AlphaFi

Preliminary Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Wed May 07 2025

AlphaFi Preliminary Audit Report

1 Executive Summary

1.1 Project Information

Description	The Alpha Lending Protocol is a decentralized lending platform built on Sui that enables users to deposit assets as collateral and borrow other assets against them. The protocol supports both traditional token collateral and liquidity provider (LP) positions (Leverage Yield Farming). Morpho type third party markets are also supported by the protocol.
Type	DeFi
Auditors	MoveBit
Timeline	Thu Apr 03 2025 - Wed May 07 2025
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/AlphaFiTech/alphalend-contracts
Commits	47ddc2a70b50779b2740a363460718bdb3ec4607 5b39d46be506e0247419c6e286345fcc54faf87f

1.2 Files in Scope

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

ID	File	SHA-1 Hash
ALE	alpha_lending/sources/alpha_lending.move	6e167863e231f62b5ca7b1be9e4eb8aa08d1ce23
EVE	alpha_lending/sources/events.move	30d5d7eeb194b4dd9f825f5630559cfe111e90af
POS1	alpha_lending/sources/position.move	68aedc34f5f162eace3286716c83969198098d4d
FLI	alpha_lending/sources/flow_limiter.move	c0f672d0f6b79e8b503c79d9312606c044914d2c
PAR	alpha_lending/sources/partner.move	d3692909fe802a293d933b7519738b1a58d3e70b
REW	alpha_lending/sources/rewards.move	d2ca26dd43b23d1ed8b5b977780e59236a4595e0
ORA	alpha_lending/sources/oracle.move	422c0f330ef603352252a091f77c47f84c8a8105
MAR	alpha_lending/sources/market.move	2f5f5eed831cad3c11dd85faa82c947d33381335
MAT	alphafi_stdlib/sources/math.move	7888ca495bef6260d8969bb8f0e6b4d5599c2f84
ORA1	alphafi_oracle/sources/oracle.move	51d138885404643f7a5bff367a19894a29fd8091

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	14	14	0
Informational	3	3	0
Minor	5	5	0
Medium	2	2	0
Major	4	4	0
Critical	0	0	0
Discussion	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 AlphaFi to identify any potential issues and vulnerabilities in the source code of the AlphaFi 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 14 issues of varying severity, listed below.

ID	Title	Severity	Status
ALE-1	Oracle `Coin_Type` Permission Bypass Vulnerability	Major	Fixed
ALE-2	Inconsistent Token Units in `alpha_lending::liquidate`	Minor	Fixed
ALE-3	Missing Zero Amount Validation	Minor	Fixed
ALE-4	Missing Update After `add_bluefin_lp_collateral`	Informational	Fixed
ALE-5	Typographical Error in Some Field Name	Informational	Fixed
MAR-1	Wrong Implementation of `verify_time_lock`	Major	Fixed
MAR-2	Misplaced Boundary Check in Interest Rate Calculation	Major	Fixed
MAR-3	Incorrect Fee Values in FeeEvent	Minor	Fixed
MAR-4	Missing Validation for Fee Parameters	Minor	Fixed

MAR-5	Missing Length Check for Interest Rate Parameters	Minor	Fixed
MAR-6	Inconsistent Restrictions on Market Creation And Update	Informational	Fixed
POS-1	Inconsistent Token Units in `position::liquidate`	Major	Fixed
REW-1	Incorrect Use of break Prevents Complete Reward Update	Medium	Fixed
REW-2	Incorrect Comparison in Reward Update Logic	Medium	Fixed

3 Participant Process

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

3.1 Core Components

Protocol Management

- Support protocol-wide configuration parameters
- Enable protocol version control and upgrades
- Allow protocol fee collection and distribution
- Manage protocol administrators and permissions

Market Management (market.move)

- Support multiple lending markets with unique configurations:
    - Market-specific fee structures
    - Collateral ratios
    - Interest rate models
    - Deposit limits
    - Liquidation parameters
    - Borrow weight parameters (0.9x to 10.0x)

Position Management (position.move)

- Support two types of collateral:
    - Token collateral (standard tokens)
    - LP collateral (MMT v3 positions)
- Track user positions including:
    - Collateral balances
    - Borrowed amounts
    - Health ratios

