OpenZeppelin | security

Lombard Multipauser Audit

April 9, 2026

Table of Contents

Summary

Type	DeFi
Timeline	From 2026-04-01 To 2026-04-03
Languages	Solidity

Total Issues	11 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	3 (1 resolved)
Notes & Additional Information	7 (5 resolved)
Client Reported Issues	0 (0 resolved)

Scope

OpenZeppelin performed a diff audit of pull request #393 of the lombard-finance/smart-contracts repository at commit 95b424e against commit c1bef7c.

In scope were the following files:

```
contracts
    └── LBTC
        ├── interfaces
        │   └── IStakedLBTC.sol
        ├── BaseLBTC.sol
        ├── NativeLBTC.sol
        └── StakedLBTC.sol
```

System Overview

The in-scope code implements two upgradeable ERC-20 tokens, `NativeLBTC` and `StakedLBTC`, which represent Bitcoin-linked assets within the Lombard system. At a high level, these contracts allow LBTC supply to be created either by privileged minters or by submitting notarized deposit payloads, and allow holders to burn tokens or route redemptions through an external asset-routing component. The shared `BaseLBTC` contract provides core token behavior, EIP-2612 permit support, pausing controls, and the access-control framework used by both token variants.

Minting paths differ by token type. `NativeLBTC` validates deposit payloads directly against an external `INotaryConsortium`, marks payloads as used to prevent replay, optionally confirms deposits through a `IBascule` drawbridge, and then mints tokens. `StakedLBTC` delegates most proof-based minting, fee charging, deposit, and redemption flows to an external `IAssetRouter`, which becomes a central integration point for staking-specific behavior. In both tokens, direct privileged minting and burning are also available through dedicated roles.

This audit scope is centered on a control-plane change that is security-relevant across both tokens. `BaseLBTC` now inherits `AccessControlDefaultAdminRulesUpgradeable`, and the pausing model is split into two mechanisms: a transfer pause (`pauseTransfers`) that blocks all ERC-20 balance updates, and a separate mint/burn pause (`pauseMintBurn`) that blocks only entrypoints guarded with `whenMintBurnAllowed`. As implemented, this lets the system halt supply-changing operations without necessarily freezing ordinary token transfers.

`StakedLBTC` also introduces a migration path from its earlier owner- and address-based permissions into role-based access control. Legacy storage slots for owner, pauser, minters, claimers, and operator are preserved for compatibility, and `migrateToAccessControl` grants new roles only when the supplied addresses match legacy state. This change materially affects the security posture because administrative control is no longer modeled as a single owner plus bespoke allowlists, but as a delayed default admin with explicit roles for pause and mint-related actions.

Security Model and Trust Assumptions

Privileged Roles

- `DEFAULT_ADMIN_ROLE`
- Can unpause transfers through `unpauseTransfers`.
- Can unpause minting and burning through `unpauseMintBurn`.
- Can change core configuration on both tokens, including token name and symbol, consortium address, treasury address, asset router address, and redeem-related parameters.
- Can toggle BTC redeem functionality by calling into the configured `assetRouter`.
- Initializes the role system through `AccessControlDefaultAdminRulesUpgradeable`, which introduces an admin-delay based governance model at initialization and migration time.
- `PAUSER_ROLE`
- Can call `pauseTransfers`, which activates ERC-20 pausing and blocks transfers, minting, and burning because all balance updates flow through `_update`.
- `MINT_BURN_PAUSER_ROLE`
- Can call `pauseMintBurn`, which disables only external entrypoints protected by `whenMintBurnAllowed`.
- This role can stop direct mint, burn, and several redeem flows while leaving ordinary token transfers available.
- `MINTER_ROLE`
- Can mint tokens directly with `mint(address,uint256)` and `batchMint(...)`.
- Can burn tokens from arbitrary addresses with `burn(address,uint256)`.
- `CLAIMER_ROLE`
- Can execute fee-charging mint flows, `mintV1WithFee` and `batchMintV1WithFee` on `NativeLBTC`, and `mintWithFee` and `batchMintWithFee` on `StakedLBTC`.

