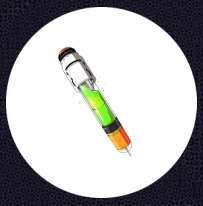

# Epipen Dao
# **Audit Report**

Mon Jan 27 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

# Epipen Dao Audit Report

## 1 Executive Summary

### 1.1 Project Information

| | |
|---|---|
| Description | Epipen DAO is an amazing LRT protocol on BNB Chain that boosts staking potential through an innovative restaking mechanism. Users can stake BNB or slisBNB to earn rewards while supporting ecosystem . Simplified, flexible, and unlocking new DeFi possibilities—Epipen DAO redefines restaking |
| Type | DeFi |
| Auditors | ScaleBit |
| Timeline | Mon Jan 20 2025 - Mon Jan 27 2025 |
| Languages | Solidity |
| Platform | BSC, BNB Greenfield |
| Methods | Architecture Review, Unit Testing, Manual Review |
| Source Code | https://github.com/scalebit/Epipen |
| Commits | 2e844aee82b0ae4088e8ac108781614839da8a72 |

## 1.2 Files in Scope

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

| ID | File | SHA-1 Hash |
|---|---|---|
| VMA | libraries/VaultMath.sol | 6297987ed8d1f389db70a1f33bc6dd126a46a1ad |
| EPI | token/Epipen.sol | 50963a4ff28b8acdca7b9539ee9d68073c1b7d22 |
| MIN | token/Minter.sol | f2c1989729ecf43d56663386493698a5654241d6 |
| ECR | token/EpipenCross.sol | 19e8a8f79d74fd2773c517e27980df4151c213d0 |
| AVA | AssetsVault.sol | 1d844345e97720df22b7f01abbf21e4fcd9c3e20 |
| EVA | EpipenVault.sol | 6ec1a1b84c866136b38c7076705f441f9ef9bea3 |
| STR | strategies/Strategy.sol | 8f5e6799dbe7694ef89414414f3fd179982e40a6 |
| SAG | strategies/SwappingAggregator.sol | 54698221a21291311b91bd8758634819b9cfcac5 |
| LDAOSS | strategies/ListaDAOStakeStategy.sol | cfa0f2d4bda45425ffa0b75f63614d67f4acb220 |
| SCO | strategies/StrategyController.sol | 56ec4bbb4c03629d32e15ddb5583455ffea55939 |

## 1.3 Issue Statistic

| Item | Count | Fixed | Acknowledged |
| --- | --- | --- | --- |
| Total | 7 | 0 | 7 |
| Informational | 0 | 0 | 0 |
| Minor | 1 | 0 | 1 |
| Medium | 4 | 0 | 4 |
| Major | 2 | 0 | 2 |
| Critical | 0 | 0 | 0 |

## 1.4 ScaleBit Audit Breakdown

ScaleBit 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
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

## 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) 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 Epipen to identify any potential issues and vulnerabilities in the source code of the Epipen Dao 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 |
| --- | --- | --- | --- |
| EVA-1 | Missing Slippage Protection for the Deposit | Major | Acknowledged |
| EVA-2 | Front-run `rollToNextRound()` to Gain Profit | Major | Acknowledged |
| EVA-3 | The `actualAmount` may be less than the Amount Users Expected | Medium | Acknowledged |
| EVA-4 | Missing check for withFee >0 | Medium | Acknowledged |
| EVA-5 | Logic Issue with `instantWithdraw` | Medium | Acknowledged |
| EVA-6 | Lack of Events Emit | Minor | Acknowledged |
| SAG-1 | The Value of Slippage Protection is Incorrect | Medium | Acknowledged |

# 3 Participant Process

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

**Admin**

- `addStrategy` : Allows the governance to add a new strategy to the portfolio.
- `destroyStrategy` : Allows the governance to mark a strategy for removal.
- `clearStrategy` : Allows the governance to clear a strategy from the portfolio.
- `updatePortfolioConfig` : Updates the configuration of the portfolio strategies and their allocation ratios.
- `setWithdrawFeeRate` : Sets the withdrawal fee rate, ensuring it does not exceed a predefined maximum.
- `setFeeRecipient` : Updates the recipient address for fees.
- `setRebaseInterval` : Updates the interval between rebase operations, ensuring it does not exceed a minimum limit.
- `setSlisBNBStrategy` : Updates the strategy address for SlisBNB handling.
- `rollToNextRound` : Advances the system to the next round, handling rebalancing of strategies and updating share prices.
- `setEpipenVault` : Updates the address of the Epipen Vault. Ensures the new address is valid and emits an event.
- `setStrategyController` : Updates the address of the Strategy Controller. Ensures the new address is valid and emits an event.
- `setEpipen` : Updates the `epipen` address. Ensures the new address is valid and different from the current one.
- `setVault` : Updates the `vault` address. Ensures the new address is valid and different from the current one.
- `setCap` (Epipen/EpipenCross): Updates the `cap` value. Ensures the new value is different from the current `cap` . Restricted to the contract owner ( `onlyOwner` modifier).