    - Liquidation thresholds

Risk Management

- Implement collateral ratio checks:
    - Safe collateral ratio
    - Liquidation threshold
    - Minimum collateral ratio gap (5%)
- Support position liquidation:
    - Close factor: 20%
    - Liquidation bonus
    - Liquidation fees
    - Protocol fees

3.2 Key Features

Lending & Borrowing (alpha_lending.move)

- Deposit assets as collateral
- Borrow against collateral
- Repay borrowed assets
- Withdraw collateral
- Liquidations
- Support isolated markets
- Apply borrow weights to loans

LP Position Management

- Add LP positions as collateral
- Value LP positions using oracle prices
- Update LP position values

- Support LP position liquidations
- Track LP position liquidity

Interest Rate Model (market.move)

- Dynamic interest rates based on utilization
- Support multiple interest rate kinks
- Compound interest calculations
- Interest rate updates

Oracle Integration (oracle.move)

- Price feed integration
- Price staleness checks
- Support for price updates
- EMA price calculations

3.3 Main Functions

User

1. User can collect accrued rewards for a position through `collect_reward()`.
2. User can creates a new position in the protocol through `create_position()` or `create_position_for_partner()`.
3. User can add LP positions as collateral through `add_bluefin_lp_collateral`.
4. User can borrow LP positions through `borrow_bluefin_lp_collateral()`.
5. User can repay LP positions through `return_bluefin_lp_collateral()`.
6. User can update LP positions value through `update_bluefin_lp_collateral_usd_value()`.
7. User can remove LP positions through `remove_bluefin_lp_collateral()`.
8. User can add collateral to a position through `add_collateral()`.
9. User can remove collateral from a position through `remove_collateral()`.

10. User can borrow tokens from a market through `borrow()` .

11. User can repay a borrowed amount from a market through `repay()` .

12. User can liquidate an unhealthy position through `liquidate()` .

13. User can liquidate an LP position through `liquidate_lp_position()` .

14. User can add a reward to a market for either depositors or borrowers through `add_reward()` .

Admin

1. Admin can add a market through `add_market()` or `add_market_native` .

2. Admin can add a coin type to oracle through `add_coin_to_oracle` .

3. Admin can withdraw protocol fees to the protocol treasury through `withdraw_protocol_fee()` .

4. Admin can change the address where protocol fees are sent through `change_protocol_fee_address()` .

5. Admin can create a new partner with fee discount capability through `create_partner()` .

6. Admin can update a partner's fee discountthrough `update_partner_cap_discount()` .

7. Admin can set the price staleness threshold for the oracle through `set_price_staleness_threshold()` .

8. Admin can update the version of the lending protocol through `update_version()` .

4 Findings

ALE-1 Oracle `Coin_Type` Permission Bypass Vulnerability

Severity: Major

Status: Fixed

Code Location:

alpha_lending/sources/alpha_lending.move#500-504

Descriptions:

```
let coin_type = type_name::get<C>();
        if (!oracle::is_coin_in_oracle(&protocol.oracle, coin_type)) {
            let _ = oracle::add_coin_to_oracle(&mut protocol.oracle, coin_type, ctx);
        };
```

The above code is called when add_market is called. It is also called in the add_coin_to_oracle function, but the latter is privilege-controlled, so an attacker can bypass the privilege by calling add_market, which is equivalent to calling add_coin_to_oracle without using LendingProtocolCap.

Suggestion:

Encountering a non-existent coin_type is handled by throwing an error directly. Instead of helping to create it.

Resolution:

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

ALE-2 Inconsistent Token Units in `alpha_lending::liquidate`

Severity: Minor

Status: Fixed

Code Location:

alpha_lending/sources/alpha_lending.move#994-1061

Descriptions:

the `liquidated` value (originally `to_liquidate` in `position::liquidate`) is calculated in USD, but it is incorrectly treated as a `Coin<D>` quantity:

```
  // Continue with liquidation...
  let (xtokens, liquidated, liquidation_bonus, liquidation_fee, to_repay, to_return) =
position.liquidate<B,D>(
      &mut protocol.markets,
      withdraw_market_id,
      borrow_market_id,
      repay_coin,
      &protocol.oracle,
      clock,
      ctx
  );

  let liquidated_amount = math::to_u64(liquidated);
......

  let withdraw_value = math::to_u64(market.get_usd_value(&protocol.oracle,
liquidated_amount));
```

In addition, `liquidation_fee_amount` should be renamed to `liquidation_fee_value` according to the meaning of the variable.

