SHERLOCK

Security Review For Lombard Finance

Collaborative Audit Prepared For: **Lombard Finance**
Lead Security Expert(s): **defsec**
n4nika
Date Audited: **April 1 - April 4, 2026**

Introduction

This audit covers two sets of changes to the Lombard Finance EVM smart-contracts. The first updates the bridge contracts to support CCIP 2.0 by introducing a new message version (v2) that extends the existing fixed-length format with an optional arbitrary payload, adding new deposit overloads that accept this optional message, and updating decodeMsgBody to accept a version range [MSG_VERSION_MIN, MSG_VERSION] for backward compatibility with v1 messages. The second refactors the NativeLBTC and StakedLBTC token contracts to introduce a split-pause mechanism that separately controls ERC-20 transfers via PAUSER_ROLE and mint/burn entry points via a new MINT_BURN_PAUSER_ROLE, with unpause in both cases requiring DEFAULT_ADMIN_ROLE; additionally, StakedLBTC includes a one-time migration from its legacy owner-based access model to AccessControlDefaultAdminRulesUpgradeable.

Scope

Repository: lombard-finance/smart-contracts

Audited Commit: 95b424e9f27d856b916bf72caece2d7ca5aec41b

Final Commit: 6784d656606637eb4cb0a8e1568404b67c5563f1

Files:

- contracts/LBTC/BaseLBTC.sol
- contracts/LBTC/interfaces/IStakedLBTC.sol
- contracts/LBTC/NativeLBTC.sol
- contracts/LBTC/StakedLBTC.sol

Repository: lombard-finance/smart-contracts

Audited Commit: a21bfd8eeb330f202666d09fac45fadc80e5c4ff

Final Commit: 6784d656606637eb4cb0a8e1568404b67c5563f1

Files:

- contracts/bridge/BridgeV2.sol
- contracts/bridge/IBridgeV2.sol

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	2	7

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Stuck funds in BridgeV2 due to overconstrained payload fields [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/7

Summary

`BridgeV2`'s `decodeMsgBody` tries to convert `sender` to an `address` which will revert if the upper 12 bytes are non-zero which is the case for `sui`, `starknet` and `solana` addresses.

Vulnerability Description

After extracting the `bytes32` fields from the payload's `msgBody`, they get converted and returned as addresses:

```
return (
    GMPUtils.bytes32ToAddress(token),
    GMPUtils.bytes32ToAddress(sender),
    GMPUtils.bytes32ToAddress(recipient),
    amount
);
```

This is fine for `token` and `recipient` since the source contract and ledger should normalize these to be of the correct format for the destination chain. The `sender`, however, is passed as-is and is in the source chain's format. This means for `sui`, `starknet` and `solana`, this field will have 32 bytes with the 12 highbytes more likely than not being non-zero.

Since `bytes32ToAddress` reverts if the highbytes are non-zero, those bridged funds will be non-redeemable on the destination EVM chain and be stuck/frozen until the contract is upgraded.

Recommendation

Since the `sender` field is unused anyways, consider returning it as `bytes32` (as-is) or ignore it.

Discussion

555-andrew

Acknowledged. Here is the fix:
https://github.com/lombard-finance/smart-contracts/pull/400

n4nika

confirmed

Issue M-1: Cross chain version mismatch burns tokens on source without minting on destination [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/10

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

When the source chain bridge is upgraded to V2 (MSG_VERSION=2) before the destination chain, a user depositing with `optionalMessage` produces a message body longer than 129 bytes with version byte `2`. On the destination chain still running V1 code, `decodeMsgBody` checks `msgBody.length != MSG_LENGTH` (129) and reverts with `BridgeV2_InvalidMsgBodyLength`.

Vulnerability Detail

The Mailbox marks the payload as `delivered` (preventing re-delivery) but not `handled` (the revert rolls back the handled flag). Tokens are already burned on the source chain. Recovery requires either upgrading the destination bridge and re-submitting the proof, or manual token recovery via `rescueERC20`.

In the CCIP flow via `LombardTokenPoolV2.releaseOrMint`, `deliverAndHandle` returns `executed = false`, causing a revert with `ExecutionError()` that requires manual CCIP intervention.

Impact

Users lose access to burned tokens until destination chain upgrades.

Code Snippet

`contracts/bridge/BridgeV2.sol` – `decodeMsgBody`, `_encodeMsg`

Tool Used

Manual Review

Recommendation

Add a controlled flag `optionalMessagesEnabled` (default `false`) that is only enabled after confirming all destination bridges are upgraded.

