yAudit Euler Vault Kit Periphery Review

Review Resources:

- Swapper documentation
- EVK whitepaper

Auditors:

- Adriro
- Panda

Table of Contents

Review Summary

Euler Vault Kit Periphery

The Euler Vault Kit is a system for constructing credit vaults. Credit vaults are ERC-4626 vaults with added borrowing functionality. Unlike typical ERC-4626 vaults, which earn yield by actively investing deposited funds, credit vaults are passive lending pools.

The periphery codebase includes factories for the oracle router and the interest rate model (IRM), the Perspectives contracts used to define validity criteria for EVK vaults, and the Swapper utilities for performing DEX swaps for EVK vault operations.

The contracts of the Euler Vault Kit Periphery repository were reviewed over 8 days. Two auditors performed the code review between June 17th and June 26th, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 090cbba2870f7ddbbbd6d360ee35a69475ab6ff8 for the EVK Periphery repository.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src
├── BaseFactory
│   ├── BaseFactory.sol
│   └── interfaces
│       └── IFactory.sol
├── IRMFactory
│   ├── EulerKinkIRMFactory.sol
│   └── interfaces
│       └── IEulerKinkIRMFactory.sol
├── OracleFactory
│   ├── AdapterRegistry.sol
│   ├── EulerRouterFactory.sol
│   └── interfaces
│       ├── IEulerRouter.sol
│       └── IEulerRouterFactory.sol
├── Perspectives
│   ├── deployed
│   │   ├── EscrowSingletonPerspective.sol
│   │   ├── EulerDefaultClusterPerspective.sol
│   │   ├── EulerFactoryPerspective.sol
│   │   └── ImmutableWhitelistPerspective.sol
│   └── implementation
│       ├── BasePerspective.sol
│       ├── DefaultClusterPerspective.sol
│       ├── PerspectiveErrors.sol
│       └── interfaces
│           └── IPerspective.sol
└── Swaps
    ├── ISwapper.sol
    ├── SwapVerifier.sol
    ├── Swapper.sol
    ├── handlers
    │   ├── BaseHandler.sol
    │   ├── OneInchHandler.sol
    │   ├── UniswapAutoRouterHandler.sol
```

```
    |   ├── UniswapV2Handler.sol
    |   └── UniswapV3Handler.sol
    └── vendor
        ├── ISwapRouterV2.sol
        ├── ISwapRouterV3.sol
        └── IUniswapV3SwapCallback.sol
```

After the findings were presented to the Euler team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Euler and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Adequate access control mechanisms are in place.
Mathematics	Good	There are no complex mathematical operations.
Complexity	Average	Despite the well-structured and modular codebase, Perspectives introduce a level of complexity that is difficult to reason about.
Libraries	Good	The codebase relies on an updated version of the OpenZeppelin library.

Category	Mark	Description
Decentralization	Good	Factories and Perspectives favor decentralization and on-chain transparency.
Code stability	Good	The codebase was under active development, but the set of contracts under scope remained stable.
Documentation	Average	Lack of formal requirements for the Perspectives made it difficult to contrast the implementation against a specification.
Monitoring	Good	Proper monitoring events are in place for the factories and the verification of perspectives.
Testing and verification	Average	The tests related to the Swapper contracts were failing. Additionally, only the 1inch adapter is covered.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings
  - Findings that can improve the gas efficiency of the contracts.
- Informational
  - Findings including recommendations and best practices.

Critical Findings

1. Critical - Insufficient checks for oracle safety

The DefaultClusterPerspective.sol perspective checks that the oracle router resolves to a valid adapter, but this alone may not be sufficient to guarantee a safe quote.

Technical Details

Some of the checks in the DefaultClusterPerspective.sol contract aim to verify the safety of the vault's oracle. These include:

1. The router is a valid deployment from the known factory.
2. Governance is null, preventing any further changes.
3. There is no fallback oracle.
4. According to the adapter registry, the resolution between the vault and the collaterals to the unit of account results in a valid adapter.

The resolution mechanism of EulerRouter.sol is given by the implementation of `resolveOracle()`:

```
123:     function resolveOracle(uint256 inAmount, address base, address quote)
124:         public
125:         view
126:         returns (uint256, /* resolvedAmount */ address, /* base */ address, /* quote
*/ address /* oracle */ )
127:     {
128:         // 1. Check the base case.
129:         if (base == quote) return (inAmount, base, quote, address(0));
130:         // 2. Check if there is a PriceOracle configured for base/quote.
131:         address oracle = getConfiguredOracle(base, quote);
132:         if (oracle != address(0)) return (inAmount, base, quote, oracle);
133:         // 3. Recursively resolve `base`.
134:         address baseAsset = resolvedVaults[base];
135:         if (baseAsset != address(0)) {
136:             inAmount = IERC4626(base).convertToAssets(inAmount);
137:             return resolveOracle(inAmount, baseAsset, quote);
138:         }
139:         // 4. Return the fallback or revert if not configured.
140:         oracle = fallbackOracle;
141:         if (oracle == address(0)) revert Errors.PriceOracle_NotSupported(base,
quote);
142:         return (inAmount, base, quote, oracle);
143:     }
```

As we can see in the code snippet, the resolution is not solely determined by the configured oracle. There is also a recursion on the configured vaults to convert between assets. A malicious actor could configure an intermediate step using a vault they control, manipulating the exchange rate while having the resolution end up with a valid adapter.

Impact

Critical. The perspective can verify vaults configured with a tampered oracle.

Recommendation

Intermediate ERC4626 vaults involved in the conversion should also be verified.

Developer Response

Fixed as part of: PR#40.

High Findings

1. High - Any vault can be added to the immutable whitelist perspective

Verifying the vault with the “fail early” option disabled won’t trigger any errors and will add any arbitrary vault to the verified set.

Technical Details

The ImmutableWhitelistPerspective.sol implementation of `perspectiveVerifyInternal()` calls `testProperty()` with a hardcoded `false` value to make it always fail.

```
29:     function perspectiveVerifyInternal(address) internal virtual override {
30:         testProperty(false, 0);
31:     }
```

However, using a zero error code here won’t trigger any revert when the “fail early” option is set to false since `perspectiveVerify()` will only raise when the aggregated set of error codes is non-zero when checks are delayed.

```
71:         uint256 errors;
72:         assembly {
73:             // restore the cached values
74:             tstore(transientVault.slot, _vault)
75:             tstore(transientFailEarly.slot, _failEarly)
76:
77:             errors := tload(transientErrors.slot)
78:         }
79:
80:         // if early fail was not requested, we need to check for any property errors
that may have occurred.
81:         // otherwise, we would have already reverted if there were any property
errors
82:         if (errors != 0) revert PerspectiveError(address(this), vault, errors);
83:
84:         // set the vault as permanently verified
85:         verified.add(vault);
86:         emit PerspectiveVerified(vault);
```

Proof of Concept

Place the following test in the ImmutableWhitelistPerspective.t.sol file.

```
function test_ImmutableWhitelistPerspective_VerifyArbitraryVault() public {
    address[] memory whitelist = new address[](0);

    ImmutableWhitelistPerspective perspective = new
ImmutableWhitelistPerspective(whitelist);

    address vault = address(1);

    assertFalse(perspective.isVerified(vault));

    perspective.perspectiveVerify(vault, false);

    assertTrue(perspective.isVerified(vault));
}
```

Impact

High. Any vault can be added to the set of verified addresses in the ImmutableWhitelistPerspective.sol perspective.

Recommendation

Set an error code different than zero, or revert directly in `perspectiveVerifyInternal()`.

Developer Response

Fixed: 0398176.

Medium Findings

None.

Low Findings

1. Low - Router deployments are not added to the list

The EulerRouterFactory.sol factory fails to push new contracts to the internal list of deployed routers.

Technical Details

New contracts are added to the `deploymentInfo` mapping but are not pushed to the `deployments` array.