Trust Assumptions

- `NativeLBTC` trusts the configured `INotaryConsortium` to admit only valid mint payloads.
- A successful proof check is sufficient to authorize proof-based minting, subject to replay protection and any configured Bascule validation.
- `NativeLBTC` optionally trusts the configured `IBascule` contract when its address is non-zero.
- If enabled, mint finalization depends on `validateWithdrawal`. If disabled by setting the address to zero, that additional deposit confirmation layer is removed.
- `StakedLBTC` trusts the configured `IAssetRouter` for core user-facing behavior.
- Proof-based minting, batch minting, mint-with-fee flows, deposits, redemptions, redeem-for-BTC operations, rate reporting, and redeem configuration all depend on the router. A faulty or malicious router could affect message admission, fee handling, or fund movement for this token.
- Both tokens trust privileged configuration changes made by the default admin.
- Changing the consortium, asset router, treasury, or Bascule address can materially alter who can authorize minting, how redemptions are processed, and where fees are sent.
- `StakedLBTC` migration to role-based access control assumes correct migration inputs.
- `migrateToAccessControl` verifies that the supplied default admin matches the legacy owner and that supplied minter and claimer addresses were present in legacy storage. Operationally, migration safety depends on calling this one-time function with complete and correct role lists.

Additional Considerations

The introduced split-pause design changes the operational response model. A transfer pause is now stronger and freezes all token balance changes, while a mint/burn pause is narrower and relies on each external supply-changing path being explicitly protected with `whenMintBurnAllowed`. In the current implementation, these guards are applied to the principal mint, burn, and certain redeem functions in scope, which means users may still be able to transfer tokens during a mint/burn pause but not during a full transfer pause.

Medium Severity

M-01 Unrestricted `migrateToAccessControl` Allows Front-Running of Migration Parameters

`StakedLBTC` includes a one-time migration entry point, `migrateToAccessControl`, for proxies deployed before `AccessControlDefaultAdminRulesUpgradeable`. This migration appears intended to run atomically with the implementation upgrade. However, the repository upgrade tooling defaults `calldata` to `0x` in `upgrade-proxy` while still calling `upgradeAndCall`. As also documented in `DEPLOYMENT.md`, this can leave a post-upgrade window in which the new implementation is active but the migration has not yet been executed.

In that window, any account can call `migrateToAccessControl` first because the function is `external`, uses `reinitializer(3)`, and does not authenticate the caller. Although `initialDefaultAdmin` must match the legacy owner through `_legacyOwner`, the caller still controls `initialAdminDelay` passed to `__AccessControlDefaultAdminRules_init`. The caller also controls the initial role grants, because `MINTER_ROLE` and `CLAIMER_ROLE` are assigned only to addresses explicitly supplied in `minters_` and `claimers_` through the migration loops. An attacker can therefore finalize the migration with attacker-chosen timing parameters and incomplete role lists.

This can leave expected minters or claimers without access and temporarily break role-gated minting paths such as `mint` and `batchMint` and `mintWithFee` and `batchMintWithFee` until the `DEFAULT_ADMIN_ROLE` holder remediates the configuration.

Consider restricting `migrateToAccessControl` to a trusted caller, such as the legacy owner or a designated upgrader, and ensuring that the upgrade and migration are always executed atomically. Consider also bounding `initialAdminDelay` and deriving initial role grants from legacy storage rather than caller-supplied arrays, if that matches the intended migration design.

Update: *Resolved in pull request #408 at commit e721e51.*

Low Severity

L-01 Fee Minting Relies on Two Independent `CLAIMER_ROLE` Checks

`StakedLBTC` fee-minting wrappers, such as `mintWithFee` and `batchMintWithFee`, are protected by `onlyRole(CLAIMER_ROLE)`. These functions forward the request through `_mintWithFee`, which makes a regular external call to `AssetRouter`.

`AssetRouter` separately defines `CLAIMER_ROLE` and applies `onlyRole(CLAIMER_ROLE)` to its fee-minting entry points, including `mintWithFee` and `batchMintWithFee`. Because the router authorizes based on `msg.sender`, the forwarded call succeeds only if `address(StakedLBTC)` has `AssetRouter.CLAIMER_ROLE`, not merely if the external caller has `StakedLBTC.CLAIMER_ROLE`. As a result, the token wrappers are configuration-sensitive and can revert unless `StakedLBTC` is also granted the router role. The split authorization model is also easy to misconfigure. Revoking `StakedLBTC.CLAIMER_ROLE` from an operator does not prevent that operator from calling the router directly if `AssetRouter.CLAIMER_ROLE` remains granted.

Consider consolidating fee-mint authorization into a single layer. For example, either rely on token-side access control for relayers and use a separate router role for trusted calling contracts such as `StakedLBTC`, or clearly document the required cross-contract role assignments if the dual-layer model is intentional.

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

> This is expected behaviour. In fact, `mintWithFee` and `batchMintWithFee` are deprecated and kept only for compatibility. We acknowledge the risk of misconfiguration, but it's a temporary need.

L-02 `StakedLBTC.deposit` Can Burn `NativeLBTC` While `StakedLBTC` Minting Is Paused

`StakedLBTC` exposes `deposit`, which forwards to `AssetRouter`. In `_deposit`, the router burns the configured `nativeToken` and emits a ledger message for a later mint in the same transaction.

