Euler Earn Security Review

Auditors

Christoph Michel, Lead Security Researcher

M4rio.eth, Security Researcher

Report prepared by: Lucas Goiriz

October 23, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Euler labs is a team of developers and quantitative analysts building DeFi applications for the future of finance.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of euler-earn according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 10 days in total, Euler engaged with Spearbit to review the euler-earn protocol. In this period of time a total of **25** issues were found.

Summary

Project Name	Euler
Repository	euler-earn
Commit	a83c71...2a11
Type of Project	DeFi, Vaults
Audit Timeline	Oct 1 to Oct 14

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	7	3	4
Gas Optimizations	2	2	0
Informational	14	7	7
Total	**25**	**14**	**11**

5 Findings

5.1 Medium Risk

5.1.1 Unsafe type cast for `strategy.allocated` in `harvest`

Severity: Medium Risk

Context: EulerEarnVault.sol#L649

Description: In `_harvestStrategy`, the `strategy.allocated` is updated to the new strategy's total value:

```
uint256 eulerEarnShares = IERC4626(_strategy).balanceOf(address(this));
uint256 eulerEarnAssets = IERC4626(_strategy).previewRedeem(eulerEarnShares);
$.strategies[_strategy].allocated = uint120(eulerEarnAssets);
```

This type cast is not safe, the strategy's share value could have grown beyond `uint120.max`. The impact is that `strategy.allocated` is truncated to a smaller value but the function continues booking it as a normal profit of `eulerEarnAssets (uint256) - strategyAllocatedAmount` and adding it to the `uint256 $.totalAllocated`.

The allocated amount is now desynced and the invariant that `$.totalAllocated` is the sum of all strategies' `allocated` values does not hold anymore. Breaking proper rebalancing and blocking withdrawals with an underflow (see EulerEarnVault.sol#L511-L517) when deallocating from the strategy.

Recommendation: Consider capping `eulerEarnAssets` to `uint120.max` in case it grows beyond it. This will lead to temporarily ignoring any excess profit for that strategy (until deallocating) but a coherent `totalAllocated` accounting that does not block withdrawals.

Euler: Fixed in PR 111.

Spearbit: Verified.

5.1.2 Permissionless rebalance can lead to loss of funds for specific strategies

Severity: Medium Risk

Context: EulerEarn.sol#L245-L249, EulerEarnVault.sol#L754-L764

Description: The `rebalance` action is permissionless and can be executed by anyone. It will withdraw from strategies that are over-allocated and deposit into strategies that are under-allocated (according to the allocation points for each strategy). The strategies and the rebalance order can be defined as a parameter to `rebalance`.

Note that the value allocated to a vault with a deposit can be less than what is received by withdrawing from the vault again, for example, if the vault takes fees:

> The differences may be small (like if due to rounding error), or very significant (like if a Vault implements withdrawal or deposit fees, etc). EIP-4626

Note that the loss due to fees is not immediately booked in the `rebalance()` as the `strategy.allocated` value increased/decreased by the deposited / withdrawn value. It gets booked in a subsequent `harvest` operation, when the value of the shares is computed:

```
strategy.allocated = strategy.previewRedeem(strategy.balanceOf(this))
```

An attacker can:

1. `rebalance` the strategies.
2. `harvest` to book the fee loss, decreasing the strategy's `strategy.allocated` and the overall `totalAllocated`, resulting in the allocation percentage `strategy.allocated / totalAllocated` for that vault to drop below it's desired share again.
3. repeat the `rebalance`.

The `rebalance` step can be combined with finding *Flash deposit and withdraw can be used to off-balance assets* to rebalance almost the entire total assets to a specific strategy, increasing the loss for each iteration.

The impact is that the vault's funds can be lost entirely to fees by continuous rebalancing.

