# yAudit Euler EVC Review

**Review Resources:**

- Documentation and diagrams
- evc-playground

**Auditors:**

- HHK
- NibblerExpress

## Table of Contents

## Review Summary

**Euler EVC**

The Ethereum Vault Connector (EVC) is a foundational layer designed to facilitate the core functionality required for a lending market. It serves as a base building block for various protocols, providing a robust and flexible framework for developers to build upon. The EVC primarily mediates between vaults, contracts that implement the ERC-4626 interface and contain additional logic for interfacing with other vaults. The EVC not only provides a common base ecosystem but also reduces complexity in the core lending/borrowing contracts, allowing them to focus on their differentiating factors.

The contracts of the Euler EVC Repo were initially reviewed over 8 days then an updated version was reviewed over 2 days for a total of 10 days. The code review was performed by 2 auditors between December 4th and December 19th, 2023. The review was limited to the latest commit at the start of each review. These commits were 962722a69a275c424adaa89e595ca60107376b68 for the first review and c1288b02c2c028378bde4e7bec0dddc288dd51ce for the second of the Euler EVC repo.

## Scope

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

```
src
├── interfaces
│   ├── IERC1271.sol
│   ├── IEthereumVaultConnector.sol
│   └── IVault.sol
├── Errors.sol
├── EthereumVaultConnector.sol
├── Events.sol
├── ExecutionContext.sol
├── Set.sol
└── TransientStorage.sol
```

The EVC-playground was not in scope but was used to assist in reviewing the files listed above. As discussed in informational finding 6, the EVC playground will need to be adapted to be made production ready.

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, users of the contracts agree to use the code at their own risk.

## Code Evaluation Matrix

| Category | Mark | Description |
| --- | --- | --- |
| Access Control | Good | The codebase implements a strong access control policy except for a few functions. |
| Mathematics | Good | The mathematics are simple and correctly handled. |
| Complexity | Average | The contracts are well coded and easy to read but can be complex to integrate with as multiple flows and edge cases of the EVC need to be dealt with at the vault level. |
| Libraries | Average | The codebase integrates two slightly modified functions from OpenZeppelin and two homemade `Set` and `ExecutionContext` libraries. |
| Decentralization | Good | Contract are not upgradable and there is no global admin ownership or trusted actors that haven't been chosen by the users. |
| Code stability | Good | The repository was not under active development during both reviews outside the documentation section and some tests files. |
| Documentation | Good | The contracts are documented through NatSpec, a documentation section is available on the repository and the evc-playground has examples of vaults integrating with the EVC. |
| Monitoring | Good | Multiple events are emitted through the protocol life-cycle. |
| Testing and verification | Good | 100% coverage is achieved on the EthereumVaultConnector contract, these tests includes fuzzing and invariant testing. Additionally, the team is trying to formally verify some features. |

## 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

None.

## High Findings

None.

## Medium Findings

### 1. Medium - Operators can enable and disable other operators

Operators cannot directly authorize other operators. This is the intended behavior. An operator can circumvent this protection by using the `permit()` function to call the `setAccountOperator()` function.

### Technical Details

The following proof of concept was included at the end of `test_SetOperator_Permit()` test.

```
        //Reset otherOperator to remove permissions
        vm.prank(alice);
        evc.setAccountOperator(alice, otherOperator, false);

        //Operator adds permissions for otherOperator
        data = abi.encodeWithSelector(IEVC.setAccountOperator.selector, alice,
otherOperator, true);
        signature = signerECDSA.signPermit(operator, 0, 2, block.timestamp, 0, data);
        evc.permit(operator, 0, 2, block.timestamp, 0, data, signature);
        vm.expectRevert(Errors.EVC_NotAuthorized.selector);
        //Remove the line above and include the line below to pass test on successful
attack
        //assertEq(evc.isAccountOperatorAuthorized(alice, otherOperator), true);

        //Operator removes permissions for otherOperator
        data = abi.encodeWithSelector(IEVC.setAccountOperator.selector, bob,
otherOperator, false);
        signature = signerECDSA.signPermit(operator, 0, 3, block.timestamp, 0, data);
        evc.permit(operator, 0, 3, block.timestamp, 0, data, signature);
        vm.expectRevert(Errors.EVC_NotAuthorized.selector);
        //Remove the line above and include the line below to pass test on successful
attack
        //assertEq(evc.isAccountOperatorAuthorized(bob, otherOperator), false);
```

**Impact**

Medium. The account operator can add other operators or aliases that are not authorized. For example, the owner may try to remove the account operator only to find out the account operator maintains an overlooked account. The account operator could also disable other account operators to disrupt their management of the account.