Suggestion:

The fix should ensure consistent unit handling throughout the liquidation process.

Resolution:

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

ALE-3 Missing Zero Amount Validation

Severity: Minor

Status: Fixed

Code Location:

alpha_lending/sources/alpha_lending.move

Descriptions:

The lending protocol lacks zero amount validation in some functions including add_collateral, remove_collateral, borrow, repay...... This allows transactions with zero amounts to be processed normally, potentially leading to unnecessary gas consumption, invalid event logs.

Suggestion:

It is recommended to add explicit zero amount validation at the beginning of each function.

Resolution:

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

ALE-4 Missing Update After `add_bluefin_lp_collateral`

Severity: Informational

Status: Fixed

Code Location:

alpha_lending/sources/alpha_lending.move#585-626

Descriptions:

In the `add_bluefin_lp_collateral` implementation, the call to the `update_bluefin_lp_collateral_usd_value` function is missing. Although this function provides a user interface, it makes sense to update it after `add_bluefin_lp_collateral`.

Suggestion:

Suggest confirming the need to add `update_bluefin_lp_collateral_usd_value` call.

Resolution:

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

ALE-5 Typographical Error in Some Field Name

Severity: Informational

Status: Fixed

Code Location:

alpha_lending/sources/alpha_lending.move

Descriptions:

There is a typographical error in the field name of the LPPositionDepositEvent struct. The field postion_id is misspelled and should be position_id. This error is consistently present in both the struct definition and its usage throughout the codebase.

```
/// Event emitted when an LP position is deposited as collateral
/// * `postion_id` - ID of the position that received the LP collateral
/// * `lp_position_id` - ID of the LP position
/// * `pool_id` - ID of the liquidity pool
/// * `partner_id` - Optional partner ID for fee discounts
public struct LPPositionDepositEvent has copy, drop {
    postion_id: ID,
    lp_position_id: ID,
    pool_id: ID,
    partner_id: Option<ID>
}
```

Suggestion:

It is recommended to use correct and meaningful variable or field names.

Resolution:

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

MAR-1 Wrong Implementation of `verify_time_lock`

Severity: Major

Status: Fixed

Code Location:

alpha_lending/sources/market.move#1045-1051;

alpha_lending/sources/market.move#1153-1156

Descriptions:

The two issues are in the verify_time_lock function:

```
fun verify_time_lock(market: &Market, clock: &Clock) {
    if (!market.config.is_native) {
        assert!(
            clock.timestamp_ms() < (market.config.last_updated + market.config.time_lock),
            ErrTimeLockNotElapsed
        );
    }
}
```

The assertion uses a "less than" (<) comparison, when it should use a "greater than or equal to" (>=) comparison. This reverses the intended functionality of the time lock:

- The current implementation allows parameter changes only during the time lock period
- Once the time lock expires, the assertion will fail, preventing any further updates This is exactly opposite to the intended behavior of a time lock.

And the inconsistency in how time_lock is used should also be addressed. In `update_config_timestamp`, it's treated as an absolute timestamp, while in verify_time_lock it's treated as a duration.

```
fun update_config_timestamp(config: &mut MarketConfig, clock: &Clock) {
    config.last_updated = clock.timestamp_ms();
    config.time_lock = clock.timestamp_ms() + ONEDAY;
}
```

Suggestion:

It is recommended to use correct comparisons and to harmonize the meaning of variables.

Resolution:

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

MAR-2 Misplaced Boundary Check in Interest Rate Calculation

Severity: Major

Status: Fixed

Code Location:

alpha_lending/sources/market.move#709

Descriptions:

The function `calculate_current_interest_rate_apr` calculates the interest rate based on the current utilization rate of a market and a piecewise linear interest rate model defined by `interest_rate_kinks` and `interest_rates`. However, there is a logic flaw in the boundary handling when `i = 1`.