Recommendation:

- All strategies should have the invariant that depositing/withdrawing `assets` amount should increase/decrease `previewRedeem(strategy.balanceOf(this))` by this same `assets` amount. Otherwise, `rebalance` will incur losses.
- Consider restricting the permissionless rebalances that can be performed by implementing a cooldown for permissionless calls (always rebalancing *all strategies*), whereas a privileged role for `rebalance` can rebalance at any time. This can lead to delayed withdrawals for users if the cash reserve cannot be rebalanced right now, and rely on keepers / trusted roles to rebalance if the cash reserve is low but the trust assumption is already that users have to trust privileged roles to correctly allocate their funds to non-malicious vaults.

Euler: Fixed in PR 105.

Spearbit: Verified.

5.2 Low Risk

5.2.1 Performance fee is instantly given to fee recipient while yield is smeared for everyone else

Severity: Low Risk

Context: EulerEarnVault.sol#L620

Description: When a positive yield is harvested this yield is smeared and earned by the depositors over a 2 week period. However, the performance fee taken on the yield is not streamed to the fee recipient, they receive it instantly in the form of `EarnVault` shares. This can be considered unfair as the full shares earn their fair share of the very own yield the fee came from (and of any other yield earned from now on). The yield for other depositors will also get erased again if a loss occurs, but the fee recipient is not affected by this. This dilutes the yield for other participants.

Recommendation: A fairer solution would be to take the fee whenever the interest is accrued to the users (`_updateInterestAccrued`), not when it's earmarked to be paid out over the smearing period (`_harvest`/`_gulp`).

Euler: Acknowledged. After discussing with the team, and considering other options, we do think that the current implementation is the most suited for the use case of Earn vault, and we will make sure users are aware of this.

Spearbit: Acknowledged.

5.2.2 Strategies with fees not properly supported

Severity: Low Risk

Context: EulerEarnVault.sol#L764-L765

Description: When allocating to a strategy via `rebalance()`, the strategy's allocated amount is set to the deposited amount, instead of what the received shares are worth (`previewRedeem`).

```
// rebalance
IERC4626(_strategy).deposit(amountToRebalance, address(this));
$.strategies[_strategy].allocated = (strategyData.allocated + amountToRebalance).toUint120();

// harvest
// Use `previewRedeem()` to get the actual assets amount
uint256 eulerEarnShares = IERC4626(_strategy).balanceOf(address(this));
uint256 eulerEarnAssets = IERC4626(_strategy).previewRedeem(eulerEarnShares);
$.strategies[_strategy].allocated = uint120(eulerEarnAssets);
```

There might be a discrepancy between these two values if the strategy takes fees or, due to rounding, or for other reasons. This loss is only booked once `harvest` is called and the `previewRedeem()` value is fetched and compared to the allocated amount.

The loss will be socialized among all depositors. Even users that join after a large depositor will take that depositor's deposit/withdrawal fee loss once `harvest` is called which might be considered unfair.

Recommendation: As strategies with fees on the principal (management fees) will incur a loss on every rebalance, they shouldn't be supported. Ensure that the following invariant holds for every enabled strategy: depositing / withdrawing `assets` should increase/decrease `previewRedeem(strategy.balanceOf(this))` by `assets` (in practice, most vaults will incur a tiny rounding loss as `deposit` mints fewer shares to the depositor and `previewRedeem` further underestimates the assets the depositor receivers for the shares).

Euler: Acknowledged. We will highlight in future docs that strategies with fees on deposit/withdraw are not supported.

Spearbit: Acknowledged.

