STATE MIND

Mellow LRT

06-05-2024 - 27-05-2024

Table of contents

1. Project Brief

Title	Description
Client	Mellow
Project name	Mellow LRT
Timeline	06-05-2024 - 27-05-2024
Initial commit	8021ff9affa4710b79d72afbad8026f0ac2a68bd
Final commit	4ceae7e28979a7ecc5ff4b34531e4e548efe67c2

Short Overview

Mellow LRT functions as an LRT constructor, enabling users to deploy and manage their LRTs securely. Key features include robust access control and strategic asset management through modules and strategies. The system centers around the Vault – a pivotal contract for managing digital assets. Users can stake their LST tokens in a set of vaults, each implementing different restaking strategies. These vaults restake users' tokens into various restaking protocols and AVSs, maximizing rewards and optimizing yield farming strategies

Key features:

- Deposit Management: Handles the process of depositing assets into the vault and minting corresponding LP tokens.
- Withdrawal Management: Supports both regular and emergency withdrawals, ensuring asset accessibility and flexibility for users.
- Token and TVL Module Management: Allows the addition and removal of tokens and tvl modules.
- External and Delegate Calls: Supports secure execution of external and delegate calls, with permission checks via validators.

Project Scope

The audit covered the following files:

- Vault.sol
- VaultConfigurator.sol
- ManagedValidator.sol
- DefaultBondStrategy.sol
- DepositWrapper.sol
- ChainlinkOracle.sol
- DefaultAccessControl.sol
- ERC20SwapValidator.sol
- ManagedRatiosOracle.sol
- ERC20SwapModule.sol
- DefaultBondTvlModule.sol
- DefaultBondValidator.sol
- DefaultBondModule.sol
- ManagedTvlModule.sol
- ERC20TvlModule.sol
- DefaultModule.sol
- AllowAllValidator.sol
- ConstantAggregatorV3.sol
- WStethRatiosAggregatorV3.sol

2. Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
🟥 Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
🟧 High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
🟨 Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
🟩 Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	0 (0 fixed, 0 acknowledged)
Medium	4 (4 fixed, 0 acknowledged)
Informational	21 (15 fixed, 6 acknowledged)
Total	25 (19 fixed, 6 acknowledged)

4. Conclusion

During the audit of the codebase, 25 issues were found in total:

- 4 medium severity issues (4 fixed)
- 21 informational severity issues (15 fixed, 6 acknowledged)

The final reviewed commit is 4ceae7e28979a7ecc5ff4b34531e4e548efe67c2

5. Findings report

MEDIUM-01	Incorrect rounding in deposit	Fixed at 35b83a

Description

Line: Vault.sol#L291

In the **Vault.deposit()** function, there is an issue with how rounding is handled when calculating the total value, which can lead to the minting of more LP tokens than required.

```
for (uint256 i = 0; i < tokens.length; i++) {
    uint256 priceX96 = priceOracle.priceX96(address(this), tokens[i]);
    totalValue += totalAmounts[i] == 0
        ? 0
        : FullMath.mulDiv(totalAmounts[i], priceX96, Q96); // <-- should ceil instead of floor
    if (ratiosX96[i] == 0) continue;
    uint256 amount = FullMath.mulDiv(ratioX96, ratiosX96[i], Q96);
    IERC20(tokens[i]).safeTransferFrom(
        msg.sender,
        address(this),
        amount
    );
    actualAmounts[i] = amount;
    depositValue += FullMath.mulDiv(amount, priceX96, Q96);
}
```

The deposit calculation details:

- $\text{depositValue} = \sum_t \lfloor \frac{\text{actualAmount}_t \cdot \text{priceX96}_t}{2^{96}} \rfloor$
- $\text{totalValue} = \sum_t \lfloor \frac{\text{underlyingTvl}_t \cdot \text{priceX96}_t}{2^{96}} \rfloor$
- $\text{lpAmount} = \lfloor \frac{\text{depositValue}}{\text{totalValue}} \cdot \text{totalSupply} \rfloor$

The current implementation rounds down the **totalValue**, which can cause **totalValue** to be lower than its actual value. This makes the ratio **depositValue / totalValue** higher than it should be, potentially resulting in a higher **lpAmount**.

Recommendation

We recommend adjusting the rounding method for **totalValue** calculations to use ceiling instead of flooring.

