LOMBARD

Lombard

Audit Report

MOVEBIT

✉ contact@bitslab.xyz

https://twitter.com/movebit_

Fri Dec 06 2024

Lombard Audit Report

1 Executive Summary

1.1 Project Information

Description	Lombard is on a mission to expand the digital economy by transforming Bitcoin’s utility from a store of value into a productive financial tool with LBTC
Type	DeFi
Auditors	MoveBit
Timeline	Tue Dec 03 2024 - Fri Dec 06 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/lombard-finance/sui-contracts
Commits	928f3cfe0f87dcae422e4fd82da8351465299051 abae24a3e8ad11e3625dc9b9cf75b9a7108244ab

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV	move/lbtc/Move.toml	231820016bb8b291c2c336eb9069a771460614e6
LBT	move/lbtc/sources/lbtc.move	34145647bb2a69a46f2f419046230643c02a76f1
VER	move/lbtc/sources/verify.move	b7fafba755c7607249aadc2a1fd3705964b5eb91
MUL	move/lbtc/sources/multisig.move	4e4b52fb2901f98593e8f71f9214dfeb9df8ae0b
TRE	move/lbtc/sources/treasury.move	6e66c3dbb679dae8754bfec0347f28d6b90fd1ef

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	7	6	1
Informational	1	0	1
Minor	6	6	0
Medium	0	0	0
Major	0	0	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification(Optional)

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Lombard Finance to identify any potential issues and vulnerabilities in the source code of the Lombard smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 7 issues of varying severity, listed below.

ID	Title	Severity	Status
LBT-1	Centralized Storage Used for `ICON_URL`	Informational	Acknowledged
MUL-1	Missing Public Key Length Validation in `address_from_bytes`	Minor	Fixed
TRE-1	Unused `public(package)` Visibility Function	Minor	Fixed
TRE-2	Missing Validation for limit in `new_minter_cap`	Minor	Fixed
TRE-3	Missing Validation for `amount` in `mint_and_transfer`	Minor	Fixed
TUP-1	Incorrect Time Interval Validation in `authorize_upgrade`	Minor	Fixed
TUP-2	Missing Validation for `delay_ms` in `new_timelock`	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Lombard Smart Contract :

Admin

- `add_capability` : Assigns a capability (e.g., AdminCap, MinterCap, PauserCap) to an address.
- `remove_capability` : Removes a capability from an address, ensuring at least one admin remains.
- `enable_global_pause` : Activates a global pause on mint_and_transfer.
- `disable_global_pause` : Deactivates the global pause on mint_and_transfer.
- `mint_and_transfer` : Mints and transfers coins to a specified address within set limits.

User

- `derive_multisig_address` : Derives a multisig address using public keys, weights, and a threshold.
- `is_sender_multisig` : Validates if the transaction sender matches the multisig address derived from specified public keys, weights, and a threshold.
- `ed25519_key_to_address` : Converts an Ed25519 public key into its corresponding address.
- `secp256k1_key_to_address` : Converts a Secp256k1 public key into its corresponding address.
- `secp256r1_key_to_address` : Converts a Secp256r1 public key into its corresponding address.
- `burn` : Burns coins from the sender’s account.
- `list_roles` : Lists roles assigned to a specified address.

4 Findings

LBT-1 Centralized Storage Used for `ICON_URL`

Severity: Informational

Status: Acknowledged

Code Location:

move/lbtc/sources/lbtc.move#20

Descriptions:

The ICON_URL constant currently points to a centralized storage.

```
const ICON_URL: vector<u8> = b"https://www.lombard.finance/lbtc/LBTC.png";
```

Reliance on centralized storage introduces risks such as single points of failure, loss of availability, or tampering with the resource.

Suggestion:

Adopt decentralized storage solutions like IPFS or Arweave to improve reliability and better align with decentralized philosophy.

Resolution:

This is up to Lombard to decide.

MUL-1 Missing Public Key Length Validation in address_from_bytes

Severity: Minor

Status: Fixed

Code Location:

move/lbtc/sources/multisig.move#106

Descriptions:

```
/// Converts a public key to an address based on its type.
fun address_from_bytes(pk: &vector<u8>, flag: u8): address {
    assert!(pk[0] == flag, EInvalidPublicKey);
    address::from_bytes(blake2b256(pk))
}
```

The function address_from_bytes lacks validation to ensure the length of the public key (pk) is appropriate. Without this check, an invalid pk (e.g., one with an incorrect length) may pass through, leading to the generation of meaningless addresses.