5.2.3 Potential underflow when withdrawing from strategies

Severity: Low Risk

Context: EulerEarnVault.sol#L511-L517

Description: When a withdrawal needs to withdraw from a strategy, a `withdrawAmount` is computed and the strategy's allocated amount is reduced by it:

```
uint256 underlyingBalance = strategy.maxWithdraw(address(this));
uint256 desiredAssets = _assets - assetsRetrieved;
uint256 withdrawAmount = (underlyingBalance >= desiredAssets) ? desiredAssets : underlyingBalance;
// allocated = strategy.previewRedeem(strategy.balanceOf(this))
$.strategies[address(strategy)].allocated -= uint120(withdrawAmount);
```

This performs an unsafe typecast to `uint120` and it should ensure that the subtraction never underflows. However, this requires the following conditions:

1. `strategy.allocated` being up-to-date, i.e., a harvest was performed: `$.strategies[_strategy].allocated = uint120(strategy.previewRedeem(eulerEarnShares))`. This condition always holds because a harvest is always performed if it's not a cash-reserve-only withdrawal.
2. `withdrawAmount <= strategy.allocated`: From the code we see that `withdrawAmount <= strategy.maxWithdraw(this)`. The strategy vault needs the **following invariant** to ensure the `allocated` subtraction doesn't underflow: `strategy.maxWithdraw(this) <= strategy.previewRedeem(strategy.balanceOf(this)) (= strategy.allocated)`.

Recommendation: Consider the following:

1. Cap the withdrawal amount for each strategy (`withdrawAmount`) at `uint120.max` to ensure the type-cast is safe and to not revert in case a strategy grows to a larger amount. (this will ignore any excess profit of a strategy but the current accounting only supports a `uint120` for `strategy.allocated`) See related issue "Unsafe type cast for `strategy.allocated` in `harvest`".
2. Ensure the invariant `strategy.maxWithdraw(this) <= strategy.previewRedeem(strategy.balanceOf(this))` holds for all enabled strategies. Alternatively, set `strategy.totalAllocated = 0` in case it would underflow (`withdrawAmount > strategy.totalAllocated`) to not block withdrawals.
3. Document these assumptions in the code.

Euler: Fixed in PR 110.

Spearbit: Acknowledged. Recommendation 1) has been addressed, the others have not been addressed yet.

5.2.4 `maxWithdraw`/`maxRedeem` returns wrong values for fee recipient

Severity: Low Risk

Context: EulerEarnVault.sol#L277

Description: Withdrawals can result in a harvest with a positive yield from which a performance fee is credited to the fee recipient in terms of minted shares. The `maxWithdraw(user)`/`maxRedeem(user)` functions do not take into account this increase in shares for the fee recipient.

The behavior of `withdraw` and `redeem` regarding the performance fee shares also differs:

- `withdraw` performs the harvest before the `require(_assets <= maxAssets, Errors.ERC4626ExceededMaxWithdraw(_owner, _assets, maxAssets))` check, so the fee recipient can withdraw their accrued performance fees.
- `redeem` performs the harvest **after** the `require(_shares <= maxShares, Errors.ERC4626ExceededMaxRedeem(_owner, _shares, maxShares))` check, so the fee recipient **cannot** withdraw their accrued performance fees immediately.

Currently, the `maxRedeem` and `redeem` functions are coherent but they both underestimate the max amount that can be withdrawn for the fee recipient. The `maxWithdraw` underestimates the amount that can currently be withdraw for the `feeRecipient`.

Recommendation: Consider fixing `redeem`, `maxWithdraw`, `maxRedeem` or adding a note that these functions underestimate the amounts for the fee recipient.

Euler: Fixed by adding notes in PR 119.

Spearbit: Verified.

5.2.5 `maxWithdraw`/`maxRedeem` does not always return optimal maximum amount

Severity: Low Risk

Context: EulerEarnVault.sol#L274-L281

Description: When performing a withdrawal, a harvest is skipped if withdrawn only from the cash reserve. Otherwise, a harvest could be performed (depending on the harvest cooldown) which can result in a loss.

The current algorithm starts by estimating the max amount to withdraw as the user's entire amount (`share balance * share price`) and simulates the harvest conditions based on that.

However, it could happen that the user's entire amount results in withdrawing more than the cash reserve which results in a harvest ending up with a loss, reducing the share price and the amount the user receives. Whereas if the user only withdrew the cash reserve, they would have skipped the loss and withdrawn more.

