Echo Lending

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Jan 21 2025

Echo Lending Audit Report

1 Executive Summary

1.1 Project Information

Description	A decentralized finance protocol for lending and collateralization
Type	DeFi
Auditors	MoveBit
Timeline	Fri Jan 10 2025 - Tue Jan 21 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/echo-proto/echo-lending
Commits	5b57d1385788564db9ed9a80a8bdf29bdb2f7f18 c09b879cfd5c442a4939cfccf60ab1aef846fab8

1.2 Files in Scope

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

ID	File	SHA-1 Hash
AWR	echo-core/echo-router/sources/asset_wrapper.move	ced1a0dde49e96460c618d4ad110523bd75b619e
RFA	echo-core/echo-router/sources/router_fa.move	1bee9bb0f7a09640d151e1dcf27df0f80e8c4176
FCO	echo-core/echo-router/sources/fa_coin.move	8c22e1875335cab473a3748631818619e8ac8733

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	2	2	0
Informational	0	0	0
Minor	1	1	0
Medium	1	1	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 Echo Protocol to identify any potential issues and vulnerabilities in the source code of the Echo Lending FA 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 2 issues of varying severity, listed below.

ID	Title	Severity	Status
RFA-1	Repay Does Not Unwrap Remaining Funds	Medium	Fixed
RFA-2	Always Unwrapping Collateral Coin to FA May Lead to Failure	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Echo Lending FA Smart Contract :

Admin

- `map_token` : Maps a new token type to its fungible asset metadata and initializes the coin.

User

- `wrap` : Converts fungible assets (FAs) into the corresponding coin type.
- `unwrap` : Converts coins back into their underlying fungible assets (FAs).
- `supply` : Wraps fungible assets (FA) into the corresponding coin type and supplies the asset to the protocol on behalf of a user.
- `withdraw` : Withdraws assets from the protocol and unwraps the coins back into their underlying FA type.
- `borrow` : Borrows assets from the protocol, converting the borrowed coins into their corresponding FA type.
- `repay` : Wraps fungible assets (FA) into the corresponding coin type and repays the borrowed amount to the protocol.
- `liquidation` : Wraps debt into coins, performs liquidation using the coin representation, and unwraps the collateral and any remaining debt back into their FA types.

4 Findings

RFA-1 Repay Does Not Unwrap Remaining Funds

Severity: Medium

Status: Fixed

Code Location:

echo-core/echo-router/sources/router_fa.move#70-81

Descriptions:

In the `repay` function, the `fa` asset is wrapped in the specified `amount`. However, the actual repayment amount may be less than the wrapped amount. If the user's outstanding debt is less than the specified `amount`, the repayment will be adjusted to the actual debt amount, resulting in unused wrapped `fa`.

For example:

```
if (amount < payback_amount) {
    payback_amount = amount;
}
```

```

    public entry fun repay_simple<CoinType>(
        account: &signer,
        asset: address,
        amount: u256,
        interest_rate_mode: u8,
        on_behalf_of: address,
        dryrun: bool,
    ) {
        //echo_whitelist::assert_white_list_user(signer::address_of(account));
        let _payback_amount = borrow_logic::repay<CoinType>(account, asset, amount,
interest_rate_mode, on_behalf_of, false);

        emit_health_factor(account, dryrun);
    }
```

In the `repay_simple` function, the actual repayment amount (`_payback_amount`) is calculated using the `borrow_logic::repay` function. Since the actual repayment may be less than the wrapped amount, the unused portion of the `fa` asset remains unnecessarily wrapped.

Suggestion:

It is recommended to add logic that unwarps the unused portion of the `fa` asset based on `_payback_amount`.

Resolution:

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

RFA-2 Always Unwrapping Collateral Coin to FA May Lead to Failure

Severity: Minor

Status: Fixed

Code Location:

echo-core/echo-router/sources/router_fa.move#83-118

Descriptions:

In the `liquidation` function, the collateral coin is always unwrapped to `FA`. However, if the `receive_a_token` parameter is set to `true`, it indicates that the liquidator will receive `A token` instead of the collateral. In this scenario, the unwrapping operation will fail, potentially causing the liquidation process to fail entirely.

For example:

```
// Unwrap Collateral Coin to FA
asset_wrapper::unwrap<CoinTypeCollateral>(account, (actual_collateral_to_liquidate as
u64), fa_collateral_asset);
```

If `receive_a_token` is `true`, this unwrapping operation is unnecessary and incompatible with the process, as no collateral will be transferred to the liquidator.

Suggestion:

Modify the logic to perform the unwrapping of the collateral coin to `FA` only when `receive_a_token` is set to `false`.

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