Discussion

hashxtree

We acknowledge the risk. We anyway need a coordinated upgrade, because CCIP also needs to be upgraded to start using V2 messages. In other words, there is no reason not to upgrade all the chains.

Issue M-2: `_changeNativeToken` does not reassign `CALLER_ROLE` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/14

Summary

When the `AssetRouter`'s native token is set using `_checkAndSetNativeToken`, the token is granted the `CALLER_ROLE`. However, when it's changed by an admin using `_changeNativeToken`, the original token keeps the role and the new one does not get it.

Vulnerability Description

`_checkAndSetNativeToken` is called by `setRoute` in the `AssetRouter` which is used by the admin to add routes and grants the native token the `CALLER_ROLE`.

```
function _checkAndSetNativeToken(
    AssetRouterStorage storage $,
    bytes32 token
) internal {
    address tokenAddress = GMPUtils.bytes32ToAddress(token);
    grantRole(CALLER_ROLE, tokenAddress);
    if (IBaseLBTC(tokenAddress).isNative()) {
        if ($.nativeToken != address(0) && $.nativeToken != tokenAddress) {
            revert AssetRouter_WrongNativeToken();
        }
        $.nativeToken = tokenAddress;
    }
}
```

However, whenever the native token is changed (very rare to occur if ever) `_changeNativeToken` is used which does neither grant nor revoke the `CALLER_ROLE`:

```
function _changeNativeToken(address newValue) internal {
    AssetRouterStorage storage $ = _getAssetRouterStorage();
    address prevValue = $.nativeToken;
    $.nativeToken = newValue;
    bytes32 key = keccak256(abi.encode(prevValue, LChainId.get()));
    delete $.routes[key];
    emit AssetRouter_NativeTokenChanged(prevValue, newValue);
}
```

This means the previous native token will keep its role and the new one will only receive it once `setRoute` gets called anew by the admin.

Recommendation

Consider handling the `CALLER_ROLE` in `_changeNativeToken`.

Discussion

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/413/changes/fe955e427495f58bf5a5eee3029c14081f7abff9

n4nika

confirmed

Issue L-1: `pauseMintBurn` and `unpauseMintBurn` silently no-op while `pauseTransfers` and `unpauseTransfers` revert [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/8

Summary

The two pause mechanisms on `BaseLBTC` behave inconsistently when the contract is already in the target state.

Vulnerability Detail

`pauseMintBurn` returns silently if `mintBurnPaused` is already true. `unpauseMintBurn` returns silently if it is already false. Neither emits an event nor reverts. By contrast, `pauseTransfers` delegates to OZ's `_pause()` which reverts with `EnforcedPause` if already paused, and `unpauseTransfers` reverts with `ExpectedPause` if not paused.

Impact

An automated system calling `pauseMintBurn` may believe it successfully paused when no state change occurred, with no error signal.

Code Snippet

```
// BaseLBTC.sol lines 133-140 - silent no-op
function pauseMintBurn() external onlyRole(MINT_BURN_PAUSER_ROLE) {
    BaseLBTCStorage storage $ = _getBaseLBTCStorage();
    if ($.mintBurnPaused) { return; } // <-- silent return
    $.mintBurnPaused = true;
    emit MintBurnPauseChanged(true);
}

// BaseLBTC.sol lines 169-171 - reverts
function pauseTransfers() external onlyRole(PAUSER_ROLE) {
    _pause(); // <-- reverts with EnforcedPause if already paused
}
```

Tool Used

Manual Review

Recommendation

Either revert with a descriptive error when already in the target state (matching OZ Pausable).

Discussion

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/413/changes/cb510bb681c8640c98c4491c9bf4db1903f922fe

defsec

Fix is confirmed.

Issue L-2: Fee approval signature reusable across multiple deposit payloads [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/9

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The EIP-712 fee approval digest contains no reference to the specific BTC deposit payload, allowing a `CLAIMER_ROLE` holder to reuse a single user fee signature across N distinct mint payloads in a batch call, charging the user N×fee instead of 1×fee.

Vulnerability Detail

Users sign EIP-712 fee approval messages to authorize the CLAIMER to collect a service fee when processing BTC deposits. The fee approval digest is constructed in `BaseLBTC.getFeeDigest` and verified in `NativeLBTC._mintWithFee` as:

```
bytes32 digest = _hashTypedDataV4(
    keccak256(
        abi.encode(
            Actions.FEE_APPROVAL_EIP712_ACTION,
            block.chainid,
            fee,
            expiry
        )
    )
);
```

The type string is `feeApproval(uint256 chainId, uint256 fee, uint256 expiry)`. The digest contains no reference to the specific BTC deposit payload, no `sha256(mintPayload)`, no nonce, no per-use counter. The only replay protection on the fee signature is the expiry timestamp.