Recommendation: The optimal algorithm should take into account losses from the harvest and consider if it would be better to only withdraw the cash reserve to maximize the withdrawal amount. The current algorithm can underestimate the amount if a harvest results in a net loss.

Euler: Acknowledged, but giving that we do not want to incentivize users to frontrun harvesting a negative yield and the execution of loss deduction, we think the current implementation is more suited for a safer vault.

We may add some alert on frontend when a user will trigger a harvest/loss deduction because of the amount to withdraw, etc...

Spearbit: Acknowledged.

5.2.6 `ERC4626` `max*` view functions ignore hook behavior

Severity: Low Risk

Context: EulerEarnVault.sol#L80-L383

Description: The `deposit`, `mint`, `withdraw`, `redeem` functions all call a hook (if defined) at the beginning which can revert to implement pause states, or user/global limits. However, the corresponding `max*` functions do not take the hook behavior into account. This violates EIP4626-behavior for these functions:

> MUST factor in both global and user-specific limits, like if deposits are entirely disabled (even temporarily) it MUST return 0. maxDeposit

Recommendation: As the hook behavior can depend on the exact amount deposited/withdrawn but the `max*` functions try to figure out the amount in the first place, it's hard to perfectly simulate the hook behavior for every amount state. One could consider calculating the max amount as it is currently done and performing a subsequent hook call simulation with this amount.

Alternatively, accept the non-compliance and document that the `max*` functions ignore hook behavior and the actual deposit/withdrawal can end up reverting because of the hook. Integrators need to be able to handle this case.

Euler: Fixed in PR 118 by adding comments for the non-compliance.

Spearbit: Verified.

5.2.7 Flash deposit and withdraw can be used to off-balance assets

Severity: Low Risk

Context: EulerEarnVault.sol#L507

Description: The `_withdraw` function iterates over strategies in the withdrawal queue order and withdraws from the strategies in case more withdrawable assets are needed. This can be exploited by a griefer to off-balance the Earn vault strategy allocation, for example, leaving all assets in the last strategy of the withdrawal queue, affecting the APR of the Earn vault.

Example: Imagine 10*Misallocatedevenlyacrossthecashreserveand*4*strategies*(2M in each "strategy"). A user performs the following actions in a single transaction:

- flashloan $40M.
- deposit $40M.
- `rebalance(allStrategies)`. Each strategy now holds $10M.
- Withdraw $40M. The cash reserve only has $10M, the other $30M will be withdrawn from the first three strategies in the withdrawal queue.
- The final result is that the entire $10M initial total assets are in the last strategy.

Recommendation: A `rebalance` should be performed periodically.

Euler: Fixed in commit b5b19d67.

Spearbit: Verified.

5.3 Gas Optimization

5.3.1 `$.withdrawalQueue.length` should be cached inside `_previewHarvestBeforeWithdraw`

Severity: Gas Optimization

Context: EulerEarnVault.sol#L797

Description: `$.withdrawalQueue.length` can be cached inside the `_previewHarvestBeforeWithdraw` even if this is a `view` function and is called only inside view functions, this might be used onchain by 3rd parties.

Recommendation: Cache `$.withdrawalQueue.length` to save gas.

Euler: Fixed in PR 114.

Spearbit: Verified.

5.3.2 Internal balances should be fetched via `_balanceOf` instead of `nonReentrant balanceOf`

Severity: Gas Optimization

Context: Shared.sol#L191-L194, ERC20VotesUpgradeable.sol#L73

Description: The inherited `ERC20VotesUpgradeable._getVotingUnits` function uses the overridden `balanceOf` function, which is the function that has a `nonReentrant` modifier, resulting in reverts when called while the contract is locked. There is currently no impact as `_getVotingUnits` is only used for the `delegate` functions which do not have a reentrancy lock.

Recommendation: As a best practice and to reduce gas cost, all internal balance calls should use the `Shared._balanceOf` function instead of `balanceOf`. Consider overriding the `_getVotingUnits` and using the `_balanceOf` in the function body instead.