However, `StakedLBTC.deposit` is not gated by `paused()` or `whenMintBurnAllowed`. As a result, deposits remain callable during emergency stops intended to halt supply changes or balance updates. The corresponding mint occurs later, when the `Mailbox` calls `handlePayload`, which then calls `mint` on the destination token. For `StakedLBTC`, `mint` is protected by `onlyRole(MINTER_ROLE)` and `whenMintBurnAllowed`, so it can revert while minting is paused. If that happens, the `Mailbox` catches the revert and returns `success = false` in `_handle`, but the `NativeLBTC` burn has already occurred and no user-facing refund or cancellation path exists.

Consider gating `StakedLBTC.deposit` with `whenMintBurnAllowed` and, if aligned with the intended pause model, with transfer pausing as well. Consider also ensuring that deposits remain reversible while minting is paused, for example by delaying the burn until mint execution succeeds or by providing an explicit recovery path for failed mints.

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

> The pause, anyway, is only for emergency cases, which means we must unpause it later and process the failed transactions. We accept this behaviour.

In other words, the team considers this scenario acceptable within their operational model, relying on eventual unpausing and manual or deferred processing of failed mint operations. While this approach introduces a temporary inconsistency and potential user-facing risk (burn without immediate mint), it is treated as an acceptable trade-off given the intended use of the pause mechanism.

L-03 Incomplete Docstrings for Public API Functions

Several public API functions across the codebase have incomplete NatSpec documentation, and some docstrings use inappropriate tags or omit required entries entirely. Examples include:

- In `BaseLBTC.sol`, `mintBurnPaused`, where not all return values are documented.
- In `NativeLBTC.sol`, `getMintFee`, where not all return values are documented.
- In `NativeLBTC.sol`, `decimals`, where not all return values are documented.
- In `NativeLBTC.sol`, `Bascule`, where not all return values are documented.
- In `NativeLBTC.sol`, `burn`, where the `from` parameter is not documented.
- In `StakedLBTC.sol`, `decimals`, where not all return values are documented.
- In `StakedLBTC.sol`, `Bascule`, where not all return values are documented.
- In `StakedLBTC.sol`, `mint`, where not all return values are documented.
- In `StakedLBTC.sol`, `burn`, where the `from` parameter is not documented.

Incomplete or inconsistent NatSpec reduces code readability and can make the intended behavior of external interfaces harder to understand for integrators, reviewers, and maintainers.

Consider thoroughly documenting all functions and events that are part of the public API, including all parameters and return values, and aligning docstrings with the Ethereum Natural Specification Format where applicable.

Update: *Resolved in pull request #401 at commit f480d82. Note that in the referenced PR, the* `whenMintBurnAllowed` *modifier was added to the* `StakedLBTC.deposit` *function.*

Notes & Additional Information

N-01 Unused Role Declaration in `NativeLBTC`

In `NativeLBTC.sol` declares the `OPERATOR_ROLE` constant, an access control role constant that is not used. This unused declaration can create confusion about the intended permission model and expose role-related ABI surface that does not correspond to any implemented behavior.

Consider removing `OPERATOR_ROLE` from `NativeLBTC` if no operator functionality is intended, or integrating it consistently through role grants and `onlyRole` checks if operators are expected to exist, as in `StakedLBTC`.

Update: *Resolved in pull request #402 at commit 664caa1.*

N-02 Missing Security Contact

Including a dedicated security contact, such as an email address or ENS name, in contract NatSpec makes vulnerability disclosure easier. It gives external researchers and third-party library maintainers a clear channel to report issues or share mitigation guidance.

The following contracts do not include a security contact:

- The `BaseLBTC` abstract contract
- The `NativeLBTC` contract
- The `StakedLBTC` contract

- The `IStakedLBTC` interface

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

Update: *Resolved in pull request #403 at commit 57a657f.*

N-03 Unnecessary `unchecked` Blocks Reduce Clarity

Since Solidity `0.8.22`, the compiler automatically removes overflow checks for arithmetic increments in `for` loops. In `StakedLBTC.sol`, the `unchecked { ++i; }` blocks in `migrateToAccessControl` at lines 174 to 176 and 184 to 186 are therefore unnecessary.

These blocks do not provide a benefit on this compiler version and add avoidable noise to the code, which can reduce readability and maintainability.

Consider removing these unnecessary `unchecked` blocks where the compiler already applies the same optimization automatically.

Update: *Resolved in pull request #404 at commit f4da8ce.*

N-04 Unused Imports

Throughout the codebase, there are imports that are unused and could be removed.