In `batchMintV1WithFee` (NativeLBTC lines 315-340), the CLAIMER controls all four parallel arrays (`mintPayload[]`, `proof[]`, `feePayload[]`, `userSignature[]`). By supplying the same `feePayload` and `userSignature` in positions 0..N-1 while providing N distinct valid `mintPayload` entries (each representing a separate BTC deposit for the same user), the CLAIMER causes N identical fee approval validations to pass each consuming the same signature:

```
for (uint256 i; i < mintPayload.length; ++i) {
    bytes32 payloadHash = sha256(mintPayload[i]);
    if (_getNativeLBTCStorage().usedPayloads[payloadHash]) {
        emit BatchMintSkipped(payloadHash, mintPayload[i]);
        continue; // only the MINT payload is deduplicated
```

```
    }

    _mintWithFee(
        mintPayload[i],
        proof[i],
        feePayload[i],    // feePayload[i] == feePayload[0] in all iterations
        userSignature[i]   // userSignature[i] == userSignature[0] in all iterations
    );
    // same digest → same signature validates every time
}
```

The `usedPayloads` guard deduplicates BTC deposits (preventing double-minting) but does not track fee signature usage. A user who signs one fee approval for one deposit may have that signature reused for every concurrent pending deposit in a batch. Grep over the entire `contracts/` directory found zero occurrences of `usedFee`, `feeNonce`, `usedSignature`, or any equivalent per-use tracking for fee approvals.

Impact

A CLAIMER holding N distinct consortium-signed mint payloads for the same user charges that user N×fee instead of 1×fee.

Code Snippet

```
// BaseLBTC.sol lines 60-75 @audit digest has no payload binding
function getFeeDigest(
    uint256 fee,
    uint256 expiry
) external view virtual returns (bytes32) {
    return
        _hashTypedDataV4(
            keccak256(
                abi.encode(
                    Actions.FEE_APPROVAL_EIP712_ACTION,
                    block.chainid,
                    fee,
                    expiry
                )
            )
        );
}
```

```
// NativeLBTC.sol lines 315-340 @audit batch reuses same fee signature
function batchMintV1WithFee(
    bytes[] calldata mintPayload,
    bytes[] calldata proof,
```

```
    bytes[] calldata feePayload,
    bytes[] calldata userSignature
) external onlyRole(CLAIMER_ROLE) whenMintBurnAllowed {
    Assert.equalLength(mintPayload.length, proof.length);
    Assert.equalLength(mintPayload.length, feePayload.length);
    Assert.equalLength(mintPayload.length, userSignature.length);

    for (uint256 i; i < mintPayload.length; ++i) {
        bytes32 payloadHash = sha256(mintPayload[i]);
        if (_getNativeLBTCStorage().usedPayloads[payloadHash]) {
            emit BatchMintSkipped(payloadHash, mintPayload[i]);
            continue;
        }

        _mintWithFee(
            mintPayload[i],
            proof[i],
            feePayload[i],
            userSignature[i]
        );
    }
}
```

Tool Used

Manual Review

Recommendation

Bind the fee approval digest to the specific BTC deposit payload by including the mint payload hash.

Discussion

hashxtree

This is expected behaviour. User allows us to claim any deposit before the approval expires.

Issue L-3: `_validateAndMint amountToMint > depositAmount` guard always false [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/11

Summary

The guard `if (amountToMint > depositAmount) revert InvalidMintAmount()` inside `NativeLBTC._validateAndMint` is dead code because the sole call site always passes the same value for both parameters.

Vulnerability Detail

`NativeLBTC._validateAndMint` checks whether `amountToMint` exceeds `depositAmount`. However, the only function that calls `_validateAndMint` is `_mintV1`, which passes `action.amount` for both the `amountToMint` and `depositAmount` parameters. Since both arguments originate from the same decoded payload field, the condition `action.amount > action.amount` is always false and the revert can never be reached.

Impact

Dead code.

Code Snippet

https://github.com/lombard-finance/smart-contracts/blob/95b424e9f27d856b916bf72caece2d7ca5aec41b/contracts/LBTC/NativeLBTC.sol#L473

Tool Used

Manual Review

Recommendation

Either make the check meaningful by supporting partial mints where `amountToMint` and `depositAmount` can differ, or remove the dead parameter and guard to reduce complexity.