Euler: Fixed in PR 109.

Spearbit: Verified.

5.4 Informational

5.4.1 Missing `nonReentrantView` on `maxMint`

Severity: Informational

Context: PR 94

Description: During the engagement, the team provided additional changes, including a rework of the `maxMint` and `maxDeposit` functions to better reflect their intended values.

The `maxMint` function is missing the `nonReentrantView` modifier, as it now reads from storage rather than returning a constant. On the other hand, while the `maxDeposit` function also reads from storage via the `previewMint` call, that subsequent call is already guarded by the `nonReentrantView` modifier.

Recommendation: Consider adding `nonReentratView` modifier to the `maxMint`.

Euler: Fixed in PR 155.

Spearbit: Verified.

5.4.2 Missing functions from the `IEulerEarn` interface

Severity: Informational

Context: IEulerEarn.sol

Description: The `IEulerEarn` interface is missing the following function signatures:

- `asset()`
- `permit2Address()`
- `EVC()`
- `isHarvestCoolDownCheckOn()`

Recommendation: Consider adding the extra functions to the interface.

Euler: Fixed in PR 113.

Spearbit: Verified.

5.4.3 The `harvest` will be executed even if the queue is empty or fully emergencied

Severity: Informational

Context: EulerEarnVault.sol#L599-L621

Description: The `_harvest` function iterates through all strategies to collect the Profit and Loss (PnL). After the iteration, the PnL is applied to the storage, and an event is emitted:

```
emit Events.Harvest($.totalAllocated, totalPositiveYield, totalNegativeYield);
```

The issue arises when the withdrawal queue is empty, or all strategies are marked as `Emergency`. In such cases, no PnL is collected, yet the event is still emitted with `0` values for both `totalPositiveYield` and `totalNegativeYield`. This can lead to misleading data being recorded in the event logs, suggesting that harvesting occurred when in reality, no profit or loss was generated.

Recommendation: We recommend documenting this behavior. While an additional check could be added to prevent emitting the event when no PnL is collected, it may not be worth the extra gas, as this is likely a rare edge case.

Euler: Acknowledged.

This should be okay as the `(uint256 totalAllocated, uint256 totalYield, uint256 totalLoss)` emitted in the event should all be `(0,0,0)` so no misleading data is recorded.

Spearbit: Acknowledged.

5.4.4 Missing `init` subsequent calls from the `init` function

Severity: Informational

Context: EulerEarn.sol#L53-L58

Description: The `EulerEarn` contract is an upgradeable contract that inherits from several dependencies. In upgradaeble contracts, it is standard practice to invoke the `init` functions of all inherited dependencies to ensure proper initialization. Currently, the following `init` functions are missing:

- `__Votes_init_unchained`
- `__Nonces_init_unchained`
- `__Context_init_unchained`
- `__AccessControl_init_unchained`
- `__ERC165_init_unchained`

This issue is classified as informational, as these missing calls do not execute any logic. Therefore, omitting them has no impact on the contract's storage or functionality. However, including them would adhere to best practices for initialization.

Recommendation: Consider calling the remaining `init` functions to maintain consistency. Although the missing functions do not currently affect storage or logic, adhering to this best practice ensures that all inherited contracts are properly initialized, which may prevent potential issues in future upgrades or modifications.

Euler: Fixed in PR 112.

Spearbit: Verified.

5.4.5 Strategies in `Emergency` are not removed from the rewards

Severity: Informational

Context: Strategy.sol#L137-L163, Strategy.sol#L77-L91

Description: The `EULER_EARN_MANAGER` can call `optInStrategyRewards` and `enableRewardForStrategy` for a particular strategy only if the strategy is active, as we can see by the following check:

```
require($.strategies[_strategy].status == IEulerEarn.StrategyStatus.Active,
↪  Errors.StrategyShouldBeActive());
```

However, when a strategy is marked as `Emergency`, it is essentially decommissioned and should no longer influence the vault. Ideally, a decommissioned strategy should not continue to accrue rewards, even if it theoretically could, given that it's no longer active. Currently, this is not the case: a decommissioned strategy can still generate rewards unless it is manually removed from the rewards system. To fully decommission a strategy, the manager must manually call `optOutStrategyRewards` and `disableRewardForStrategy`.