```
17:     function deploy(address governor) external returns (address) {
18:         address router = address(new EulerRouter(governor));
19:         deploymentInfo[router] = DeploymentInfo(msg.sender, uint96(block.timestamp));
20:         emit ContractDeployed(router, msg.sender, block.timestamp);
21:         return router;
22:     }
```

Impact

Low. Deployments for EulerRouter.sol contracts cannot be enumerated.

Recommendation

Push the new address to the `deployments` array.

Developer Response

Fixed: 11637ff.

2. Low - Deposit will be skipped when repaying max debt

The deposit action in `repayAndDeposit()` won't be executed when repaying the whole debt as the balance comparison will always be false.

Technical Details

The `repayAndDeposit()` function repays the specified amount of debt and deposits any balance that could be left back to the vault.

```
128:     function _repayAndDeposit(address token, address vault, uint256 repayAmount,
address account) internal {
129:         uint256 balance = setMaxAllowance(token, vault);
130:         repayAmount = _capRepayToBalance(repayAmount, balance);
131:
132:         IEVault(vault).repay(repayAmount, account);
133:
134:         if (balance > repayAmount) {
135:             IEVault(vault).deposit(type(uint256).max, account);
136:         }
137:     }
```

When `repayAmount` is set to `type(uint256).max` to repay the debt in full, the check in line 134 will always fail, despite potentially having some excess after the repayment.

Impact

Low. Deposit may inadvertently be skipped when fully repaying the debt.

Recommendation

Use the resulting value from the call to `repay()` to compare against the balance to check if there is an excess to be deposited.

Developer Response

Fixed as recommended: 6ec1912.

3. Low - Swapper griefing attack is still possible despite reentrancy guards

A malicious actor could exploit previously granted token allowances to sweep assets out of the Swapper.sol contract.

Technical Details

The Swapper.sol contract enforces a reentrancy lock to prevent potential denial-of-service attacks in which an attacker could reenter the contract between execution to sweep any tokens out of the contract.

However, the same attack can be executed without reentering the contract by taking advantage of existing allowances and pulling the tokens using `transferFrom()` in any potential callback. The allowance can be set up by using `deposit()` or `repay()` functions with the attacker's address as the vault argument.

```
121:     function _deposit(address token, address vault, uint256 amountMin, address
account) internal {
122:         uint256 balance = setMaxAllowance(token, vault);
123:         if (balance >= amountMin) {
124:             IEVault(vault).deposit(balance, account);
125:         }
126:     }
```

Impact

Low. Swaps can be griefed by pulling tokens out of the contract during callbacks.

Recommendation

Ensure vaults are whitelisted or reset the allowance back to zero (may cause conflicts with some ERC20 implementations).

Developer Response

Fixed in 4148719.

Remove allowance from user-provided addresses.

Gas Saving Findings

1. Gas - Max liquidation discount is checked for each collateral in DefaultClusterPerspective.sol

The check is independent of the collateral but is performed in each iteration of the LTV list loop.

Technical Details

The implementation of `perspectiveVerifyInternal()` performs the check inside the `ltvList` loop.

```
106:         // cluster vaults must have recognized collaterals
107:         for (uint256 i = 0; i < ltvList.length; ++i) {
108:             address collateral = ltvList[i];
109:
110:             // the router must contain a valid pricing configuration for all the
collaterals
111:             {
112:                 (,,, address resolvedOracle) =
IEulerRouter(oracle).resolveOracle(1e18, collateral, unitOfAccount);
113:                 testProperty(
114:                     adapterRegistry.isValidAdapter(resolvedOracle, block.timestamp),
115:                     ERROR__ORACLE_INVALID_COLLATERAL_ADAPTER
116:                 );
117:             }
118:
119:             // cluster vaults must have liquidation discount in a certain range
120:             uint16 maxLiquidationDiscount = IEVault(vault).maxLiquidationDiscount();
121:             testProperty(
122:                 maxLiquidationDiscount >= 0.05e4 && maxLiquidationDiscount <= 0.2e4,
ERROR__LIQUIDATION_DISCOUNT
123:             );
```

Impact

Gas savings.

Recommendation

Move the max liquidation discount check out of the loop.

Developer Response

Fixed: 0ac514b.

2. Gas - Return early to save some gas

Return early to save gas.

Technical Details