**Recommendation**

In `setAccountOperator()`, if `address(this)==msg.sender`, check that `onBehalfOfAccount` is `owner` or `operator` (e.g., `if (address(this) == msg.sender && owner != executionContext.getOnBehalfOfAccount() && operator != executionContext.getOnBehalfOfAccount() { revert EVC_NotAuthorized()}`).

**Developer Response**

Fixed - PR#67.

## Low Findings

None.

## Gas Savings Findings

### 1. Gas - Redundant check for zero address

#### Technical Details

The `permit()` function checks whether `signer == address(0)` and then calls `isSignerValid()`. The `isSignerValid()` function checks whether `signer` and `address(0)` have a common owner (i.e., `lt(xor(account, otherAccount), 0x100)`). This check is redundant to checking whether `signer == address(0)`.

#### Impact

Gas savings.

#### Recommendation

Remove the check of whether `signer == address(0)`.

#### Developer Response

Although we agree that the check is redundant at the moment, it was implemented this way to mitigate potential issues in the future. We assume that the implementation of the `isSignerValid()` function may change depending on the characteristics of the chain on which the EVC is to be deployed. This is indicated by the following inline comment: `IMPORTANT: revisit this logic when deploying on chains other than the Ethereum mainnet`. If the `signer == address(0)` check in `permit()` did not exist and, due to oversight, the `isSignerValid()` implementation changed, and no longer included that check, it would pose a high severity issue. This is because `recoverECDSASigner()` does not check the `ecrecover()` result. Therefore, we believe it is safer to keep that check as it should be present for any chain, regardless of the chain-specific `isSignerValid()` implementation.

### 2. Gas - Unchecked nonce increment and decrement

#### Technical Details

The `permit()` function increments `currentNonce` after it has checked whether `currentNonce == type(uint256).max`. The check guarantees the increment will not overflow and allows the increment to be unchecked.

Similarly, the `setNonce()` function reverts if the new nonce equals the previous nonce, so the new nonce must be greater than 0. The decrement of the nonce in the event emission can be unchecked.

#### Impact

Gas savings.

#### Recommendation

Use unchecked around the nonce increment in `permit()` and the nonce decrement in `setNonce()`.

#### Developer Response

Fixed - PR#62.

### 3. Gas - No need to check `value > 0` in `callWithContextInternal()`

#### Technical Details

The `callWithContextInternal()` function checks whether `value > 0` and whether `value > address(this).balance`. Because the balance must be at least zero, the first check is redundant.

#### Impact

Gas savings.

#### Recommendation

The check of `value > 0` could be removed. The code will be less expensive as currently written when called with `value = 0`. The code would be less expensive if revised for transactions with `0 < value < type(uint256).max`.

#### Developer Response

Fixed - PR#63.

### 4. Gas - Hardcode empty calldata in `recoverRemainingETH()`

#### Technical Details

In the function `recoverRemainingETH()`, the internal function `callWithContextInternal()` is called with `msg.data[:0]` to pass an empty calldata. This value could be hardcoded to empty strings `""`.

#### Impact

Gas savings. After testing, it seems that even with optimizer on, `msg.data[:0]` is more expensive than `""`.

#### Recommendation

Consider hardcoding an empty string `""` to save gas.

#### Developer Response

The `callWithContextInternal()` function is heavily used by several functions, including `recoverRemainingETH()`. All of these functions, except for `recoverRemainingETH()`, call `callWithContextInternal()` with user-specified calldata. To save gas by avoiding the copying of this calldata into memory, `callWithContextInternal()` is declared to accept `bytes calldata` as an input. Unfortunately, passing a hardcoded empty string `""` as suggested leads to a compilation error, as the compiler expects `calldata`, not a literal string, to be passed into the function.

```
Error (9553): Invalid type for argument in function call. Invalid implicit conversion
from literal_string "" to bytes calldata requested.
   --> src/EthereumVaultConnector.sol:519:78:
    |
519 |             callWithContextInternal(recipient, recipient, type(uint256).max, "");
    |
```

**Auditors' response:** Agree with the developer response, this gas optimization was tested in a sandbox environment where the variable didn't have to be `calldata`. The `recoverRemainingETH()` has been removed in the second version audited and thus doesn't require optimization anymore.

### 5. Gas - Code repetition in `remove()`

#### Technical Details

In the `remove()` function of the Set library, there is some code repetition between the lines 144 and 150.

