature of tokens and their prices are supposed to be different. So we would like to keep the code like it is now to simplify future updates of this kind.*

N-03 Inaccurate Comments

- The `GMPUtils::bytes32ToAddress()` function converts a `bytes32` input to an `address` by discarding the higher 12 bytes. A comment warns the user that many different inputs could be decoded to the same address. However, this warning is misleading. The function includes an explicit check to ensure that the higher 12 bytes are zero and reverts otherwise, guaranteeing a one-to-one mapping from valid inputs to addresses. Consider updating the comment to accurately reflect the function's behavior
- In `Actions.sol` the comment above `struct RatioUpdate` is not related to the structure and should be removed.

Consider fixing these comments to improve clarity and maintainability for users and developers.

Update: *Resolved in pull request #250.*

N-04 Gas Optimization Opportunities

- In the `Mailbox::deliverAndHandle()` function, the `rawPayload` is first decoded to get the `payload` and subsequently `payloadHash` is computed as the `hash()` of `rawPayload`. However, this hash has been already computed during decoding and is available as `payload.id`. Consider using the `payload.id` instead of recalculating the `hash(rawPayload)`, to reduce gas consumption and simplify the code.
- In the `AssetRouter::_AssetRouter_init()` function, there is an initial check to ensure that the provided `mailbox_` is not the zero address. However, this same check is performed again in the `_changeMailbox` function. Consider removing the initial zero address check to eliminate redundancy and reduce gas consumption.

Consider fixing these issues to optimize gas usage.

Update: *Resolved in pull request #250.*

N-05 Unrestricted Empty `reinitialize()` Function

In the `StakedLBTC` contract, there is an empty `reinitialize()` function with a `reinitializer(2)` modifier, but no access control. While this function has no other effect on the contract, calling it will still increment the version (`_initialized` variable) to 2, which could be confusing for users monitoring the contract.

Consider removing the `reinitialize()` function or if you plan to add logic to it in a future update, ensure appropriate access control is in place.

Update: *Resolved at commit 3217859.*

Conclusion

In this audit, we examined the contracts of the GMP Protocol and the changes made to the Lombard contracts to integrate with it. The code was well written, accompanied by an extensive test suite. No critical vulnerabilities were identified, but a few medium and low severity issues were found.

The Lombard team was cooperative, providing all necessary context and responded promptly to our questions, enabling a smooth and effective audit process.