MEDIUM-02	ChainlinkOracle stale price	Fixed at 35b83a

Description

Line: ChainlinkOracle.sol#L57

The price feeds are updated after the price deviation (usually by 0.5%) or by the time interval of the heartbeat. Chainlink price feeds contracts.

Price feeds have a unique heartbeat interval. Therefore, an interval of 2 days ChainlinkOracle.sol#L10 is not suitable for all price feeds and for some, it's stale data.

Recommendation

We recommend adding the ability to set a unique time interval for every price feed.

MEDIUM-03	Process withdrawals blocking	Fixed at 35b83a

Description

Lines:

- DefaultBondStrategy.sol#L105
- DefaultBondValidator.sol#L33

In the **DefaultBondStrategy.processWithdrawals()** function, the process reverts if there is a bond token on which the **Vault** has a zero balance. This issue occurs because of the check in **DefaultBondValidator.validate()** that doesn't allow zero withdrawals.

```
function validate(address, address, bytes calldata data) external view {
    if (data.length != 0x44) revert InvalidLength();
    bytes4 selector = bytes4(data[:4]);
    if (
        selector == IDefaultBondModule.deposit.selector ||
        selector == IDefaultBondModule.withdraw.selector
    ) {
        (address bond, uint256 amount) = abi.decode(
            data[4:],
            (address, uint256)
        );
        if (amount == 0) revert ZeroAmount(); // <-- revert on zero amount
        if (!isSupportedBond[bond]) revert UnsupportedBond();
    } else revert InvalidSelector();
}
```

This will block withdrawals in some possible cases:

- There is a recently added bond with a zero balance in the **Vault**.
- Admin set **ratioX96** for a bond to zero which is allowed. This means no deposits will be made in this bond, and the **Vault** 's balance in it will be zero.
- Admin made duplicate bonds for a token which is allowed.

In all these cases, it isn't possible to make withdrawals.

Recommendation

We recommend adding a check for zero amount in **DefaultBondStrategy._processWithdrawals()** to prevent from attempting to withdraw zero amount.

MEDIUM-04	Everyone could trigger a deposit of Vault assets	Fixed at 547a88

Description

Line: DefaultBondStrategy.sol#L89

In **DefautlBondStrategy** the withdrawal process consists of three steps:

- 1. Withdrawal of all assets from bonds
- 2. Processing withdrawal requests from users
- 3. Depositing back to bonds rest of the assets

Instant withdrawals are possible only in cases when a user calls withdraw only for himself (e.g. **processWithdrawals([msg.sender])**). In other cases, only admin or operator may trigger DefaultBondStrategy.processWithdrawals() function. Also, a deposit of assets could occur only when users make deposits and **DepositCallback** address is set or when DefaultBondStrategy.depositCallback() function is triggered by admin or operator roles. So, for a user to have the ability to deposit **Vault** assets into bonds, he either should have admin or operator roles or should deposit if **DepositCallback** address is set.

But these conditions for a deposit could be bypassed via instant withdrawals. Everyone could call the function DefaultBondStrategy.processWithdrawals() with only one **msg.sender** in the **users** array. Check for admin or operator roles will be bypassed, and then withdraw from bonds will happen. Processing of the request in Vault.processWithdrawals() won't revert even if a user is not in **_pendingWithdrawers** and hasn't submitted a withdrawal request. Then the actual deposit of Vault's assets will happen.

Recommendation

We recommend checking if **msg.sender** is in Vault's **_pendingWithdrawers** before processing instant withdrawals.

INFORMATIONAL-01	Incorrect Vault.emergencyWithdraw() logic with Vault.baseTvl().amounts == 0	Fixed at 35b83a

Description

Line: Vault.sol#L364

The **Vault.emergencyWithdraw()** function allows users to withdraw their liquidity and receive tokens back. The classic scenario implies that the user specifies the minimum value (**minAmounts**) for each token received. If the user receives a smaller amount of the token than he wishes, an immediate revert should occur. However, this implementation does not take into account the zero value of the **Vault.baseTvl().amounts** token and skips it.

Impact: The logic of the function is such that the user must be sure that he has received the required number of tokens specified in **minAmounts**. Skipping the zero value of the token leads to the fact that the user may not receive the desired amount of the token if he passed a non-zero value of **minAmounts** on it.