`setStorage.numElements = uint8(lastIndex);` will be reach no matter the results of the different conditions and thus could be written only once after the conditions.

#### Impact

Gas savings. Improve code readability and reduce the codesize.

#### Recommendation

Consider having only one `setStorage.numElements = uint8(lastIndex);` after the `If (searchIndex != lastIndex)` condition.

#### Developer Response

We agree that the suggested recommendation could improve code readability and reduce code size. However, keeping the code unchanged allows for storage gas optimization. Writing to `firstElement` and `numElements` together (literally on consecutive lines inside the branch) saves a warm `SSTORE` and one round of bit-masking.

Implementing your suggestion would increase the gas consumption of the `remove()` function. If the removed element is `firstElement`, which is expected to be the hot path in our codebase, the gas would increase by about 200. If the removed element is at index 1, the gas would increase by about 10.

**Auditors' response:** Agree with the developer response, storage writing optimization wasn't properly taken into account when writing this finding.

## Informational Findings

### 1. Informational - Vaults should not rely on account checks when disabling controller

#### Technical Details

Account status checks are automatically passed if there are no controllers, and `disableController()` removes the controller before it asks for an account status check. Vaults cannot assume that an account status check will occur after a call to `disableController()` on the EVC.

#### Impact

Informational.

#### Recommendation

Documents for vault best practices should note the issue.

#### Developer Response

Fixed - PR#64.

### 2. Informational - Vaults must not allow state to change without reestablishing checks

#### Technical Details

The `requireAccountStatusCheckNow()` and `requireVaultStatusCheckNow()` can clear checks in the middle of a set of bundled transactions. The vault must ensure checks are reestablished if a user makes any change that could require an account status check or vault status check as part of a larger set of transactions.

#### Impact

Informational.

#### Recommendation

Documents for vault best practices should note the issue.

#### Developer Response

Current documentation states that the account and vault status checks MUST be required by the vaults whenever an operation may potentially affect account solvency or, respectively, the vault state. This implies that even if the checks were cleared in the middle of a set of bundled operations, any operation that comes after and potentially affects account solvency or vault state, MUST require relevant checks.

Improved documentation - PR#64.

### 3. Informational - Vaults should not allow calls to arbitrary contracts

#### Technical Details

A call to the EVC `callback()` function allows for a spoofed `onBehalfOfAccount`. If a vault allows for arbitrary calls, an attacker can use the vault to call the EVC `callback()` with a spoofed `onBehalfOfAccount`. The vault then receives the callback from the EVC and could behave as if the `onBehalfOfAccount` were authentic.

#### Impact

Informational.

#### Recommendation

Documents for vault best practices should note the issue.

#### Developer Response

Fixed - PR#64.

### 4. Informational - `onlyOwner()` and `onlyOwnerOrOperator()` don't confirm `onBehalfOfAccount` is authenticated

For calls coming from the EVC to itself, `onBehalfOfAccount` is used instead of `msg.sender` in `onlyOwner()` and `onlyOwnerOrOperator()`, but there is no confirmation `onBehalfOfAccount` hasn't been spoofed.

#### Technical Details

A number of functions are able to set `onBehalfOfAccount` without authenticating the caller, including `callback()` and `impersonate()`. The contract protects against attacks by not allowing `callback()`, `call()`, or `impersonate()` to call the EVC and by restricting `batch()` to delegate calls. Using `callback()` to an attack contract that delegate calls a function of the EVC allows an attacker to spoof the `onBehalfOfAccount` and have `msg.sender` be the EVC. Although the attacker is able to successfully authenticate in `onlyOwner()` or `onlyOwnerOrOperator()`, the attack fails due to being in the context of the attack contract.

#### Impact

Informational.

#### Recommendation

For multiple layers of defense, use a flag (similar to `operatorAuthenticated`) to track whether the `onBehalfOfAccount` is authentic or spoofed.

#### Developer Response

Acknowledged. The EVC relies on the stored `onBehalfOfAccount` address only if the call is

coming from the EVC itself, in other words, if the call is in the `permit()` function context. The team will use formal verification techniques to prove that:

1. The EVC can only call itself in the `permit()` function and therefore `msg.sender == EVC` is only possible in the `permit()` context.
2. The `permit()` function always sets the `onBehalfOfAccount` to the authenticated `signer` address.
3. It is not possible for the EVC to rely on the spoofed `onBehalfOfAccount` address that was set either by `callback()` or `impersonate()` (or any function other than `permit()`) because the stored value of `onBehalfOfAccount` is only used by the EVC when in the `permit()` context.

