Euler Rescue Strategy

Security Review

Cantina Managed review by:
M4rio.eth, Lead Security Researcher

November 12, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

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

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

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

From Nov 7th to Nov 12th the Cantina team conducted a review of euler-earn on commit hash f5bd06fd. The team identified a total of **6** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	3	0
Gas Optimizations	0	0	0
Informational	3	1	2
Total	**6**	**4**	**2**

The Cantina Managed team reviewed Euler's euler-earn holistically on commit hash 8858d7a6 and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Missing enough flashloan avenues can make the rescue impossible

Severity: Low Risk

Context: RescueStrategy.sol

Description: Currently, for the rescue operation to be successful we have 3 avenues of flashloaning funds that it will be deposited into the affected vault:

1. Direct Euler vault loan (`rescueEuler`) – asks an Euler flash-loan vault for the requested amount in a single call, then repays once the rescue workflow is finished.
2. Batched Euler loan (`rescueEulerBatch`) – runs an IEVC batch that temporarily enables the flash-loan vault as controller, borrows, executes the rescue step, and repays before disabling the controller again.
3. Morpho loan (`rescueMorpho`) – requests liquidity from a Morpho flash-loan provider for the asset, processes the rescue, and returns the funds.

Unfortunately this is not effective for all the chains where Euler is deployed which means that if a vault is affected on a chain where Euler has low liquidity and Morpho does not exists, the rescue operation would not be possible.

Recommendation: Consider adding more flashloan avenues, for example AAVE could be a possible flashloan avenue, even if it incurs a fee.

Euler: Fixed in commit 8858d7a6 by adding AAVE

Cantina Managed: Verified.

3.1.2 Relaying on EOA as a rescuer is dangerous

Severity: Low Risk

Context: RescueStrategy.sol#L26

Description: We assume vaults that opt into this strategy are trusted because once the strategy is enabled and the cap set, the `fundsReceiver` can effectively withdraw the vault's liquidity. `isStrategyAllowed` is only checked during `_setCap`, so after acceptance there's no on-chain protection preventing the owner or `fundsReceiver` from exfiltrating assets.

Main risks:

- **Single EOA rescuer** — the biggest operational risk is key compromise or rogue behavior by the rescue EOA. Because the rescuer must be an EOA to satisfy our `tx.origin` checks, a compromised key or malicious operator can trigger rescues or abuse the `call()`, it also has permit approves set in the constructor.
- **Immutable attack surface after cap set** — once the strategy cap is accepted, the factory check is no longer effective; that means compromise after rescue mode set is fatal.

Recommendation:

- First option:
  - Use an **MPC / multisig** for the rescuer key instead of a single EOA.
- Second option:
  - Use **double permissions**: enforce both `tx.origin == RESCUE_EOA` and `msg.sender == RESCUE_SAFE` for sensitive entry points. That way a rescue must (a) originate from the designated EOA and (b) be executed via the multisig's `exec` so the multisig owners must have approved the transaction. Allow `RESCUE_SAFE` to change `RESCUE_EOA`.

E.g. flash-loan execution requiring both:

- `tx.origin == RESCUE_EOA` (ensures the EOA executed the multisig transaction), and

- `msg.sender == RESCUE_SAFE` (ensures the multisig transaction was approved by multisig to be executed).

This preserves our `tx.origin` guarantee while adding multisig threshold for approval.

Second option: Instead of relying on `tx.origin`, we could just use a **`msg.sender` check for the `rescueX` functions and a temporary boolean flag** to track when a rescue is active.

E.g.

1. Add a private flag bool `rescueActive` to track when a rescue is in progress.
2. In `rescue*` functions replace `onlyRescueAccount` with:

```
require(msg.sender == rescueAccount, "unauthorized");
require(!rescueActive, "rescue ongoing");
```

3. Inside each `rescueX` function, set `rescueActive = true` at the start and reset it to false after processing the flashloan.
4. In the callback functions add:

```
require(rescueActive, "rescue inactive")`
```

5. For deposit (and any other helpers that used `onlyRescueAccount`), just require `rescueActive` instead (like above). that way, deposits only work while a rescue is running.

Euler: Fixed in commit d1b0ddf6.

Cantina Managed: Verified.

3.1.3 Invariant not enforced in a pool that is in Rescue mode

Severity: Low Risk

Context: RescueStrategy.sol#L188-L203

Description: A vault that is in Rescue mode should have the following setup:

- `supplyQueue[0] == RescueStrategy`.
- `supplyQueue.length == 1`.
- `withdrawalQueue[0] == RescueStrategy`.

Currently checks that ensure this is respected are missing from the rescue flow (flashloan + flashloan callback), which means that a vault can be in rescue mode and for example have another strategy as the first supply strategy which could lead to unexpected edge-cases.

Recommendation: Consider adding the following checks:

```
function _assertRescueMode() internal view {
    IEulerEarn vault = IEulerEarn(earnVault);

    // Must be the ONLY supply target
    require(vault.supplyQueueLength() == 1, "rescue: supplyQueue len != 1");
    require(address(vault.supplyQueue(0)) == address(this), "rescue: supplyQueue[0] !=
    ↪  rescue");

    // Must be first in withdraw queue (bank-run guard)
    require(address(vault.withdrawQueue(0)) == address(this), "rescue: withdrawQueue[0]
    ↪  != rescue");
}

function _processFlashLoan(uint256 loanAmount) internal {
    _assertRescueMode();

   // ...
}
```

Euler: Fixed in commit 180c1e89.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 Performance fee should be set to 0 during rescue state

Severity: Informational

Context: RescueStrategy.sol#L199

Description: A vault currently incurs a performance fee, for the vaults that will be in rescue mode, the performance fee should be 0 so that full assets could be rescued to the `fundsReceiver`.

Recommendation: Consider setting the performance fee to 0 for the vaults that it will adopt the rescue strategy.

Euler: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 The `deposit` returns the amount of assets deposited

Severity: Informational

Context: RescueStrategy.sol#L100

Description/Recommendation: The `deposit` function should theoretically function as an ERC4626.deposit that returns the amount of shares, not the amount of assets that got deposited.

Currently this strategy was built as a way to rescue liquidity in vaults where lenders can not withdraw due to utilization being 100%. As we do not see necessarily a risk with this current flow, the reason for pointing this issue out, is to make sure it's documented enough so that, in case this strategy is used in any other context, in the future, this will be taken into account.

Euler: Acknowledged. This shouldn't really matter. Returning assets one to one, and then 0 on maxWithdraw from earn vault perspective is like depositing into a fresh new EVK vault with exchange rate 1:1 and having all assets borrowed out immediately.

Cantina Managed: Acknowledged.

3.2.3 Emit event with rescued amount for easier offchain accounting

Severity: Informational

Context: RescueStrategy.sol#L189-L202

Description/Recommendation: Emit an event with the rescued amount in the `_processFlashLoan` so that we can easily account offchain all the rescued amounts.

Euler: Fixed in commit b6225bab.

Cantina Managed: Verified.