- `setEnable` (Epipen/EpipenCross): Updates the `enable` status. Ensures the new status is different from the current `enable`. Restricted to the contract owner (`onlyOwner` modifier).
- `Epipen/EpipenCross` :Inherited using OFT token standard and public functions, such as `setMsgInspector`, `send`, `setPeer`, etc.
- `setSlippage`: Updates the slippage value for a specified token. Emits the `SetSlippage` event. Restricted to the governance role (`onlyGovernance` modifier).
- `setUniRouter`: Configures a Uniswap V3 pool for a token, including slippage and fee settings. Restricted to the governance role (`onlyGovernance` modifier).
- `setStrategies`: Configures a set of strategies with their respective allocation ratios. Restricted to the vault (`onlyVault` modifier).
- `addStrategy`: Adds a new strategy to the portfolio. Ensures no duplicates. Restricted to the vault (`onlyVault` modifier).
- `destroyStrategy`: Permanently removes a strategy if its value is negligible and allocation is zero. Restricted to the vault (`onlyVault` modifier).
- `clearStrategy`: Clears the data of a strategy without removing it from the list. Restricted to the vault (`onlyVault` modifier).
- `setNewVault`: Updates the vault address. Restricted to the vault (`onlyVault` modifier).
- `setWithdrawQueueParams`: Updates the maximum withdraw queue length and minimum withdraw queue amount. Requires `onlyGovernance`.
- `setRouter`: Configures whether to buy or sell on a decentralized exchange. Requires `onlyGovernance`.
- `setSwap`: Sets a new swapping address. Requires `onlyGovernance`.
- `delegateLista`: Delegates a specified amount of tokens to the `ListaWallet`. Requires `onlyGovernance`.
- `delegateToWallet`: Delegates tokens to a specific wallet. Requires `onlyGovernance`.
- `undelegate`: Undelegates a specified amount of tokens. Requires `onlyGovernance`.
- `undelegateAll`: Undelegates all locked tokens. Requires `onlyGovernance`.

- `swapToToken` : Swaps a specified amount of BNB to tokens. Requires `onlyGovernance` .
- `swapToBNB` : Swaps a specified amount of tokens to BNB. Requires `onlyGovernance` .

**User**

- `deposit` : Allows a user to deposit BNB and receive minted tokens.
- `depositFor` : Allows a user to deposit BNB on behalf of another user and mint tokens for them.
- `depositSlisBNB` : Allows a user to deposit SlisBNB tokens and receive minted tokens.
- `depositSlisBNBFor` : Allows a user to deposit SlisBNB tokens on behalf of another user.
- `requestWithdraw` : Allows a user to request a withdrawal by locking their shares for future rounds.
- `cancelWithdraw` : Cancels a pending withdrawal for the user, returning their locked shares.
- `instantWithdraw` : Enables a user to withdraw assets immediately, subject to available liquidity and fees.
- `currentSharePrice` : Returns the current share price of the assets in the system, accounting for strategies and liquidity.
- `getVaultAvailableAmount` : Provides the current idle and invested amounts in the asset vault.
- `deposit` (EpipenStake): The user transfers the `Epipen` to the current contract.
- `withdraw` (EpipenStake): The user withdraws the Epipen from the current contract.
- `deposit` (AssetsVault): Allows a user to deposit BNB into the contract. Ensures the deposit amount is non-zero.
- `withdraw` (AssetsVault): Allows permitted entities (Epipen Vault or Strategy Controller) to withdraw BNB to a specified address. Emits an event.
- `mint` : Mints a specified amount of tokens to a user address. Restricted to calls from the `vault` (via `onlyVault` modifier).
- `burn` : Burns a specified amount of tokens from a user address. Restricted to calls from the `vault` (via `onlyVault` modifier).

