yAudit EVK Periphery Update Review

Review Resources:

- Custom Liquidators documentation
- Selector Access Control documentation

Auditors:

- Invader-Tak
- Adriro

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 current review focuses on the following updates to the EVK Periphery codebase: Introduction of a custom Liquidator contract that implements specific logic to handle collaterals during liquidations. This includes the SBLiquidator instance, which handles the sBUIDL token by Securitize. A selector-based access control used in GovernorAccessControl.sol and HookTargetAccessControl.sol, including additional logic for some emergency actions. The addition of an unpause functionality in the FactoryGovernor.sol contract.

The contracts of the EVK Periphery repository were reviewed over three days. Two auditors performed the code review between October 17th and October 21st, 2024. The repository was under active development during the review, but the review was limited to commit `6c74e21ead5074df9125b1511d4613b92172bfc1` for the Custom Liquidators and Selector Access Control features, and commit `ce7b05b5d5b04faec5a75693aa9b164a4da74ee1` for the updated FactoryGovernor.sol contract. After the team reviewed and fixed the initial issues, the GovernorAccessControl.sol contract underwent further refactoring, which was done in commit `4a4fdfdc805bcf735596cc03bc0a1a71a0a03b5e` and reviewed in the context of the audit.

Scope

The scope for the Custom Liquidators and Selector Access Control features included the following contracts:

```
src/AccessControl/
└── SelectorAccessControl.sol
src/Governor
└── GovernorAccessControl.sol
src/HookTarget
├── BaseHookTarget.sol
└── HookTargetAccessControl.sol
src/Liquidator
├── CustomLiquidatorBase.sol
└── SBLiquidator.sol
```

The scope for the updated FactoryGovernor.sol contract included was limited to just this file.

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	Leaked complexity from the EVC led to an incorrect integration.
Libraries	Good	The codebase relies on an updated version of the OpenZeppelin library.
Decentralization	Good	Emergency actions are only allowed under specific requirements.
Code stability	Good	The codebase remained stable during the review.
Documentation	Good	Features are accompanied by their respective documentation.
Monitoring	Good	Proper monitoring events are in place.
Testing and verification	Average	We recommended improving the tests related to custom liquidators.

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.

Medium Findings

1. Medium - Controller is incorrectly disabled in CustomLiquidatorBase.sol

The custom liquidator implementation incorrectly disables the controller by calling the EVC directly instead of routing the request through the vault.

Technical Details

The implementation of CustomLiquidatorBase.sol temporarily sets the liability vault as its controller to perform the liquidation.

```solidity
        evc.enableController(address(this), liability);

        if (isCustomLiquidationVault(collateral)) {
            // Execute custom liquidation logic
            _customLiquidation(receiver, liability, violator, collateral, repayAssets, minYieldBalance);
        } else {
            // Pass through liquidation
            liabilityVault.liquidate(violator, collateral, repayAssets, minYieldBalance);

            // Pull the debt from this contract into the liquidator
            evc.call(
                liability, _msgSender(), 0, abi.encodeCall(liabilityVault.pullDebt, (type(uint256).max, address(this)))
            );

            // Send the collateral to the receiver
            collateralVault.transferFromMax(address(this), receiver);
        }

        evc.disableController(liability);
```

The custom liquidator calls the EVC `disableController()` function to disable the controller. However, the controller vault should call this function, not the account.

Impact

Medium. Controllers for the custom liquidator account are not disabled, leading to a potential denial of service if more than one liability vault is intended to be used by the custom liquidator.

Recommendation

Route the request through the liability vault.

```diff
-   evc.disableController(liability);
+   liabilityVault.disableController();
```

Developer Response

Fixed: euler-xyz/evk-periphery@cece47d

Low Findings

1. Low - Potential overlapping of role and selector domains in SelectorAccessControl.sol

Access control is overloaded to use roles and selectors, creating a potential conflict if values for these two domains overlap.

Technical Details

The SelectorAccessControl.sol contract leverages OpenZeppelin's AccessControl library, which implements a mapping between accounts and *roles* represented as an opaque `bytes32` value.

The implementation extends this functionality by incorporating selector-based access control, where `bytes4` values corresponding to function selectors are used as roles.

A potential conflict could occur if values for roles inadvertently intersect the values of selectors. In particular, if a function has a zero value function selector, it could be confused with the default admin role, as it is represented with the zero value in the underlying library.

Impact

Low. The probability of overlapping is low.

Recommendation

Ensure there is no conflict between both domains, as this could lead to an accidental role misinterpretation.

Developer Response

Acknowledged.

2. Low - SBuidlLiquidator.sol can be used to redeem sBUIDL tokens arbitrarily

The custom liquidator can be abused to unwrap sBUIDL by transferring the tokens to the contract and calling liquidate with dummy parameters.