Recommendation

We recommend adding a separate check like this:

```
if (amounts[i] == 0 && minAmounts[i] > 0) revert InsufficientAmount();
```

INFORMATIONAL-02	Potential DOS by initial deposit	Acknowledged

Description

Line: Vault.sol#L321

In the **Vault._processLpAmount()** function, the amount of LP tokens to mint during the first deposit is set by the **minLpAmount** parameter. This creates a risk where an operator can make the initial deposit and set **minLpAmount** to **configurator.maximalTotalSupply()**, thus blocking further operations on the **Vault** contract.

```
function _calculateLpAmount(
    address to,
    uint256 depositValue,
    uint256 totalValue,
    uint256 minLpAmount
) private returns (uint256 lpAmount) {
    uint256 totalSupply = totalSupply();
    if (totalSupply == 0) {
        // scenario for initial deposit
        _requireAtLeastOperator();
        lpAmount = minLpAmount; // <-- operator can set it too big and send Vault to DoS
        if (lpAmount == 0) revert ValueZero();
        if (to != address(this)) revert Forbidden();
    } else {
        lpAmount = FullMath.mulDiv(depositValue, totalSupply, totalValue);
        if (lpAmount < minLpAmount) revert InsufficientLpAmount();
        if (to == address(0)) revert AddressZero();
    }

    if (lpAmount + totalSupply > configurator.maximalTotalSupply())
        revert LimitOverflow();
    _mint(to, lpAmount);
}
```

Recommendation

We recommend minting a **lpAmount** related to **depositValue**, such as **depositValue** itself. Additionally, we recommend adding a minimum liquidity check for the first deposit to prevent inaccuracies with small amounts.

Client's comments

Calculating the initial lpAmount based on depositValue leads to the requirement that the value of depositValue must be calculated with the same decimals as the lp token of vault (18). We also consider the initial deposit to be a trusted operation, so we believe there should be no code changes here.

INFORMATIONAL-03	ChainlinkOracle incorrect price and sanity checks	Fixed at 35b83a

Description

Line: ChainlinkOracle.sol#L56 - unsafe cast negative price leads to an incorrect value of **answer**. It may lead to incorrect calculations of the **priceX96**, calculations of **depositVault**, and **totalValue**. As a result of incorrect LP accounting or revert transaction without error message.
There are also some sanity checks missing.

Recommendation

We recommended performing a sanity check to verify that the fetched data is valid.

```
(uint80 roundId,
int256 signedAnswer,
,
lastTimestamp,
uint80 answeredInRound) = IAggregatorV3(aggregatorV3).latestRoundData();
if (
   roundId == 0 &&
   signedAnswer < 0 &&
   lastTimestamp == 0
) {
  revert InvalidOracleData();
}
```

INFORMATIONAL-04	Vault.pendingWithdrawers() DOS	Fixed at 35b83a

Description

Line: Vault.sol#L46
An attacker can create many 1-wei deposits with immediate withdrawal (in a single transaction) from different addresses. **Vault.pendingWithdrawers()** will be reverted for calls from external contracts.
OpenZeppelin docs warning.

Recommendation

We recommend adding the ability to request N items.

INFORMATIONAL-05	Adding an existing module emits an unnecessary event	Fixed at 35b83a

Description

Line: Vault.sol#L206
Adding an existing module using a **Vault.addTvlModule()** function does nothing, but emits an unnecessary event.

Recommendation

We recommend adding a check.

```
if (!_tvlModules.add(module)) {
  revert AlreadyAdded();
}
```

INFORMATIONAL-06	Setting incorrect **maximalTotalSupply** value could lead to locking deposits	Fixed at 35b83a

Description

Line: VaultConfigurator.sol#L178

maximalTotalSupply could be changed via **VaultConfigurator**. Its value can either decrease or increase. In case of decreasing if a new value is less than the current total supply, it will block further deposits.

Recommendation

We recommend adding a check when staging a new value for **maximalTotalSupply**. In case of decreasing, the new value could be less than the total supply only when deposits have been previously locked.

INFORMATIONAL-07	Using regular mapping instead of **EnumerableSet.AddressSet** for **_underlyingTokensSet**	Fixed at 35b83a

Description

Line: Vault.sol#L28

_underlyingTokensSet is an **EnumerableSet.AddressSet**, but it is used only to check the presence or absence of a certain address. It can be replaced with regular mapping, because functions connected to an inner array (e.g. **length()**, **values**) are not used.