- `swap` : Allows users to swap tokens through the configured Uniswap V3 pool. Supports both token-to-WETH and WETH-to-token swaps.
- `getBestRouter` : Determines the best output for a token swap by querying Uniswap V3 pools.
- `getUniV3Out` : Retrieves the output amount for a swap using Uniswap V3 pools, supporting both sell and buy scenarios.
- `forceWithdraw` : Withdraws funds from strategies to meet a requested amount. Partially utilizes current balance before withdrawing from strategies. Restricted to the vault ( `onlyVault` modifier).
- `rebaseStrategies` : Balances the portfolio by depositing or withdrawing funds to/from strategies based on their allocation ratios. Restricted to the vault ( `onlyVault` modifier).
- `deposit` : Allows the controller to deposit BNB into the strategy. Requires `onlyController` .
- `withdraw` : Allows the controller to withdraw a specified amount. Requires `onlyController` .
- `instantWithdraw` : Allows the controller to withdraw immediately with a specified amount. Requires `onlyController` .
- `clear` : Clears all tokens in the contract. Requires `onlyController` .
- `claimPendingAssets` : Claims pending withdrawal requests.
- `checkPendingAssets` : Retrieves pending asset details, including claimable and pending values.
- `convertBnbToSnBnb` : Converts BNB to SnBNB. Callable by anyone.
- `convertSnBnbToBnb` : Converts SnBNB to BNB. Callable by anyone.
- `claimLaunchPoolReward` : Claims launch pool rewards for a specific epoch and account.

# 4 Findings

## EVA-1 Missing Slippage Protection for the Deposit

Severity: Major

Status: Acknowledged

Code Location:

EpipenVault.sol#101-132

Descriptions:

In the `_depositFor()` function, if `_isSlisBNB == true`, the code calls the function `ISlisBNBStrategy(slisBNBStrategy).convertSnBnbToBnb()` to calculate the BNB amount based on the `silsBNBAmount`, and then calculates the `mintAmount = _amount / sharePrice`.

```
 uint256 silsBNBAmount;
      if(_isSlisBNB == false)
      {
         mintAmount = (_amount * MULTIPLIER) / sharePrice;
         IAssetsVault(assetsVault).deposit{value: address(this).balance}();
      }
      else
      {
         silsBNBAmount = _amount;
          _amount =
ISlisBNBStrategy(slisBNBStrategy).convertSnBnbToBnb(silsBNBAmount);
         mintAmount = (_amount * MULTIPLIER) / sharePrice;
         address SlisBNBAddress = ISlisBNBStrategy(slisBNBStrategy).SLISBNB();
         TransferHelper.safeTransferFrom(SlisBNBAddress, msg.sender, slisBNBStrategy,
silsBNBAmount);
      }
      IMinter(minter).mint(_user, mintAmount);
```

The problem here is that there is no slippage protection. If the share price or the `convertSnBnbToBnb` process is manipulated, the `mintAmount` will not align with the users' expectations.

Suggestion:

It is recommended to allow the user to set the minimum received amount and to implement a check in the code to ensure that the `mintAmount` is greater than or equal to the minimum received amount.

# EVA-2 Front-run `rollToNextRound()` to Gain Profit

Severity: Major

Status: Acknowledged

Code Location:

EpipenVault.sol#268-305

Descriptions:

The `instantWithdraw()` function allows users to input both `amount` and `shares`. If `amount > 0`, the code will use `roundPricePerShare[receipt.withdrawRound]` to calculate the `withdrawAmount`. If `shares > 0`, the code will burn the user's shares and then use `currentSharePrice()` to calculate the user's `ethAmount`.

```
if (_amount != 0) {
        UserReceipt storage receipt = userReceipts[msg.sender];

        if (receipt.withdrawRound != latestRoundID && receipt.withdrawRound != 0) {
          // Withdraw previous round share first
          uint256 withdrawAmount = VaultMath.sharesToAsset(receipt.withdrawShares,
roundPricePerShare[receipt.withdrawRound]);

          epipenMinter.burn(address(this), receipt.withdrawShares);

          withdrawingSharesInPast = withdrawingSharesInPast - receipt.withdrawShares;
          receipt.withdrawShares = 0;
          receipt.withdrawableAmount =  receipt.withdrawableAmount +
withdrawAmount;
          receipt.withdrawRound = 0;
        }

        require( receipt.withdrawableAmount >= _amount, "exceed withdrawable");

        receipt.withdrawableAmount = receipt.withdrawableAmount - _amount;
        withdrawableAmountInPast = withdrawableAmountInPast - _amount;
        actualWithdrawn = _amount;

        emit Withdrawn(msg.sender, _amount, latestRoundID);
      }
```

```
        if (_shares != 0) {
            uint256 sharePrice;

            if (latestRoundID == 0) {
                sharePrice = MULTIPLIER;
            } else {
                uint256 currSharePrice = currentSharePrice();
                uint256 latestSharePrice = roundPricePerShare[latestRoundID - 1];

                sharePrice = latestSharePrice < currSharePrice
                    ? latestSharePrice
                    : currSharePrice;
            }

            uint256 ethAmount = VaultMath.sharesToAsset(_shares, sharePrice);

            epipenMinter.burn(msg.sender, _shares);

            if (ethAmount <= idleAmount) {
                actualWithdrawn = actualWithdrawn + ethAmount;

                emit Withdrawn(msg.sender, ethAmount, latestRoundID);
            }
            else {
                actualWithdrawn = actualWithdrawn + idleAmount;
                ethAmount = ethAmount - idleAmount;

                StrategyController controller = StrategyController(strategyController);
                uint256 actualAmount = controller.forceWithdraw(ethAmount);

                actualWithdrawn = actualWithdrawn + actualAmount;

                emit WithdrawnFromStrategy(msg.sender, ethAmount, actualAmount,
latestRoundID);
            }
        }
```

