CANTINA

# Euler EVK Periphery PR 361

## Security Review

Cantina Managed review by:

**M4rio**, Lead Security Researcher
**Slowfi**, Security Researcher

October 14, 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 Oct 10th to Oct 12th the Cantina team conducted a review of evk-periphery on commit hash 642602f8. The team identified a total of **15** issues:

**Issues Found**

| Severity | Count | Fixed | Acknowledged |
|---|---|---|---|
| Critical Risk | 0 | 0 | 0 |
| High Risk | 2 | 2 | 0 |
| Medium Risk | 0 | 0 | 0 |
| Low Risk | 0 | 0 | 0 |
| Gas Optimizations | 2 | 2 | 0 |
| Informational | 11 | 7 | 4 |
| **Total** | **15** | **11** | **4** |

The Cantina Managed team reviewed Euler's `evk-periphery` PR 361 holistically on commit hash 4711a8ad and concluded that all findings were addressed and no new vulnerabilities were identified.

# 3 Findings

## 3.1 High Risk

### 3.1.1 Dust removal in OFT Adapter causes the bridging to fail

**Severity:** High Risk

**Context:** FeeFlowControllerEVK.sol#L174-L175, OFTFeeCollector.sol#L73-L74

**Description:** When bridging the `paymentToken` using LayerZero OFT, the contract pulls the full auction price from the buyer, but the OFT adapter only transfers a reduced amount after removing dust. This leaves leftover dust (the truncated part) stuck in the `FeeFlowControllerEVK` because the full amount is taken but only a portion is sent.

For example, with 18-decimal tokens and OFT's shared decimals of 6, the `decimalConversionRate` is $10^{12}$. The `_removeDust` function cuts off amounts to multiples of $10^{12}$, but the contract sets the full `paymentAmount` in SendParam for both amountLD and minAmountLD.

The fact that we also set the `minAmountLD` as the same `paymentAmount` amplifies the issue because it will causes the send to fail due to slippage because `amountLD < minAmountLD`.

Here's the current code causing the issue:

```
paymentToken.safeTransferFrom(sender, address(this), paymentAmount);

SendParam memory sendParam = SendParam({
    amountLD: paymentAmount,  // Full amount specified
    minAmountLD: paymentAmount,  // Full amount specified, but actual sent is dedusted
    // ...
});
IOFT(oftAdapter).send{value: fee.nativeFee}(sendParam, fee, address(this));  // Only dedusted amount is
↪  transferred
```

And the `_debitView` function that removes dust:

```
function _debitView(
    uint256 _amountLD,
    uint256 _minAmountLD,
    uint32 /*_dstEid*/
) internal view virtual returns (uint256 amountSentLD, uint256 amountReceivedLD) {
    amountSentLD = _removeDust(_amountLD);  // Removes dust here
    amountReceivedLD = amountSentLD;
    if (amountReceivedLD < _minAmountLD) {
        revert SlippageExceeded(amountReceivedLD, _minAmountLD);
    }
}
```

The `_removeDust` function:

```
return (_amountLD / decimalConversionRate) * decimalConversionRate;
```

This mismatch means the contract overpulls from the buyer, and the send fails because `minAmountLD` doesn't account for the dedusting. The same issue is present as well in `OFTFeeCollector`.

**Recommendation:** Before sending, use the quoteOFT function to get the expected amountSentLD, then use that value for amountLD and minAmountLD in SendParam. This ensures the amounts match what will actually be transferred and avoids failures or trapped dust.

**Euler:** This should be mitigated in the commit that implements the recommendations from the LZ engineer (e47565f) and commit 130e3df.

In the `buy` function, any abandoned dust will be accumulated and will keep waiting for the next buy call. In the `collectFees` function the same will happen. Setting `minAmountLD` is safe as long as we control/trust the adapters.

Furthermore, a follow up for the approval not to be too little, in commit faf562ef.

**Cantina Managed:** Fix verified.

### 3.1.2 Contracts attempt to pay LayerZero native fee without a way to hold or receive native funds

**Severity:** High Risk

**Context:** FeeFlowControllerEVK.sol#L183, OFTFeeCollector.sol#L82

**Description:** The function `collectAndSend` from contract `OFTFeeCollector` and the function `buy` from contract `FeeFlowControllerEVK` pay the LayerZero native fee from the contract's own balance:

```
IOFT(adapter).send{value: fee.nativeFee}(sendParam, fee, address(this));
```

Both contracts lack a payable `receive()` (and the functions themselves are non-payable), so there is no reliable way to fund the required `fee.nativeFee`. Any non-zero quoted fee will cause the call to revert. Additionally, LayerZero may refund excess native fee to the sender; without a payable `receive()`, such refunds to the contract would also revert.