Recommendation: Consider documenting properly on the remove/emergency function that a manual opting out from the rewards for that strategy is needed.

Euler: Fixed in PR 116.

Spearbit: Verified.

5.4.6 Strategies in `Emergency` mode can not be removed

Severity: Informational

Context: Strategy.sol#L142-L143

Description: The `GUARDIAN` role has the ability to mark a strategy's status as `Emergency`, acting as a circuit-breaker in the event of a malfunction (e.g., when withdrawals are no longer functioning on that strategy). When this is triggered, the state is updated as follows:

- Losses are socialized across all depositors.
- The strategy's allocated points are deducted from the total points: `$.totalAllocationPoints -= strategyCached.allocationPoints`.
- The strategy's allocated capital is subtracted from the total allocated capital: `$.totalAllocated -= strategyCached.allocated`.

Additionally, the `STRATEGY_OPERATOR` can remove a strategy from the strategy queue using the `removeStrategy` function. However, this can only occur if the strategy is in the `Active` state and has no `allocated`:

```
require(strategyStorage.allocated == 0, Errors.CanNotRemoveStrategyWithAllocatedAmount());
```

Combining these two flows creates an issue: a malfunctioning strategy marked as `Emergency` cannot be removed from the withdrawal queue because:

- The `allocated` capital is not reset to 0 when the strategy is marked as `Emergency`.

- Only `Active` strategies can be removed by the `removeStrategy` function.

This can be problematic because malfunctioning strategies that remain in the queue are still iterated over in various flows, leading to unnecessary gas consumption. Furthermore, there is a limit of `Constants.MAX_STRATEGIES` that can be added. A malfunctioning strategy occupying a slot in the queue prevents new, potentially beneficial strategies from being added, which negatively impacts depositors.

Recommendation: Consider leaving the possibility for the `STRATEGY_OPERATOR` to remove strategies that are in the `Emergency` mode as well.

Euler: Acknowledged.

This is expected behavior as we want to keep the Emergency strategy in the case of toggling back to Active. We expect that it will be highly unlikely that a curator will consume the MAX_STRATEGIES and end up with 10 faulty ones.

With that being said, if such a scenario were to occur, I believe think would be a feature rather than a bug, and it will be more beneficial for the users to not allow that curator to use the Earn vault anymore, as they will be blocked and could no longer remove or add a new strategy.

Spearbit: Acknowledged by the client.

5.4.7 Re-enabling emergency-mode strategy does not gulp

Severity: Informational

Context: Strategy.sol#L91-L103

Description: When re-enabling an emergency-disabled strategy via `toggleStrategyEmergencyStatus`, the strategy's former shares are booked as yield. However, this yield is not immediately smeared, users need to call `gulp()` first.

Recommendation: Consider calling `_gulp()` at the end of the `else` block, which first performs an existing interest accrual via `_updateInterestAccrued`.

Euler: Fixed in PR 102.

Spearbit: Verified.

5.4.8 `rebalance` depends on order of `strategies`

Severity: Informational

Context: EulerEarnVault.sol#L50-L52

Description: The `rebalance` function depends on the order of `strategies`. It should be called such that all the withdrawals happen first, and all the deposits afterwards. Otherwise, one could end up skipping a deposit when being temporarily short on cash (but it would become available after a `withdraw`).

Recommendation: Authorized users should order the `strategies` parameter such that all withdrawals happen before deposits.

Euler: Acknowledged.

We update the rebalance function as recommended in another issue, and only the address with the Rebalancer role can rebalance the Earn vault. We expect that this address will be owned by an entity related to the vault curator and have an incentive to behave in an honest way and always submit an array with the most optimal order of strategies.