Recommendation

We recommend using regular mapping instead of **EnumerableSet.AddressSet** to optimize calculations during checks and not to allocate an additional slot for an array.

INFORMATIONAL-08	Gas Optimisations	Fixed at 35b83a

Description

Lines:

- Vault.sol#L69 – Values for variables **amount** and **token** are extracted in every iteration of the inner loop. They can be calculated once outside it because they only depend on the **ITvlModule.Data** from the **tvl_** array.
- Vault.sol#L67 – memory for **data** variable is allocated every iteration. It can be allocated once and only the values of the variable will change.

Recommendation

We recommend extracting values for **amount** and **token** outside the inner loop and allocating memory for the **data** variable only once.

```
...

ITvlModule.Data memory data;
for (uint256 i = 0; i < tvl_.length; i++) {
    data = tvl_[i];
    (uint256 amount, address token) = isUnderlying
        ? (data.underlyingAmount, data.underlyingToken)
        : (data.amount, data.token);

    for (uint256 j = 0; j < tokens.length; j++) {
        if (token != tokens[j]) continue;
        (data.isDebt ? negativeAmounts : amounts)[j] += amount;
        break;
    }
}

...
```

INFORMATIONAL-09	Check for the initial configuration of a TVL module when adding it	Acknowledged

Description

Line: Vault.sol#L201

When adding a new TVL module data for certain Vault is extracted and checked against internal **_underlyingTokensSet**. But in case the data is empty or the TVL module is not yet configured, the loop will be skipped and the module will be added. This allows you to bypass checks and add in the future tokens that are not in **_underlyingTokensSet** of the **Vault**.

Recommendation

We recommend checking if data in the TVL is not empty and only adding modules that have been already configured.

Client's comments

These operations can be performed on behalf of the admin of the vault, so we believe that this is an acceptable risk.

INFORMATIONAL-10	Call to **IWithdrawalCallback.withdrawalCallback()** when zero shares are burnt	Fixed at 35b83a

Description

Line: Vault.sol#L541
In function Vault.processWithdrawals() it is possible to provide an empty array of users or all requests couldn't be withdrawn. In such cases, no shares will be burnt, but anyway **withdrawal callback** will be called (even actual withdrawal didn't happen).

Recommendation

We recommend calling **IWithdrawalCallback.withdrawalCallback()** only when some share will be burnt, or processing such cases inside the callback contract.

INFORMATIONAL-11	Code refactor	Fixed at 35b83a

Description

Lines:

- Vault.sol#L386 – this check should be made in external functions, which calls private/internal function _cancelWithdrawalRequest(). This allows you to divide the logic and avoid double checks.
- Vault.sol#L527 – balances are updated after actual transfers, in case of potential reentrancy this could lead to incorrect calculations.

Recommendation

We recommend moving the check **if (!_pendingWithdrawers.contains(sender))** out of the private function and making it in external ones. Also at Vault.sol#L527 it is better to update balances before transfers.

INFORMATIONAL-12	Extra check in **DefaultBondStrategy._deposit()**	Fixed at 598fb5

Description

Lines:

- DefaultBondStrategy.sol#L52
- ERC20TvlModule.sol#L16

The **DefaultBondStrategy._deposit()** function uses ERC20 tokens for further deposit into the vault. The list of tokens is imported from **ERC20TvlModule** so each of these tokens is underlying. Therefore, there is no need to verify that the token is underlying.

Recommendation

We recommend removing this unnecessary check if expected to use only the **ERC20TvlModule** implementation for **DefaultBondStrategy.erc20TvlModule**.

INFORMATIONAL-13	Confusing **DepositWrapper._ethToSteth()** name	Fixed at 35b83a

Description

Line: DepositWrapper.sol#L30
The name of the **DepositWrapper._ethToSteth()** function assumes that it converts the incoming native token to **stETH**. However, the function returns the number of **wstETH** tokens converted from ETH.

Recommendation

We recommend changing the name of a function like **_ethToWsteth()**.

INFORMATIONAL-14	**DefaultBondStrategy** doesn't allow configuring modules	Acknowledged

Description

Line: DefaultBondStrategy.sol#L19
VaultConfigurator allows the admin to configure the allowed list of modules via **isDelegateModuleApproved** mapping. To make **DefaultBondStrategy** work, **DefaultBondStrategy.bondModule** must be in **isDelegateModuleApproved**. In the current implementation, **DefaultBondStrategy.bondModule** is immutable.
Suppose the administrator needs to replace the module in an emergency. In that case, the admin must deploy the strategy again, revoke authorization from the previous strategy, and issue authorization to the new strategy. A more convenient way is to replace the desired module in strategy.