**Recommendation:** Consider to:

- Make the sending functions payable and require the caller to provide exactly the quoted fee (`require(msg.value == fee.nativeFee)`), refunding any excess if applicable; or add a payable `receive()` so the contracts can be pre-funded and accept LayerZero refunds.
- If using pre-funding, add monitoring and an optional minimum ETH balance check before attempting sends.
- Optionally skip the send when `fee.nativeFee == 0` to avoid accidental reverts on unfunded contracts.
- Emit events recording the quoted and paid fees for observability.

**Euler:** Fixed in commit 471ca2de.

**Cantina Managed:** Fix verified.

## 3.2 Gas Optimization

### 3.2.1 `feeToken` should be immutable to harden configuration and save gas

**Severity:** Gas Optimization

**Context:** FeeCollectorUtil.sol#L22

**Description:** The function `constructor` from contract `FeeCollectorUtil` assigns `feeToken` once and there is no setter:

```
IERC20 public feeToken;
```

Leaving it as a mutable storage slot increases the risk of unintended changes in future upgrades and adds a persistent storage read on every access.

**Recommendation:** Consider to declare it immutable and set it in the constructor:

```
IERC20 public immutable feeToken;
```

This hardens the configuration against accidental mutation and reduces gas by avoiding storage reads. If this contract is intended to be used behind a proxy (and thus needs per-proxy configurability), consider to keep it as storage but still avoid any setter and emit an initialization event documenting its value.

**Euler:** Fixed in commit b9982d59.

**Cantina Managed:** Fix verified.

### 3.2.2 Inefficient error handling after `convertFees()` failure increases gas use

**Severity:** Gas Optimization

**Context:** FeeCollectorUtil.sol#L100-L101

**Description:** The function `_convertAndRedeemFees` from contract `FeeCollectorUtil` attempts `redeem(...)` even when `convertFees()` has already reverted, due to the current `try/catch` structure:

```
try IEVault(vault).convertFees() {} catch {}
try IEVault(vault).redeem(type(uint256).max, address(this), address(this)) {} catch {}
```

When `convertFees()` fails, proceeding to `redeem(...)` for the same `vault` is wasteful and can unnecessarily inflate gas usage within the loop.

**Recommendation:** Consider to insert a `continue` in the first `catch` block to skip `redeem(...)` for that `vault`, e.g.:

```
try IEVault(vault).convertFees() {} catch { continue; }
try IEVault(vault).redeem(type(uint256).max, address(this), address(this)) {} catch {}
```

This reduces gas and aligns with the intent to keep the overall loop resilient without incurring avoidable costs.

**Euler:** Fixed in commit 850b60df.

**Cantina Managed:** Fix verified.

## 3.3 Informational

### 3.3.1 Typos & Minor Issues

**Severity:** Informational

**Context:** FeeFlowControllerEVK.sol#L13, FeeFlowControllerEVK.sol#L44, FeeFlowControllerEVK.sol#L69, FeeFlowControllerEVK.sol#L220, IRMBasePremium.sol#L36, IRMBasePremium.sol#L119, OFTGulper.sol#L7, FeeCollectorGulper.sol#L6

**Description:** Throughout the scope we noticed various typos or minor issues that could be aggregated into one.

- src/IRM/IIRMBasePremium.sol:36: `_rateOverrides` should be `rateOverrides` because it's public.
- src/IRM/IRMBasePremium.sol:119: `premiumRate_` should be `uint248`.
- src/FeeFlow/FeeFlowControllerEVK.sol:221: Consider using `encodeWithSelector` as it is more `idiomatic` to encode a selector.
- src/Util/FeeCollectorGulper.sol:6, src/OFT/OFTGulper.sol:7: Consider using an interface instead of `EulerSavingsRate`.
- src/FeeFlow/FeeFlowControllerEVK.sol:69: The `EmptyError` is not used.
- src/FeeFlow/FeeFlowControllerEVK.sol:13: `Continous` ⇒ `Continuous`.
- src/FeeFlow/FeeFlowControllerEVK.sol:14: Wrong comment on `lock` state numeration.