Technical Details

The implementation of SBuidlLiquidator.sol works by first liquidating the violator's collateral, redeeming the sBUIDL tokens from the vault, and then liquidating those tokens for its underlying (USDC).

When performing these actions, the implementation grabs whatever sBUIDL balance is present in the contract:

```
        uint256 sbTokenBalance = sbToken.balanceOf(address(this));
        sbToken.liquidate(sbTokenBalance);
```

A third-party actor can take advantage of this logic to arbitrarily unwrap their tokens by transferring them to the contract and then simulating a liquidation, leveraging the permissions granted to the custom liquidator.

Impact

Low.

Recommendation

Unwrap the portion of tokens resulting from the liquidation.

Developer Response

Fixed: euler-xyz/evk-periphery@e41f74c

Gas Saving Findings

1. Gas - Unused/Redundant variables in CustomLiquidatorBase/SBLiquidator

Technical Details

`CustomLiquidatorBase.liquidate()` makes variable declarations at the start of the function.

```
    function liquidate(
        address receiver,
        address liability,
        address violator,
        address collateral,
        uint256 repayAssets,
        uint256 minYieldBalance
    ) public callThroughEVC {
        IEVault liabilityVault = IEVault(liability);
        IEVault collateralVault = IEVault(collateral);
        ...
```

However, these variables are not used in the custom liquidation code branch. They are, however, initiated again inside the SBLiquidator implementation of the `_customLiquidation()` function.

```
    function _customLiquidation(
        address receiver,
        address liability,
        address violator,
        address collateral,
        uint256 repayAssets,
        uint256 minYieldBalance
    ) internal override {
        IEVault collateralVault = IEVault(collateral);
        IEVault liabilityVault = IEVault(liability);
        ...
```

Impact

Gas savings.

Recommendation

Move the initiation of variables into the else branch of the liquidate function.

Developer Response

Acknowledged. Fixing Medium #1 requires liabilityVault to be used outside of the else branch. Hence, keeping as is.

Informational Findings

1. Informational - `onlyEVCAccountOwner` is not enforced for emergency actions in GovernorAccessControl.sol

Emergency actions are not affected by the `onlyEVCAccountOwner` modifier.

Technical Details

The `onlyEVCAccountOwner` modifier is applied as a decorator of the `_authenticateCaller()` function.

Emergency actions implement their custom authentication logic, which directly queries the corresponding emergency role instead of using the `_authenticateCaller()` function. As a result, the `onlyEVCAccountOwner` modifier is not applied in these cases.

Impact

Informational.

Recommendation

Consider if these actions should also be restricted to the account owner when routed through the EVC.

Developer Response

2. Informational - Emergency condition in setCaps() can be triggered without decreasing the caps.

The condition for the emergency action in `setCaps()` has a redundant clause.

Technical Details

The following expression,

```
        bool isEmergency = (
            supplyCapResolved <= currentSupplyCapResolved || borrowCapResolved <= currentBorrowCapResolved
        ) && (supplyCapResolved <= currentSupplyCapResolved && borrowCapResolved <= currentBorrowCapResolved);
```

It is equivalent to `supplyCapResolved = currentSupplyCapResolved && borrowCapResolved = currentBorrowCapResolved` and doesn't necessarily imply that at least one of the caps is being decreased.

Impact

Informational.

Recommendation

```diff
    bool isEmergency = (
-       supplyCapResolved <= currentSupplyCapResolved || borrowCapResolved <=
currentBorrowCapResolved
+       supplyCapResolved < currentSupplyCapResolved || borrowCapResolved <
currentBorrowCapResolved
    ) && (supplyCapResolved <= currentSupplyCapResolved && borrowCapResolved <=
currentBorrowCapResolved);
```

Developer Response

Fixed in euler-xyz/evk-periphery@618a8ca

Final remarks

The EVK Periphery update demonstrates the capability of extending core vault functionality through specialized components. The implementation focuses on three key areas: Custom Liquidation Framework: Enables protocol-specific liquidation handling, as demonstrated by the SBLiquidator implementation for sBUIDL. Selector-Based Access Control: Provides granular permission management with emergency capabilities. Emergency Actions: Implements carefully constrained emergency functions for critical operations like LTV adjustments, vault pausing, and caps management. The codebase's complexity primarily stems from ensuring proper integration with the EVC system, particularly around controller management and access control flows. The implementation maintains clear security boundaries, especially in the swap verification system, where untrusted components are explicitly separated from core functionality. The modular design is evident in how new features like custom liquidators can be added without modifying core vault logic; however, care must be taken to maintain proper controller state management and access control invariants.