```
File: src/Swaps/handlers/BaseHandler.sol
38 |         if (params.mode == MODE_EXACT_OUT && params.receiver == address(this)) {
39 |             amountOut = balanceOut >= amountOut ? 0 : amountOut - balanceOut;
40 |         }
```

BaseHandler.sol#L39

Impact

Gas savings.

Recommendation

Return early after line 39.

Developer Response

Acknowledged, fixed in 79e9969.

3. Gas - Cache array length

Array length should be cached to save gas.

Technical Details

```
File: src/Perspectives/implementation/DefaultClusterPerspective.sol
104 |         testProperty(ltvList.length > 0 && ltvList.length <= 10,
ERROR__LTV_COLLATERAL_CONFIG_LENGTH);
105 |
106 |         // cluster vaults must have recognized collaterals
107 |         for (uint256 i = 0; i < ltvList.length; ++i) {



134 |         for (uint256 j = 0; j < recognizedCollateralPerspectives.length; ++j) {
146 |         for (uint256 j = 0; j < recognizedCollateralPerspectives.length; ++j) {
```

DefaultClusterPerspective.sol#L104-L107 DefaultClusterPerspective.sol#L134-146

Impact

Gas savings.

Recommendation

Cache the length in a variable.

Developer Response

Fixed: 5d544bb.

4. Gas - Use unchecked math if there is no overflow risk

There are math operations that can be done unchecked arithmetic for gas savings.

Technical Details

- BaseHandler.sol#L39
- BaseHandler.sol#L49
- BaseHandler.sol#L52

Impact

Gas savings.

Recommendation

Use unchecked block if there is no overflow or underflow risk for gas savings.

Developer Response

Fixed as recommended in f44abd3.

5. Gas - Cache perspective address while iterating the set of recognized perspectives

Recognized perspectives are potentially loaded twice from storage in the implementation of DefaultClusterPerspective.sol.

Technical Details

In each iteration loop, the current element `recognizedCollateralPerspectives[j]` is loaded once to compare against `address(0)` and a second time after the check.

DefaultClusterPerspective.sol#L134-L137.

```
134:             for (uint256 j = 0; j < recognizedCollateralPerspectives.length; ++j) {
135:                 address perspective = recognizedCollateralPerspectives[j] ==
address(0)
136:                     ? address(this)
137:                     : recognizedCollateralPerspectives[j];
```

DefaultClusterPerspective.sol#L146-L149.

```
146:                 for (uint256 j = 0; j < recognizedCollateralPerspectives.length;
++j) {
147:                     address perspective = recognizedCollateralPerspectives[j] ==
address(0)
148:                         ? address(this)
149:                         : recognizedCollateralPerspectives[j];
```

Impact

Gas savings.

Recommendation

Load `recognizedCollateralPerspectives[j]` once in a local variable.

```
+          address perspective = recognizedCollateralPerspectives[j]
+          perspective = perspective == address(0)
+            ? address(this)
+            : perspective;
-          address perspective = recognizedCollateralPerspectives[j] == address(0)
-            ? address(this)
-            : recognizedCollateralPerspectives[j];
```

Developer Response

Fixed: c04a9a4.

Informational Findings

1. Informational - Ineffective check in EulerDefaultClusterPerspective.sol

The preconfigured cluster perspective verifies that the given escrow perspective's name matches the intended naming, but this does not offer any substantial guarantee.

Technical Details

The constructor in EulerDefaultClusterPerspective.sol verifies that the given `escrowSingletonPerspective_` name is "Escrow Singleton Perspective". This check doesn't provide any guarantee, as any perspective can report itself under that name.