**Recommendation:** Consider fixing the issues mentioned above.

**Euler:** Fixed in commits 4339283 and 471ca2de.

**Cantina Managed:** Fix verified.

### 3.3.2 Consider emitting and event if hook fails

**Severity:** Informational

**Context:** FeeFlowControllerEVK.sol#L221

**Description:** The code performs a call to an external hook target but does not check if the call succeeded or handle any errors. This can lead to silent failures where problems go unnoticed, and it misses a chance to capture error details safely.

```
 // Perform the hook call if the hook target is set
if (hookTarget != address(0)) {
    // We do not check the success of the call as we allow it silently fail
    (bool success,) = hookTarget.call(abi.encode(hookTargetSelector));
    success;
}
```

**Recommendation:** Add an event to log when the hook call fails, so failures are not silent. Furthermore if we want to log the revert reason would require to use assembly for the call and limit the revert message to 32 bytes. Just an example, not production code:

```
bytes32 copied;
assembly ("memory-safe") {
    // ...
    success := call(gas(), target, value, dptr, dlen, 0, 0)
    // ...
    let returndata_size := mload(returndata)
    copied := (add(32, returndata), returndata_size)
}
emit Fail(copied);
```

**Euler:** Fixed in commit b1f523d0.

**Cantina Managed:** Fix verified.

### 3.3.3 Add recover native token

**Severity:** Informational

**Context:** OFTGulper.sol#L37

**Description:** The contract has a function to recover ERC20 tokens, but it does not have a way to recover ETH. Since the `lzCompose` function is payable, ETH can be sent to the contract and get stuck there with no way to get it out.

**Recommendation:** Consider using `recoverToken` from `FeeCollectorUtil`.

**Euler:** Fixed in commit fb346bb1.

**Cantina Managed:** Fix verified.

### 3.3.4 The `_feeToken` should be equal with `esr.asset()`

**Severity:** Informational

**Context:** FeeCollectorGulper.sol#L25

**Description:** The constructor takes a fee token address separately and assigns it directly without checking if it matches the asset of the EulerSavingsRate (ESR) contract. This could lead to mistakes where the wrong token is used, this is a different unlike in the OFT Gulper where the fee token is taken directly from the ESR.

**Recommendation:** Add a check in the constructor to ensure that `esr.asset() == _feeToken` before assigning it.

**Euler:** Fixed in commit 4defa324.

**Cantina Managed:** Fix verified.

### 3.3.5 The compose message is hardcoded

**Severity:** Informational

**Context:** OFTFeeCollector.sol#L76

**Description:** The compose message in the SendParam in the `OFTFeeCollector` is currently hardcoded to a fixed value `0x01` when `isComposedMsg` is `true`. This might limit future flexibility if you need to use different or more detailed calldata for composed messages, requiring code changes later:

```
SendParam memory sendParam = SendParam({
    dstEid: dstEid,
    to: bytes32(uint256(uint160(dstAddress))),
    amountLD: balance,
    minAmountLD: balance,
    extraOptions: "",
    composeMsg: isComposedMsg ? abi.encode(0x01) : bytes(""), // <==
    oftCmd: ""
});
```

**Recommendation:** Make the compose message a configurable parameter in the `configure` that defaults to 0x01 for now. This way, you can easily adjust it without modifying the core function if new requirements come up.

**Euler:** Fixed in commit 61ae26c8.

**Cantina Managed:** Fix verified.

### 3.3.6 The `collectFees` fees could revert if redeem from a vault with a hook that could gas bomb

**Severity:** Informational

**Context:** FeeCollectorUtil.sol#L101

**Description:** In FeeCollectorUtil.sol, the `redeem` call to the vault with hooks can cause a gas bomb, reverting the entire operation. Try-catch won't catch OOG, only exceptions.

```
try IEVault(vault).convertFees() {} catch {}
try IEVault(vault).redeem(type(uint256).max, address(this), address(this)) {} catch {}
```

**Recommendation:** Add a gas limit to `redeem` call, e.g., `redeem{ gas: 50000 }(type(uint256).max, address(this), address(this))`, but this might limit the vaults that can be used or simply ack this with a comment and keep it mind that you will have to remove problematic vaults from the list if this happens.

**Euler:** Acknowledged.

**Cantina Managed:** Acknowledged.

### 3.3.7 `dstAddress` allows bridging to `address(0)`

**Severity:** Informational

**Context:** OFTFeeCollector.sol#L54

**Description:** The function `configure` from contract `OFTFeeCollector` assigns `dstAddress` without checking for zero. Later, `collectAndSend()` builds the LayerZero `SendParam` using `dstAddress`:

```
to: bytes32(uint256(uint160(dstAddress))),
```

If `dstAddress` is zero, the send targets `address(0)` on the destination, which may result in funds being credited to a black-hole address or the message being rejected, depending on the receiver implementation.