The `rollToNextRound()` function updates `latestRoundID`, and updates the current round's price, settlementTime, `withdrawingSharesInPast`, `withdrawingSharesInRound`, and `rebaseTime`.

```
 uint256 newSharePrice = currentSharePrice();
      roundPricePerShare[latestRoundID] = previewSharePrice < newSharePrice
        ? previewSharePrice
        : newSharePrice;

      settlementTime[latestRoundID] = block.timestamp;
      latestRoundID = latestRoundID + 1;

      withdrawingSharesInPast = withdrawingSharesInPast +  withdrawingSharesInRound;
      withdrawableAmountInPast =  withdrawableAmountInPast +
VaultMath.sharesToAsset( withdrawingSharesInRound, newSharePrice);
      withdrawingSharesInRound = 0;
      rebaseTime = block.timestamp;
```

The issue here is that a malicious user can front-run the `rollToNextRound()` function, calculate in advance whether `currentSharePrice()` or `roundPricePerShare[latestRoundID]` is higher, and then deposit ahead of time. The user can then choose to either withdraw based on `amount` or `shares` in order to maximize their profit, thus exploiting the system.

Suggestion:

# EVA-3 The `actualAmount` may be less than the Amount Users Expected

Severity: Medium

Status: Acknowledged

Code Location:

EpipenVault.sol#248

Descriptions:

In the `EpipenVault.instantWithdraw()` function, if `ethAmount > idleAmount`, the code will call the `controller.forceWithdraw()` function to withdraw funds from the strategy pool based on the ratios, and then transfer the assets to the `assetsVault`.

```
  function forceWithdraw(uint256 _amount) external onlyVault returns (uint256
actualAmount) {
      uint256 balanceBeforeRepay = address(this).balance;

      if (balanceBeforeRepay >= _amount)
      {
        _repayToVault();
        actualAmount = balanceBeforeRepay;
      }
      else
      {
        actualAmount =_forceWithdraw(_amount - balanceBeforeRepay) +
balanceBeforeRepay;
      }
  }
```

The issue here is that there is precision loss when calculating the withdrawal amount based on the ratios. This may result in the `actualAmount` being less than the amount users expected, leading to potential financial loss for the users.

Suggestion:

# EVA-4 Missing check for withFee >0

Severity: Medium

Status: Acknowledged

Code Location:

EpipenVault.sol#260

Descriptions:

In the `instantWithdraw()` function, if the `withdrawFeeRate != 0`, the code calculates the `withFee` as follows: `withFee = (actualWithdrawn * withdrawFeeRate) / ONE_HUNDRED_PERCENT`.

```
if (withdrawFeeRate != 0) {
    withFee = (actualWithdrawn * withdrawFeeRate) / ONE_HUNDRED_PERCENT;
    aVault.withdraw(feeRecipient, withFee);

    emit FeeCharged(msg.sender, withFee);
}
```

However, the code does not check if `withFee > 0`. If the product of `actualWithdrawn * withdrawFeeRate` is less than `ONE_HUNDRED_PERCENT`, the `withFee` will be 0. This allows users to avoid paying the withdrawal fee.

Suggestion:

It is recommended to check `withFee > 0`.