Recommendation

We recommend adding a configuration option for strategy modules.

Client's comments

> The idea here is that Strategy is a contract with corresponding immutable delegateModules. Thus, strategy contracts are more reliable

INFORMATIONAL-15	**ERC20SwapValidator** inflexible setting	Acknowledged

Description

Line: ERC20SwapValidator.sol#L13
ERC20SwapValidator allows the administrator to configure 2 sets: tokens and routers. However, the list of supported tokens in routers may differ depending on DEXs pools' availability. Also, some routers may support only regular ERC20 tokens, while others may support rebase, transfer fee tokens, or other types.

Recommendation

We recommend adding the ability to configure supported tokens for each router individually.

Client's comments

> Formally, we check here that valid tokens are exchanged across valid routers. If the swap fails in the router, then revert SwapFailed() occurs in the module itself

INFORMATIONAL-16	Due to the absence of wstETH mainnet oracle, it is impossible to configure a multi-asset vault properly	Fixed at 35b83a

Description

Vault uses price oracle to estimate **totalValue** of the underlying tokens.

```
IPriceOracle priceOracle = IPriceOracle(configurator.priceOracle());
```

In case when we have more than one token in vault, price of the underlying token will be calculated relative to the baseToken.

```
priceX96_ = FullMath.mulDiv(
    tokenPrice * 10 ** baseDecimals,
    Q96,
    baseTokenPrice * 10 ** decimals
);
```

However, the **ChainlinkOracle** contract only supports single asset price data. This makes it completely incompatible with **wstETH** because chainlink doesn't have a **wstETH** oracle on mainnet. This means it is impossible to create a multi-asset vault with **wstETH** because you can't estimate the token values accurately. For example, the admin would like to create a 50:50 **wstETH/rETH** vault to use various liquid staking protocols.

Recommendation

We recommend adding custom oracles for tokens that do not have a chainlink data feed.
For example, there is an oracle specifically for wstETH utilizing the stETH oracle and its current exchange rate.

```
contract WstEthOracle {
  address internal wsteth;
  AggregatorV3Interface internal aggregatorV3;
  ...
  function latestRoundData()
    external
    view
    returns (
      uint80 roundId,
      int256 answer,
      uint256 startedAt,
      uint256 updatedAt,
      uint80 answeredInRound
    ) {
      (, int256 signedAnswer, , uint256 lastTimestamp, ) = IAggregatorV3(aggregatorV3).latestRoundData();
      uint256 stEthPrice = SafeCast.toUint256(signedAnswer); // get Chainlink price for stETH
      uint256 stEthPerToken = IWETH(wsteth).stEthPerToken();
      uint256 wstEthPrice = (stEthPrice * stEthPerToken) / 1e18;
      return wstEthPrice;
    }
}
```

INFORMATIONAL-17	Gas optimisation: excessive check	Fixed at 35b83a

Description

Line: Vault.sol#L444

```
if (
    request.tokensHash != s.tokensHash ||
    lpAmount == 0 ||
    request.deadline < s.timestamp
)
```

The lpAmount check will always be bypassed as **Vault.registerWithdrawal()** does not register requests with zero **lpAmount**.

```
if (lpAmount == 0) revert ValueZero();
```

In case the **request** does not exist, **request.tokensHash** will be **""** and revert as **s.tokensHash** is not empty.

Recommendation

We recommend removing excessive check.

```
if (
    request.tokensHash != s.tokensHash ||
-   lpAmount == 0 ||
    request.deadline < s.timestamp
)
```

INFORMATIONAL-18	Lack of reentrancy lock in Vault.delegateCall()	Acknowledged

Description

Lines:

- Vault.sol#L237
- ERC20SwapModule.sol#L23

The **Vault.delegateCall()** function lacks reentrancy protection, which creates a potential reentrancy scenario when it is used to call the **ERC20SwapModule**. In the **ERC20SwapModule.swap()** function, it is possible to call any DEX approved in the **ERC20SwapValidator** contract. There are no restrictions preventing calls to third-party addresses in this DEX contract.
The dangerous scenario arises if the **Vault.deposit()** function is called during the DEX swap. This will break **Vault**'s math and allows the caller to mint an excessive amount of LP tokens.
Here's a detailed scenario that may be possible:

- The operator calls the **ERC20SwapModule.swap()** function through the **Vault.delegateCall()**, which makes a call to a DEX contract.
- DEX contract contains a callback after **tokenIn** transfer from the **Vault** and before **tokenOut** transfer.
- Inside the callback, there is a call to the **Vault.deposit()** function.
- The deposit's **totalValue** calculation is manipulated because **tokenIn** has been reduced. This results in a lower **totalValue**, making the ratio **depositValue / totalValue** higher.
- The higher ratio leads to the minting of more LP tokens than needed, which creates a profit for the attacker.
- The DEX may also transfer a lower amount of **tokenOut** because the amount of **tokenOut** has already been increased during the deposit.

Recommendation

We recommend not using routers with callbacks called during the exchange to register in the whitelist or modifying the re-entry protection mechanism for the **Vault.delegateCall()** function.

Client's comments

We mean using only trusted swap routers here.

INFORMATIONAL-19	Safety checks when changing params in TVL modules	Fixed at 598fb5

Description

Lines:

- ManagedTvlModule.sol#L12
- DefaultBondTvlModule.sol#L12

When adding a new TVL module in a Vault, its data is passed through checks at Vault.sol#L202-205. **ManagedTvlModule** and **DefaultBondTvlModule** modules have admin functions to change params and set new data, even after addition to the Vault. New data is not checked against **underlyingTokens** of the Vault.

Recommendation

We recommend adding checks for new data against Vault's **underlyingTokens** when changing params in **ManagedTvlModule** and **DefaultBondTvlModule**. To allow such checks there should be a separate function in **Vault** to check if **_underlyingTokensSet** contains a **token**. Code example of sanity checks for **ManagedTvlModule**:

```
function setParams(
    address vault,
    Data[] memory data
) external noDelegateCall {
  IDefaultAccessControl(vault).requireAdmin(msg.sender);

  for (uint256 i = 0; i < data.length; i++) {
    if (!IVault(vault).containsUnderlyingToken(data[i].underlyingToken)) {
    //          ^-- additional function to check the presence
    //            of a token in Vault's underlying tokens
      revert InvalidToken();
    }
  }

  vaultParams[vault] = abi.encode(data);
  emit ManagedTvlModuleSetParams(vault, data, block.timestamp);
}
```

INFORMATIONAL-20	Lack of checking **bond.asset()** when setting data in **DefaultBondStrategy**	Fixed at 35b83a

Description

Line: DefaultBondStrategy.sol#L41

When adding bonds for a token there is only one check for **bond.asset()** not to equal zero address. So, it allows the admin to add a bond whose **bond.asset()** differs from **token**. This may break the deposit and withdrawal logic because unexpected transfers of tokens would occur.

Recommendation

We recommend adding a check for equality of **bond.asset()** to **token** in DefaultBondStrategy.setData():

```
...

if (IBond(data[i].bond).asset() != token) revert CustomError();

...
```

INFORMATIONAL-21	Result of **Vault.delegateCall()** is not checked in **DefaultBondStrategy._deposit()**	Acknowledged

Description

Line: DefaultBondStrategy.sol#L64

In **DefaultBondStrategy** during the deposit process underlying tokens of Vault are deposited to bonds via Vault.delegateCall() function. This function doesn't revert on a failed call, it just returns **success** and **response** variables, so the status of the call should be checked externally. But in DefaultBondStrategy._deposit() function these values are ignored. The deposit process occurs on every user deposit in Vault if **DepositCallback** is configured or when the admin or operator triggers it via DefaultBondStrategy.depositCallback() . Due to the unchecked status of **Vault.delegateCall()** at DefaultBondStrategy.sol#L64, users or operators may spend more gas for failing deposit cases in bonds. Also, there is no possibility to check if deposits in bonds would revert, besides analyzing full call trace, which is a bad solution for UX.

Recommendation

We recommend checking the status of **Vault.delegateCall()** in DefaultBondStrategy._deposit() function and revert on failing case.

Client's comments

We don't want the deposit transaction to fail because of the internal logic of the DefaultBondModule. Therefore, the result of the delegated call is simply ignored. From the point of view of UX, you can understand whether the deposit was successful by the presence of the **DefaultBondModuleDeposit** event.

STATE
MIND