**Recommendation:** Consider to enforce non-zero in `configure`:

```
require(_dstAddress != address(0), "dstAddress is zero");
```

Optionally re-validate in `collectAndSend()` (defense in depth) and emit an event on configuration updates for monitoring.

**Euler:** Fixed in commit 02d8f2bd.

**Cantina Managed:** Fix verified.

### 3.3.8 Unbounded vault iteration can run out of gas; introduce batched fee claims

**Severity:** Informational

**Context:** FeeCollectorUtil.sol#L98-L101

**Description:** The function `_convertAndRedeemFees` from contract `FeeCollectorUtil` iterates over the entire `_vaultsList` and performs external calls (`convertFees()` and `redeem(...)`) per vault. As the vault list grows or individual vault calls are expensive, a single call may exceed the block gas limit, causing consistent reverts and creating an operational DoS on fee collection.

**Recommendation:** The function `_convertAndRedeemFees` from contract `FeeCollectorUtil` should support batched processing. E.g., add `convertAndRedeemFees(uint256 start, uint256 max)` to handle at most `max` vaults starting at `start`, optionally track a rolling cursor in storage for progress across transactions, and emit per-vault events so off-chain automation can resume reliably, then prefer this batched path operationally to avoid out-of-gas DoS on large vault sets.

**Euler:** Acknowledged, won't fix. If it becomes a problem, the way most likely it will need to be handled is multiple fee flow contracts with multiple fee collectors.

**Cantina Managed:** Acknowledged by Euler team.

#### 3.3.9 Silent failures in fee conversion/redeem hinder observability and debugging

**Severity:** Informational

**Context:** FeeCollectorUtil.sol#L100-L101

**Description:** The function `_convertAndRedeemFees` from contract `FeeCollectorUtil` catches exceptions from `convertFees()` and `redeem(...)` with empty `catch` blocks, resulting in silent failures that conceal which vaults failed and why; this reduces observability, complicates incident response, and can mask persistent misconfiguration or hook-related blocks.

**Recommendation:** The function `_convertAndRedeemFees` from contract `FeeCollectorUtil` should emit lightweight events inside the `catch` paths (e.g., `event FeeConversionFailed(address vault)` and `event FeeRedeemFailed(address vault)` with optional reason bytes if available) so operators can detect and diagnose failures without reverting the entire loop.

**Euler:** Acknowledged, won't fix. If it becomes a problem, the way most likely it will need to be handled is multiple fee flow contracts with multiple fee collectors.

**Cantina Managed:** Acknowledged by Euler team.

#### 3.3.10 Improve cross-chain test coverage for OFT flows and fee handling

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** The function `collectAndSend` from contract `OFTFeeCollector` and the function `buy` from contract `FeeFlowControllerEVK` depend on LayerZero OFT behavior (shared decimals, de-dusting, native fee funding/refunds, compose handlers), but current tests do not simulate these edge cases. This leaves gaps around: (1) OFT de-dusting with `sharedDecimals` (amount rounding vs. `minAmountLD`), (2) native fee funding paths and refunds to the sender, (3) zero-address config validation for `dstAddress`, (4) event emission on catch-paths in fee collection utilities, (5) batching behavior to avoid out-of-gas in vault loops, and (6) `lzCompose` entry semantics.

**Recommendation:** Consider to extend the test suite (can be grouped under a LayerZero/OFT integration spec) to: use LayerZero contract mocks to simulate bridging; cover 18→6 decimal de-dusting and assert no dust remains stranded; verify `minAmountLD` handling with quoted `amountSentLD`; test both pre-funded and `payable` fee paths including LayerZero native fee refunds; assert revert on zero `dstAddress`; assert events emitted on `convertFees()`/`redeem()` catch paths; validate batched fee-claim iteration; and exercise `lzCompose` entry to ensure only intended side effects occur.

**Euler:** Acknowledged. The test suite is being improved.

**Cantina Managed:** Acknowledged by Euler team. This issue is being fixed.

#### 3.3.11 Max rate for `IRMBasePremium`

**Severity:** Informational

**Context:** *(No context files were provided by the reviewer)*

**Description:** The `IRMBasePremium` allows the interest rate to be set without any upper limit, as the total rate is just the sum of `baseRate` and `premiumRate` (or an override). This could result in extremely high rates that might destabilize borrowing, lead to user issues, or be exploited if admins set excessive values unintentionally.

**Recommendation:** Add a `maxRate` parameter, set during construction or a constant. In `computeInterestRateInternal`, cap the returned rate to not exceed `maxRate`.

**Euler:** Fixed in commit 5d817390.

**Cantina Managed:** Fix verified.