# EVA-5 Logic Issue with `instantWithdraw`

Severity: Medium

Status: Acknowledged

Code Location:

EpipenVault.sol#134-266

Descriptions:

In the `instantWithdraw()` function, if the amount is not equal to 0, the code verifies that `receipt.withdrawRound != latestRoundID && receipt.withdrawRound != 0`.

```
if (_amount != 0) {
        UserReceipt storage receipt = userReceipts[msg.sender];

        if (receipt.withdrawRound != latestRoundID && receipt.withdrawRound != 0) {
          // Withdraw previous round share first
          uint256 withdrawAmount = VaultMath.sharesToAsset(receipt.withdrawShares,
roundPricePerShare[receipt.withdrawRound]);

          epipenMinter.burn(address(this), receipt.withdrawShares);

          withdrawingSharesInPast = withdrawingSharesInPast - receipt.withdrawShares;
          receipt.withdrawShares = 0;
          receipt.withdrawableAmount =  receipt.withdrawableAmount +
withdrawAmount;
          receipt.withdrawRound = 0;
        }
```

In the `requestWithdraw()` function, if `receipt.withdrawRound != latestRoundID` and `receipt.withdrawRound != 0`, the code will withdraw the previous round's share first and update `receipt.withdrawShares`, `receipt.withdrawableAmount`, and `receipt.withdrawRound`.

```
  // Withdraw previous round share first
        uint256 withdrawAmount =
VaultMath.sharesToAsset(receipt.withdrawShares,roundPricePerShare[receipt.withdrawRo
```

```
        IMinter(minter).burn(address(this), receipt.withdrawShares);

        withdrawingSharesInPast = withdrawingSharesInPast - receipt.withdrawShares;

        receipt.withdrawShares = _shares;
        receipt.withdrawableAmount = receipt.withdrawableAmount + withdrawAmount;
        receipt.withdrawRound = latestRoundID;
```

In the `cancelWithdraw()` function, if `receipt.withdrawShares == 0`, the code will set `receipt.withdrawRound = 0`.

```
if (receipt.withdrawShares == 0) {
        receipt.withdrawRound = 0;
    }
```

If the user calls `requestWithdraw()` to withdraw the previous round's share and then calls `cancelWithdraw()`, the code will set `receipt.withdrawRound = 0`. If the user wants to continue using `instantWithdraw()`, they must deposit a small amount of assets in the current round, then initiate a `requestWithdraw` request, and wait until the next round to call `instantWithdraw()`. Please confirm whether this aligns with the design requirements.

Suggestion:

# EVA-6 Lack of Events Emit

Severity: Minor

Status: Acknowledged

Code Location:

EpipenVault.sol#379;

AssetsVault.sol#20,28,34;

token/Minter.sol#31;

token/Epipen.sol#72;

token/EpipenCross.sol#44;

strategies/SwappingAggregator.sol#149

Descriptions:

The contract lacks appropriate events for monitoring operations, such as `setSlisBNBStrategy`, `withdraw`, `setEpipenVault`, `setStrategyController`, `setEpipen`, `setVault`, `setCap`, `setEnable`, `setUniRouter`, `setNewVault`, `setBufferTime`, `setSwap`, `setRouter` functions which could make it difficult to track sensitive actions or detect potential issues.

Suggestion:

It is recommended to emit events for the function.

# SAG-1 The Value of Slippage Protection is Incorrect

Severity: Medium

Status: Acknowledged

Code Location:

strategies/SwappingAggregator.sol#43-61

Descriptions:

In the swap function, the code redeems tokens on the UNI v3 platform. The `amountOutMinimum` passed to the router contract is used for slippage protection.

```
function swap(address _token, uint256 _amount,bool _isSell )  external payable returns (uint256 amount)
    {
        if (uniV3Pools[_token] == address(0))
        {
            return 0;
        }

        uint256 expected = getBestRouter(_token, _amount, _isSell);

        if (!_isSell)
        {
            require(_amount == msg.value && msg.value != 0 && expected != 0, "wrong value");
        }
        else
        {
            require(msg.value == 0, "ether value should be zero");
        }
        amount = swapOnUniV3(_token, _amount, _isSell);
    }
```

It is calculated as follows: `minReceived = _amount * _slippage / ONE_HUNDRED_PERCENT`. However, this calculation is incorrect. It should call the method `IQuoter(QUOTER).quoteExactInputSingle()` to calculate off-chain and pass the result to the swap function.

Suggestion:

It is recommended to calculate the slippage protection off-chain and pass the result to the function.

# 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.