Discussion

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/413/changes/f066bfb11e56e748a7b4e9197c514249af383c71

defsec

Fix is confirmed on the https://github.com/lombard-finance/smart-contracts/commit/f066bfb11e56e748a7b4e9197c514249af383c71.

Issue L-4: Inconsistent `paused()` checking between single and batch payload-based mint functions on StakedLBTC [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/12

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`StakedLBTC.batchMint(bytes[], bytes[])` and `batchMintWithFee` explicitly check `paused()` while the corresponding single-item functions `mint(bytes, bytes)` and `mintWithFee` do not.

Vulnerability Detail

`StakedLBTC.batchMint(bytes[], bytes[])` includes `if (paused()) revert EnforcedPause()`. The single-item `mint(bytes, bytes)` does not. Both are ultimately protected by the downstream `_update` override from `ERC20PausableUpgradeable`. The same inconsistency exists between `batchMintWithFee` and `mintWithFee`.

Impact

Code inconsistency.

Code Snippet

https://github.com/lombard-finance/smart-contracts/blob/95b424e9f27d856b916bf72caece2d7ca5aec41b/contracts/LBTC/StakedLBTC.sol#L376

Tool Used

Manual Review

Recommendation

Apply consistent modifier patterns to both single and batch variants or remove it.

Discussion

hashxtree

Expected behaviour. Batch methods enforce an extra pause check because it's possible to create an array of mints that will fail only on the last item.

Issue L-5: `OPERATOR_ROLE` is unused in both `NativeLBTC` and `StakedLBTC` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/15

Summary

The `OPERATOR_ROLE`'s functionality does not exist anymore however its definition remains in the code.

Vulnerability Description

In `NativeLBTC`, the `OPERATOR_ROLE` is just unused. In `StakedLBTC`, however, the role even gets migrated to the new system even though it's not used anywhere in the contract.

Recommendation

Consider removing this unused role from the contracts.

Discussion

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/402/changes/664caa1fc1235db402a565dfbd351fa9b7bdcf3f https://github.com/lombard-finance/smart-contracts/commit/983906aaaa9dc1dcd6a03974562b0b6d2bb13c91

n4nika

confirmed

Issue L-6: Dead code in `IStakedLBTC` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/16

Summary

Multiple event definitions whose use was removed in this update remain existing in `IStakedLBTC`.

Vulnerability Description

The following definitions remain in the code but are unused:

```
event ClaimerUpdated(address indexed claimer, bool isClaimer);
event PauserRoleTransferred(
    address indexed previousPauser,
    address indexed newPauser
);
event MinterUpdated(address indexed minter, bool isMinter);
```

Recommendation

Consider removing these to clean up the code.

Discussion

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/406/changes/630cab72220f3f39e7c23a0ca326bfb661625472

n4nika

confirmed

Issue L-7: Missing `whenMintBurnAllowed` on `redeemForBtc`, `deposit` on native and staked LBTC [RESOLVED]

Source: https://github.com/sherlock-audit/2026-04-lombard-lbtc-update/issues/17

Summary

`whenMintBurnAllowed` is applied to all functions which mint or burn somewhere down the callpath. This modifier is currently missing on `StakedLBTC.deposit` and `redeemForBtc` on both `NativeLBTC` and `StakedLBTC`.

Vulnerability Description

`whenMintBurnAllowed` is supposed to pause all minting and burning. It is currently applied to almost all minting/burning functions across native and staked LBTC with the exception of the aforementioned methods.

This is technically fine since all of these mint by calling `mint` or `burn` later which have this modifier, preventing minting/burning while paused. However, this means that such transactions fail late and incur a gas cost. Looking at `StakedLBTC.redeem`, the function has the modifier even though it does not mint/burn directly, meaning it is intended to apply to all such functions.

Recommendation

Consider adding the modifier to the three mentioned functions.

Discussion

hashxtree

Fixed in https://github.com/lombard-finance/smart-contracts/pull/401/changes/f480d82d568bc876967323cd540c7dd2aba16710 https://github.com/lombard-finance/smart-contracts/pull/413/changes/451bcfe1f396b808ea4486a167eb3c230215fd2f

n4nika

Still missing on `StakedLBTC.redeemForBtc`

hashxtree

https://github.com/lombard-finance/smart-contracts/pull/413/changes/33086900b872d68827cb397501b395abf2d33c6d

n4nika

confirmed

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.