Suggestion:

Add a validation step to ensure that the length of pk matches the expected value before proceeding with the computation.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TRE-1 Unused `public(package)` Visibility Function

Severity: Minor

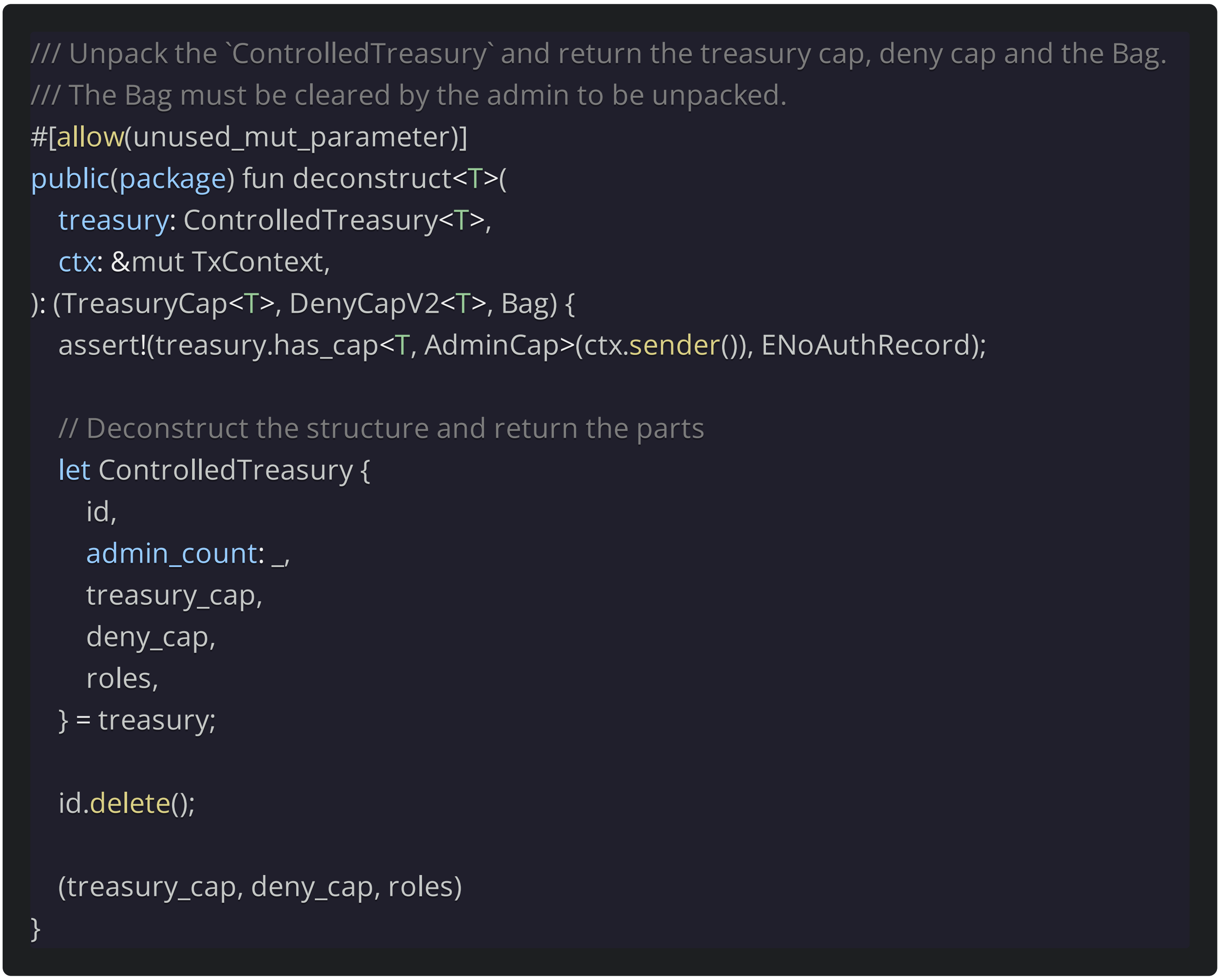
Status: Fixed

Code Location:

move/lbtc/sources/treasury.move#135-153

Descriptions:

The `deconstruct` function, declared with `public(package)` visibility, can only be invoked within the module. However, this function has not been called anywhere throughout the entire project.



```
/// Unpack the `ControlledTreasury` and return the treasury cap, deny cap and the Bag.
/// The Bag must be cleared by the admin to be unpacked.
#[allow(unused_mut_parameter)]
public(package) fun deconstruct<T>(
    treasury: ControlledTreasury<T>,
    ctx: &mut TxContext,
): (TreasuryCap<T>, DenyCapV2<T>, Bag) {
    assert!(treasury.has_cap<T, AdminCap>(ctx.sender()), ENoAuthRecord);

    // Deconstruct the structure and return the parts
    let ControlledTreasury {
        id,
        admin_count: _,
        treasury_cap,
        deny_cap,
        roles,
    } = treasury;

    id.delete();

    (treasury_cap, deny_cap, roles)
}
```

Suggestion:

Remove the function or change its visibility.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TRE-2 Missing Validation for limit in `new_minter_cap`

Severity: Minor

Status: Fixed

Code Location:

move/lbtc/sources/treasury.move#97

Descriptions:

```
/// Create a new `MinterCap` to assign.
public fun new_minter_cap(limit: u64, ctx: &TxContext): MinterCap {
    MinterCap {
        limit,
        epoch: ctx.epoch(),
        left: limit,
    }
}
```

The function `new_minter_cap` does not validate that the limit parameter is greater than zero. If limit is set to zero, it results in the creation of an invalid MinterCap that cannot perform any meaningful operations.

Suggestion:

Add a validation check to ensure that the `limit` parameter is strictly greater than zero.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TRE-3 Missing Validation for `amount` in `mint_and_transfer`

Severity: Minor

Status: Fixed

Code Location:

move/lbtc/sources/treasury.move#219

Descriptions:

The `mint_and_transfer` function does not validate whether the amount parameter is greater than zero. This oversight allows transactions with `amount == 0` to pass all checks and perform unnecessary operations such as emitting events and invoking the mint and transfer logic.

Suggestion:

Add a validation check to ensure that the `amount` parameter is strictly greater than zero before proceeding with any mint or transfer operations.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TUP-1 Incorrect Time Interval Validation in `authorize_upgrade`

Severity: Minor

Status: Fixed

Code Location:

move/timelock_policy/sources/timelock_upgrade.move#59

Descriptions:

```
public fun authorize_upgrade(
    timelock: &mut TimelockCap,
    policy: u8,
    digest: vector<u8>,
    ctx: &mut TxContext,
): UpgradeTicket {
    let epoch_start_time_ms = ctx.epoch_timestamp_ms();

    assert!(
        timelock.last_authorized_time == 0 || epoch_start_time_ms >=
timelock.last_authorized_time + MS_24_HOURS,
        ENotEnoughTimeElapsed,
    );

    timelock.last_authorized_time = epoch_start_time_ms;

    timelock.upgrade_cap.authorize(policy, digest)
}
```

In the `authorize_upgrade` function, the time interval validation uses a fixed value of `MS_24_HOURS` to determine whether sufficient time has elapsed since the last authorization. This logic fails to account for the dynamic delay specified by `timelock.delay_ms`.

Suggestion:

Modify the assertion in the authorize_upgrade function to use `timelock.delay_ms` instead of the hardcoded `MS_24_HOURS`.

Resolution:

This issue has been fixed. The client has adopted our suggestions.

TUP-2 Missing Validation for delay_ms in new_timelock

Severity: Minor

Status: Fixed

Code Location:

move/timelock_policy/sources/timelock_upgrade.move#45

Descriptions:

The new_timelock function does not validate whether the delay_ms parameter is set to an acceptable value.

Suggestion:

Add a validation check in the new_timelock function.

```
assert!(delay_ms == MS_24_HOURS || delay_ms == MS_48_HOURS, EInvalidDelayValue);
```

Resolution:

This issue has been fixed. The client has adopted our suggestions.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

MOVEBIT VERIFIED

MOVEBIT