- The import `import {EIP1271SignatureUtils} from "../libs/EIP1271SignatureUtils.sol";` in `BaseLBTC.sol`.
- The import `import {BitcoinUtils} from "../libs/BitcoinUtils.sol";` in `StakedLBTC.sol`.
- The import `import {Actions} from "../libs/Actions.sol";` in `StakedLBTC.sol`.
- The import `import {IBaseLBTC} from "./interfaces/IBaseLBTC.sol";` in `StakedLBTC.sol`.
- The import `import {Validation} from "./libraries/Validation.sol";` in `StakedLBTC.sol`.

- The import `import {Redeem} from "./libraries/Redeem.sol";` in `StakedLBTC.sol`.

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

Update: *Resolved in pull request #405 at commit 843afe3.*

N-05 Mint And Burn Pause Enforcement Depends on Entry Point Coverage

The contracts implement a dedicated mint-and-burn pause flag in `BaseLBTC`, enforced through the `whenMintBurnAllowed` modifier on external functions that change supply. This design assumes that every current and future entry point that triggers minting or burning will consistently apply the modifier.

Because the pause check is not enforced at the lowest level of token state updates, internal helpers such as `_batchMint` can mint tokens without validating the pause state. As a result, any supply-changing function that omits the modifier can bypass the pause mechanism. This increases the risk of inconsistent enforcement as the codebase evolves, particularly because there are multiple supply-changing paths.

Consider centralizing mint-and-burn pause enforcement at a single internal state-update point, such as an override of `_update` that checks `from` and `to` to determine whether the operation is a mint or burn and calls `_requireMintBurnAllowed` in those cases. This would make pause enforcement less dependent on manual modifier coverage across entry points.

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

> The problem is that the ideal solution is to overwrite only `_mint` and `_burn`, but they are not virtual. If we apply a check to `_update` we will increase the cost of transfers, which we don't want to do.

In other words, their approach prioritizes minimizing gas overhead on transfers and maintaining separation between mint/burn control and transfer functionality. While they acknowledge the risk of missing the `whenMintBurnAllowed` modifier on future supply-changing functions, they intend to mitigate this through careful auditing and code review of all such entry points. As a result, the issue is accepted as a known trade-off rather than remediated in code.

N-06 Inconsistent Pause State Introspection

`LBTC` exposes two separate pause mechanisms: the `ERC20Pausable` pause, activated via pauseTransfers and observed via `paused`, and a dedicated mint and burn pause, activated via pauseMintBurn and observed via `mintBurnPaused`. Because `ERC20Pausable` blocks all balance updates, it also prevents mint and burn operations when transfers are paused.

As a result, integrations and operators must query both `paused` and `mintBurnPaused` to determine whether minting and burning are currently allowed. This can lead to incorrect assumptions in monitoring, user interfaces, or operational procedures, since the effective mint and burn state depends on the combination of both flags.

Consider exposing a single explicit view method that reports whether minting and burning are currently permitted, such as by combining `paused` and `mintBurnPaused`, and documenting the intended operational distinction between the two pause mechanisms.

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

> The `mintBurnPaused` subject is for `MINTER_ROLE`. We're not expecting that other protocols should monitor this.

N-07 Unused Events

The `ClaimerUpdated`, `PauserRoleTransferred`, and `MinterUpdated` events declared in `IStakedLBTC` are no longer used, as the functions that emitted them have been removed.

Consider removing these unused event declarations to improve code clarity and reduce confusion about the current interface.

Update: *Resolved in pull request #406 at commit 630cab7.*

Conclusion

This audit reviewed a targeted set of changes to Lombard's upgradeable `LBTC` token contracts, focusing on updates to shared administration, pause mechanisms, and the transition of `StakedLBTC` to a role-based access control model.

Overall, the changes introduce a more structured and flexible control framework. Separating transfer pause functionality from `mint` and `burn` controls allows for more precise operational responses, enabling supply-related actions to be restricted without fully interrupting user transfers. However, this approach requires consistent enforcement across all relevant contract entry points, as well as careful handling of the `StakedLBTC` migration to ensure that existing permissions and roles are accurately preserved.

The primary security considerations identified relate to the migration process and the correct configuration of roles under the updated access control system. The introduction of delayed administrative privileges and multiple pausing roles improves governance granularity but also increases reliance on correct role assignment, validation of migration data, and well-defined operational procedures. These factors are particularly relevant in the context of external dependencies such as the `consortium`, `Bascule` integration, and `IAssetRouter`, which continue to influence minting and redemption flows.

The reviewed changes demonstrate progress toward clearer separation of responsibilities and improved operational control, though they also highlight the need for rigorous validation and coordination during deployment and ongoing administration.

OpenZeppelin thanks the Lombard team for its cooperation throughout this audit.

Appendix

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system.

These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.