Also added more Natspec in PR 117.

Spearbit: Acknowledged.

5.4.9 Strategy's `previewRedeem` might be manipulable

Severity: Informational

Context: EulerEarnVault.sol#L648

Description: The `EulerEarn` vault uses `strategy.previewRedeem` to measure the value allocated to this strategy. EIP-4626 warns that this value might be manipulable and is not always safe to be used as an oracle:

> The `preview` methods return values that are as close as possible to exact as possible. For that reason, they are manipulable by altering the on-chain conditions and are not always safe to be used as price oracles. This specification includes convert methods that are allowed to be inexact and therefore can be implemented as robust price oracles.

Recommendation: Before enabling a strategy, check that the `previewRedeem` function cannot be manipulated, for example, through "read-only reentrancy". Otherwise, the value might be over-/underestimated which can be exploited to book a profit or loss.

Euler: Acknowledged. We will highlight this in the security considerations part of the docs.

Spearbit: Acknowledged.

5.4.10 No slippage checks on withdraw

Severity: Informational

Context: EulerEarnVault.sol#L616

Description: The received assets from a `withdraw` / `redeem` call might be less than expected if a large loss happened while the transaction was pending. This can for example happen if a strategy enters emergency mode temporarily. The user might have preferred to stay allocated to the `EarnVault` instead of withdrawing at a reduced share price.

Recommendation: As interest is smeared over a period instead of added to the total assets instantaneously, the opportunity cost of withdrawing at a bad time and not being allocated is reduced. The user can deposit again and still capture the majority of the smeared interest, for example, in the case of a re-activation of an emergency-mode strategy.

Alternatively, consider the EIP-4626 recommendation or add a router that checks the received amounts:

> If implementors intend to support EOA account access directly, they should consider adding an additional function call for `deposit`/`mint`/`withdraw`/`redeem` with the means to accommodate slippage loss or unexpected deposit/withdrawal limits, since they have no other means to revert the transaction if the exact output amount is not achieved.

Euler: Acknowledged.

Spearbit: Acknowledged.

5.4.11 Typos & Doc improvements

Severity: Informational

Context: See below.

Description:

1. Shared.sol#L68: Resetting `interestLeft = 0` in `_deductLoss` so it gets gulped again afterwards only works if `$.totalAllocated` is reduced by the entire loss amount in between the `_deductLoss` and `_gulp` calls (this is done for both `_harvest` and `toggleStrategyEmergencyStatus`). Document that this relies on `$.totalAllocated` being reduced after the call.
2. StorageLib.sol#L32: It's unclear what "not for update" means for the reentrancy `lock`. Consider removing this part.
3. EulerEarnVault.sol#L592: Missing natspec for the `_isOnlyCashReserveWithdraw` parameter.

4. EulerEarnVault.sol#L842: Consider documenting that it uses cached `_totalAssets()` and `_totalSupply()` but in the positive yield branch these cached values are correct.
5. Strategy.sol#L65: `deduct` → `deducts`.
6. EulerEarn.sol#L368: `VaultModule` → `EulerEarnVaultModule`, this is just an example, should be replaced everywhere.
7. EulerEarnFactory.sol#L29-L30: `Aggreation vault' asset` → `Aggregation vault's asset` and `Vaut` → `Vault`.
8. Shared.sol#L97: `amd` → `and`.
9. EulerEarnVault.sol#L795-L800: The `if` statement order should match the one from the `harvest` function.
10. EulerEarnVault.sol#L663: `performace` → `performance` .
11. EulerEarnVault.sol#L257: The `convertToShares` comment should be improved to state that this is an approximation and should not be used to compute the share price due to missing harvest simulation.
12. EulerEarnVault.sol#L264: The `convertToAssets` comment should be improved to state that this is an approximation and should not be used to compute the share price due to missing harvest simulation.
13. EulerEarnVault.sol#L709: The comment should be → `cap.resolve()` will be `type(uint256).max` and therefore `capAmount` will be `type(uint120).max` if no cap is set.
14. Rewards.sol#L81: The caching to `rewardStreams` is unnecessary, should just do `IRewardStreams(IBalanceForwarder(_strategy).balanceTrackerAddress()).claimReward(_strategy, _reward, _recipient, _forfeitRecentReward);`.