The line:

```
let left_apr = if(i == 0) {0} else {market.config.interest_rates[i-1]};
```

is ineffective because `i` is initialized as 1 and the condition `i == 0` will never be true inside the loop. This suggests the developer may have intended to check if `i - 1 == 0`, i.e., when `i == 1`, to set a `left_apr` of 0. However, as it stands, this check is redundant and misleading. More importantly, this logic assumes that `interest_rates[0]` corresponds to the rate when utilization is 0%. If `i == 1` and `interest_rates[0]` is not 0, then the calculated rate will interpolate from a non-zero left_apr, even though utilization is close to 0%. This breaks the expected piecewise behavior.

Suggestion:

It is recommended to ensure boundary values are clearly and explicitly handled and fit your design.

Resolution:

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

MAR-3 Incorrect Fee Values in FeeEvent

Severity: Minor

Status: Fixed

Code Location:

alpha_lending/sources/market.move#857-861

Descriptions:

In the `add_fee` function, the `FeeEvent` emission contains incorrect fee value calculations. `market_fee` should equal `total_fee - protocal_share`, and `protocol_fee` is `protocal_share`.

```
public (package) fun add_fee<C>(
    market: &mut Market,
    mut xtokens: Balance<XToken<C>>,
) {
    let treasury: &mut Treasury<C> = dynamic_field::borrow_mut(&mut market.id,
TreasuryKey{});

    // Calculate protocol's share of the fee
    let total_fee = balance::value(&xtokens);
    let protocol_share = math::to_u64(math::bps_round_up(
        math::from(total_fee),
        market.config.protocol_fee_share_bps
    ));

    // Split the fee between protocol and market
    let protocol_xtokens = xtokens.split(protocol_share);

    events::emit_event(FeeEvent {
        coin_type: type_name::get<C>(),
        market_fee: total_fee,
        protocol_fee: total_fee - protocol_share
    });

    // Add to respective balances in treasury
    treasury.protocol_fee.join(protocol_xtokens);
    treasury.market_fee.join(xtokens);

}
```

Suggestion:

Suggested changes to calculations or confirming the meaning of fields in events, for example:

```
events::emit_event(FeeEvent {
    coin_type: type_name::get<C>(),
    market_fee: total_fee - protocol_share,  // Market's actual share
    protocol_fee: protocol_share             // Protocol's actual share
});
```

Resolution:

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

MAR-4 Missing Validation for Fee Parameters

Severity: Minor

Status: Fixed

Code Location:

alpha_lending/sources/market.move#266

Descriptions:

The function `create_market` includes multiple parameters that represent fee values in basis points (bps), such as `borrow_fee_bps`, `deposit_fee_bps`, `withdraw_fee_bps`, `liquidation_bonus_bps`, `liquidation_fee_bps`, `spread_fee_bps`, `protocol_fee_share_bps`, and `protocol_spread_fee_share_bps`. However, except for `protocol_fee_share_bps`, these parameters are not validated to ensure they fall within a reasonable range (e.g., 0–10000 bps, representing up to 100%).

Suggestion:

Add assertions to enforce upper bounds (e.g., `<= 10000`) and optionally lower bounds (e.g., `>= 0`) for all fee-related parameters to prevent misconfiguration. Example:

```
assert!(borrow_fee_bps <= 10000, ErrInvalidFee);
assert!(deposit_fee_bps <= 10000, ErrInvalidFee);
assert!(withdraw_fee_bps <= 10000, ErrInvalidFee);
assert!(liquidation_bonus_bps <= 10000, ErrInvalidFee);
assert!(liquidation_fee_bps <= 10000, ErrInvalidFee);
assert!(spread_fee_bps <= 10000, ErrInvalidFee);
assert!(protocol_spread_fee_share_bps <= 10000, ErrInvalidFee);
```

This ensures fee settings are bounded within safe and expected limits, minimizing the risk of abuse or misconfiguration.

Resolution:

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

MAR-5 Missing Length Check for Interest Rate Parameters

Severity: Minor

Status: Fixed

Code Location:

alpha_lending/sources/market.move#1210

Descriptions:

The function `validate_interest_rates` is responsible for verifying that two vectors—`interest_rates` and `interest_rate_kinks`—are correctly aligned and monotonically increasing. However, it does not check whether the lengths of these vectors are non-zero. If either vector is empty, the following issues may arise:

- The function will still pass the initial length equality check if both are zero, which allows the creation of a market with an undefined interest rate model.
- The subsequent assertions that assume the presence of at least one element (`interest_rate_kinks[0] == 0` and `interest_rate_kinks[length - 1] == 100`) will panic at runtime due to index out-of-bounds access.