The suggested solution requires the use of an additional flag, the value of which should be considered a runtime invariant, verified when `msg.sender == EVC` whenever the EVC accesses stored `onBehalfOfAccount` address. However, if we succeed in formally proving the properties described above, the runtime assertions will no longer be necessary.

## 5. Informational - Controller vaults must be trusted/Controller phishing can drain collateral vaults

#### Technical Details

The `impersonate()` function potentially gives full control of collateral vaults to the controller and allows the controller to drain the collateral vaults.

#### Impact

Informational.

#### Recommendation

User documentation should warn about the issue.

#### Developer Response

Fixed - PR#64.

## 6. Informational - Controller must trust collateral vaults

#### Technical Details

The controller is reliant on the collateral vault calling the account status check when funds are withdrawn from the collateral vault. Notably, the EVC playground regular borrow vault does not do any checks on the collateral vault and can be drained of funds.

#### Impact

Informational.

#### Recommendation

Documents for vault best practices should note the issue.

#### Developer Response

Fixed - PR#64.

### 7. Informational - `isAccountStatusCheckDeferred()` and `isVaultStatusCheckDeferred()` will return `false` during checks callback

#### Technical Details

The `restoreExecutionContext()` that is called at the end of `callback()`, `call()`, `impersonate()` and `batch()` will clear the storage set of accounts and vaults to check before executing the callbacks. If during one of the callback to the vault or controller the contract calls the EVC to know if the checks are deferred then it will return false although the check for a given account/vault might not have been executed yet and thus could be considered as deferred.

#### Impact

Informational. If the vault or controller were to require a new check it would revert anyway as the require check's functions have a reentrancy modifier.

#### Recommendation

Consider adding a reentrancy modifier to these view functions or document in vault best practices.

#### Developer Response

Fixed - PR#65.

### 8. Informational - Missing `nonceNamespace` in `NonceUsed` event

#### Technical Details

The functions `setNonce()` and `permit()` emit the `NonceUsed` event when successfully called. This event misses the `nonceNamespace` and only emits the `nonce` used which might not be enough when indexing data.

#### Impact

Informational. The `nonce` alone might not be enough when indexing data.

#### Recommendation

Consider adding `nonceNamespace` to the `NonceUsed` event.

#### Developer Response

Fixed - PR#66.

### 9. Informational - Update removed ETH transfer guardrail

#### Technical Details

The `recoverRemainingETH()` function included protection against sending ETH to an account other than the registered owner. This protection is not present in the `call()` function of the updated contract.

#### Impact

Informational.

#### Recommendation

The contract could revert in `callWithContextInternal()` if `value != 0 && targetContract != msgSender && haveCommonOwnerInternal(targetContract, msgSender)`. It would also be reasonable to mitigate the issue in the UI or through documentation.

#### Developer Response

The now removed `recoverRemainingETH()` function was introduced to avoid complexity in recovering remaining ETH from the EVC. This complexity arose from having both `call()` and `callback()` functions, each with different `msg.sender` and target contract restrictions.

`recoverRemainingETH()` implemented protection against sending value to a non-owner account. This was considered a nice-to-have feature and was included solely because the function served only one purpose - recovering value from the EVC.

If we included the recommended revert conditions in the `callWithContextInternal()` function, it would only prevent sending value to the `msgSender`'s sub-accounts. Unfortunately, the `msgSender` would still be able to send value to any address it wants outside of its 256 accounts. Therefore, the recommended protection would not be sufficient.

Given that addressing this issue comprehensively would increase complexity, and the fact that the protection was initially introduced as a nice-to-have feature, we have decided not to take any action on this item. The documentation adequately mentions that sending not only ETH, but any token, to a non-owner sub-account may result in loss of funds.

## Final remarks

The quality of the code, extensive testing and clear documentation makes this codebase of very high quality, a lot of work and thoughts have been put into it by the Euler team.

No major security issues where found during this audit, but it's important to note that the EVC is far from easy to integrate with and external teams that would want to write vaults compatible with it will need to be aware of all its flows. To help with that the vaults specs documentation and the FAQ have been put together by the Euler team, some informational findings from this report also provide additional edge case that should be considered by integrators. Nonetheless, the auditors are confident that it will bring interesting architecture and features to the current DeFi landscape.

After our initial review, the Euler team reduced some complexity of the EVC by removing `recoverRemainingETH()` and `callback()`, this update didn't result in new findings other than one additional informational item.