Recommendation: Consider addressing the findings mentioned above.

Euler: Fixed in commit 76d968b3.

Spearbit: Verified.

5.4.12 Document why unsafe casts and arithmetic is safe

Severity: Informational

Context: EulerEarnVault.sol#L624, Shared.sol#L76, EulerEarnVault.sol#L830, EulerEarnVault.sol#L715

Description: The code performs some non-trivial unsafe type cast, unsafe arithmetic, and subtractions that should never underflow, but it's not documented why it's safe to do so:

1. EulerEarnVault.sol#L624: `lossAmount <= $.totalAllocated`: a strategy's loss is capped by the current `strategy.allocated`. As `totalAllocated` is the sum of all strategies' `strategy.allocated`, the loss can never spill over into the cash reserve. Therefore, the subtraction cannot underflow.
2. Shared.sol#L76, EulerEarnVault.sol#L830: The loss socialization reduces `$.totalAssetsDeposited` by `lossAmount - totalNotDistributed`. We require `lossAmount - totalNotDistributed <= $.totalAssetsDeposited` for the subtraction to not underflow:

```
lossAmount - totalNotDistributed
= lossAmount - ($.totalAllocated + cashBalance) + $.totalAssetsDeposited // def
↪  totalNotDistributed
<=  $.totalAllocated - ($.totalAllocated + cashBalance) + $.totalAssetsDeposited // by 1.
= -cashBalance + $.totalAssetsDeposited // cashBalance >= 0
<= $.totalAssetsDeposited
```

3. EulerEarnVault.sol#L715: Downcasting to `uint120` is safe as it leads to a smaller cap.

Recommendation: Consider documenting any unsafe typecast and arithmetic.

Euler: Fixed in PR 107.

Spearbit: Verified.

5.4.13 MEV for large interest events

Severity: Informational

Context: EulerEarnVault.sol#L71

Description: The interest is smeared over a 2-week period. Large interest events like turning off the emergency status for a strategy will spike the APR for this period and new depositors can come in to earn a share of the large interest, bringing down the APR as the fixed smeared interest is distributed over a larger set of depositors. This might feel unfair for depositors who suffered the loss when the strategy was emergency-removed.

Recommendation: Users should be aware of this.

Euler: Acknowledged.

Spearbit: Acknowledged.

5.4.14 Sudden price drops due to `deductLoss` could have unwanted effects

Severity: Informational

Context: Strategy.sol#L83, EulerEarnVault.sol#L609

Description: During harvest or `toggleStrategyEmergencyStatus` the loss suffered by that specific strategy is socialized across all the depositors after the interest removal:

- Shared.sol#L76:

```
// socialize the loss
$.totalAssetsDeposited = totalAssetsDepositedCache - _lossAmount;
```

This will cause a sudden price drop of the share that can be problematic:

- Depositors can frontrun this and withdraw as much as possible before the price drop. This is already documented in docs:

  > Although opportunistic depositors could frontrun loss socialisation by withdrawing earlier, that would only work if `harvest` is not triggered during the withdrawal because the `HARVEST_COOLDOWN` period hasn't pass yet.

- If the vault's shares are used on another platform the sudden price drop can be used to short the price. Another issue might be that if those shares are used on a lending platform, sudden price drops could cause liquidations.

Recommendation: Consider better documentation on the potential unwanted effects of sudden price drops to ensure users are aware of the risks.

Euler: Acknowledged. Will document this and make sure it is clear that the Earn vault shares should not be used as collateral and the share price should not be used as an oracle. See PR 106.

Spearbit: Verified.