This could lead to critical contract failures or unhandled errors, especially if this function is used during market initialization or interest rate updates.

Suggestion:

Add an explicit check to ensure the input vectors are not empty before proceeding with validation logic. Example:

```
assert!(vector::length(&interest_rates) > 0, ErrEmptyInterestRates);
assert!(vector::length(&interest_rate_kinks) > 0, ErrEmptyInterestRateKinks);
```

This ensures that the interest rate configuration is meaningful and avoids runtime panics due to empty vectors.

Resolution:

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

MAR-6 Inconsistent Restrictions on Market Creation And Update

Severity: Informational

Status: Fixed

Code Location:

alpha_lending/sources/market.move

Descriptions:

The lending protocol contains inconsistencies in parameter validation between market creation (create_market) and various market update functions. These inconsistencies could allow setting of unsafe or economically unbalanced parameters.

for example: in `update_fee_config`, all fee parameters are properly validated:

```
// In update_fee_config
assert!(
    borrow_fee_bps <= 10000 &&
    deposit_fee_bps <= 10000 &&
    withdraw_fee_bps <= 10000 &&
    spread_fee_bps <= 10000,
    ErrInvalidFeeConfig
);
```

However, In `create_market`, only the `protocol_fee_share_bps` is validated, while other fee parameters are accepted without validation.

Suggestion:

It is recommended that necessary restrictions be added to variables both at the time of creation and at the time of update.

Resolution:

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

POS-1 Inconsistent Token Units in `position::liquidate`

Severity: Major

Status: Fixed

Code Location:

alpha_lending/sources/position.move#766-838

Descriptions:

The problem stems from the following code sequence:

```
// Calculate liquidation fee in USD
let mut liquidation_fee = math::mul(to_repay,
math::from_bps(deposit_market.get_liquidation_fee_bps()));
......

// Later in the code, use the USD value directly as a token quantity
let xtokens_fee = xtokens.split(math::to_u64(liquidation_fee));
```

The liquidation_fee is calculated as a USD value (by multiplying to_repay, which is in USD, by the fee basis points). However, when extracting the fee from the collateral tokens using xtokens.split(), the code incorrectly uses this USD value directly as a token quantity without converting it to the appropriate token amount.

Suggestion:

The fix requires converting the USD-based liquidation_fee to the appropriate token quantity before using it in the split operation.

Resolution:

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

REW-1 Incorrect Use of break Prevents Complete Reward Update

Severity: Medium

Status: Fixed

Code Location:

alpha_lending/sources/rewards.move#174,177

Descriptions:

The loop is intended to iterate through all reward schedules in `self.rewards` and update each one based on current time and last update timestamp. However, the current implementation uses `break` when a reward is not yet active (`reward.start_time >= clock.timestamp_ms()`) or has already ended before the last update (`reward.end_time < self.last_updated`).

Using `break` in these cases causes the loop to terminate entirely, preventing subsequent reward entries from being processed. This introduces a critical logic flaw: if one inactive or outdated reward appears earlier in the list, it will stop all later rewards from being updated, even if they are valid and need to be processed.

This results in skipped rewards and inaccurate reward distributions.

Suggestion:

Replace the `break` statements with `continue` , so that the loop simply skips over the inapplicable reward and continues processing the rest.

Resolution:

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

REW-2 Incorrect Comparison in Reward Update Logic

Severity: Medium

Status: Fixed

Code Location:

alpha_lending/sources/rewards.move#230

Descriptions:

The logic at the condition `if (self.rewards.length() == i)` is used to determine whether to add a new reward entry to the user's reward list. However, this logic is flawed when new reward types are appended in the `RewardDistributor` but not yet present in the user's record. In scenarios where multiple new rewards are introduced, this condition leads to skipping the addition of the final reward.

For example, if `RewardDistributor` has rewards A, B, C, D and `UserRewardDistributor` only has A, B, the expected result after refresh should be A, B, C, D, but the actual result would be A, B, C.

Suggestion:

Replace the condition `self.rewards.length() == i` with `self.rewards.length() <= i` to ensure all new rewards are appended correctly to the user's list.

```
if(self.rewards.length() <= i) {
    // Correct handling of newly introduced reward types
}
```

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.