```
21:         require(
22:             keccak256(bytes(BasePerspective(escrowSingletonPerspective_).name()))
23:                 == keccak256("Escrow Singleton Perspective"),
24:             "Invalid escrow perspective"
25:         );
```

Impact

Informational.

Recommendation

Deploy EscrowSingletonPerspective.sol within EulerDefaultClusterPerspective.sol, or just remove the check and let users verify the references in `recognizedCollateralPerspectives`.

Developer Response

Acknowledged. This check is only to prevent an obvious mistake on deployment.

2. Informational - Unused errors present

An unused error is defined and can be removed.

Technical Details

```
File: src/Swaps/handlers/BaseHandler.sol


26 | error Swapper_TargetDebtBalance();
```

src/Swaps/handlers/BaseHandler.sol#L26

Impact

Informational.

Recommendation

Remove the unused error.

Developer Response

Acknowledged, fixed in ad45aa6.

3. Informational - Unused imports

Unused imports could be removed to clean up the code.

Technical Details

```
File: src/Perspectives/implementation/BasePerspective.sol

7 | import {RevertBytes} from "evk/EVault/shared/lib/RevertBytes.sol";

8 | import {IERC20} from "evk/EVault/IEVault.sol";
```

src/Perspectives/implementation/BasePerspective.sol#L7,
src/Perspectives/implementation/BasePerspective.sol#L8

```
File: src/Swaps/handlers/BaseHandler.sol

7 | import {SafeERC20Lib} from "evk/EVault/shared/lib/SafeERC20Lib.sol";
```

src/Swaps/handlers/BaseHandler.sol#L7

```
File: src/Swaps/handlers/OneInchHandler.sol

7 | import {ISwapper} from "../ISwapper.sol";
```

src/Swaps/handlers/OneInchHandler.sol#L7

```
File: src/Swaps/handlers/UniswapAutoRouterHandler.sol

7 | import {ISwapper} from "../ISwapper.sol";
```

src/Swaps/handlers/UniswapAutoRouterHandler.sol#L7

```
File: src/Swaps/handlers/UniswapV2Handler.sol

7 | import {ISwapper} from "../ISwapper.sol";
```

src/Swaps/handlers/UniswapV2Handler.sol#L7

```
File: src/Swaps/handlers/UniswapV3Handler.sol

7 | import {ISwapper} from "../ISwapper.sol";
```

src/Swaps/handlers/UniswapV3Handler.sol#L7

Impact

Informational.

Recommendation

Remove unused imports for code clarity.

Developer Response

Fixed in d6a356a.

4. Informational - Positive ramp duration does not necessarily imply an ongoing ramping process

A ramp duration greater than zero could indicate that ramping has finished and LTVs are final.

Technical Details

The DefaultClusterPerspective.sol contract verifies that the ramp duration is zero to prevent non-final LTV configurations. However, this could also indicate a previously configured ramping process that has already ended.

Impact

Informational.

Recommendation

Consider checking if the ramping has ended when the ramp duration is not zero. This issue can be alternatively mitigated by re-configuring an explicit zero ramp duration in the vault.

Developer Response

We did it on purpose, but since you pointed this out we decided to relax this condition: 7f6add0.

5. Informational - Input and output tokens are not verified in swapping paths

Technical Details

The UniswapV2Handler.sol and UniswapV3Handler.sol contracts verify that the `data` argument belongs to a well-formed route but fail to check that this path aligns with the expected input and output tokens.

Impact

Informational.

Recommendation

Consider checking that the input token matches the first address in the path and that the output token matches the last address.

Developer Response

While this would be easy to do for V2 handler, where the path is a decoded array of addresses, in case of V3 we'd need to do custom decoding. Overall though, we feel like this check is better delegated to the swap verification. We prefer to keep as is.

Final Remarks

The EVK periphery components provide additional support to the vaults in a decoupled manner, maintaining the same modular design philosophy exhibited by the EVK core contracts.

The Perspectives module stands out as the most interesting within the periphery components. Its approach is a clever design that brings great flexibility in a permissionless fashion. The on-chain nature of perspectives allows for public verification of vault configurations, enhancing trust and auditability. However, flexibility comes with associated risks, as perspectives are more difficult to reason about than a factory pattern in which the actions and configurations applied to a vault are clearer and easier to understand. A major challenge is verifying immutability and ensuring potential mutations between creation and verification are safe and observable by the perspective.

Another noteworthy element in the periphery is the Swapper design. It's built around a core contract that handles most of the trade logic while delegating critical checks to a minimal contract that acts as a safety net.

The Euler development team exhibited strong technical expertise and responsiveness throughout the review. Their prompt responses to reported issues and suggestions demonstrate a commitment to security and continuous improvement.