OpenZeppelin | security

# Ethereum Vault Connector Audit

April 16, 2024

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2024-03-18<br>To 2024-04-03 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 14 (8 resolved, 1 partially resolved) |
| Critical Severity Issues | 0 (0 resolved) |
| High Severity Issues | 0 (0 resolved) |
| Medium Severity Issues | 0 (0 resolved) |
| Low Severity Issues | 5 (4 resolved) |
| Notes & Additional Information | 9 (4 resolved, 1 partially resolved) |
| Client Reported Issues | 0 (0 resolved) |

# Scope

We audited the `euler-xyz/ethereum-vault-connector` repository at the `69626eba206ae301b919f021e71b49feaf7ea8f5` commit.

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

# System Overview

The audited system revolves around the `EthereumVaultConnector` contract (referred to as the *EVC*), acting as a mediator for the interactions of vaults that adhere to the ERC-4626 standard. The system is designed to function as a framework for lending markets, facilitating the deposit and borrowing of assets from these vaults. The vault connector serves as a mechanism to monitor the health status of account positions and vaults. To ensure compatibility with the connector, vaults must implement a specific `IVault` interface.

Any account seeking to borrow funds must set up a controller. A controller, in this context, refers to a vault with authority over account balances across various collateral vaults. To initiate a borrowing transaction, an account must first enable one or more collateral vaults within the vault connector, thereby granting the controller control over their balances. The vault connector offers functions that communicate with the vaults to assess the health status of collateral positions, and allow controllers to manage collaterals across the vaults associated with enabled accounts.

The vault connector also provides the following features:

- **Subaccounts**: Accounts are assigned a set of 256 sub-accounts by using the last byte of the account's owner address. This allows users to isolate their borrows into separate positions.
- **Batch calls**: An account can batch several calls into a single transaction.
- **Permits**: An account can interact with the EVC through off-chain signed transactions which are then relayed on-chain by any account. This feature can be disabled for users who wish to maximize their security by disallowing the use of off-chain signatures.
- **Account operators**: An account can assign operators who can act on their behalf.
- **Lockdown mode**: An account can be locked to prevent its usage as an emergency measure.

The other in-scope contracts are satellite contracts that support these features, specifically:

The `Set` library is intended to provide the necessary data structures for `SetStorage`. This is a custom-crafted data structure that resembles an enumerable set and is optimized for gas efficiency to avoid cold value setting and enable efficient access to the first element. The `SetStorage` struct serves as the foundation for some of the variables within the

`EthereumVaultConnector`, particularly within the `TransientStorage` contract, which defines account and vault status checks using `SetStorage`.

The `TransientStorage` contract establishes an instance of the `EC` custom data type, governed by the `ExecutionContext` library. This type allows for the storage of multiple flags within a `uint256` type, aiding in determining the current execution status across various variables. These flags include identifying the `onBehalfOf` account for which the transaction is executed and indicating whether account or collateral checks are being performed.

Finally, the `EVCUtil` contract is meant to facilitate interactions with the `EVC` from vaults or other contracts, exposing useful modifiers.

# Security Model and Trust Assumptions

The `EthereumVaultConnector` is a general contract that is not meant to hold assets or native currencies by default. Any usage that implies depositing funds is solely the responsibility of integrators. With this in mind, users should be careful when it comes to interacting with integrations.

This contract is expected to hold the privileged role in the vaults contracts that integrate with it. Depending on the integration, it can be used by the controller to execute actions on behalf of the user who has enabled the given controller for any action within enabled collaterals (vaults). Therefore, it is extremely important for users to enable only controllers that are vetted and trusted, as they gain control over the user's funds.

# Low Severity

## L-01 Missing Docstrings

Throughout the codebase, there are multiple instances of code that does not have docstrings. For instance:

- The `name` state variable in `EthereumVaultConnector.sol`.
- The `version` state variable in `EthereumVaultConnector.sol`.
- The `receive` function in `EthereumVaultConnector.sol`.
- The events in `Events.sol`.
- The errors in `Errors.sol`.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Resolved in pull request #127.*

## L-02 Incomplete Docstrings

Throughout the codebase, there are several instances of incomplete docstrings. For instance:

- In the isValidSignature function in `IERC1271.sol` not all return values are documented.
- In the checkAccountStatus function in `IVault.sol` the `collaterals` parameter is not documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Resolved in pull request #128.*

## L-03 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase there are multiple floating pragma `solidity ^0.8.19` directives.

Consider using fixed pragma versions.

***Update:*** *Acknowledged, will resolve. The Euler team stated:*

> *This is a known issue. The Solidity version will be fixed before the deployment of the Ethereum Vault Connector. Note that we are still waiting for a Solidity version supporting the `transient` keyword so that we can make use of transient storage without needing to rewrite already audited libraries in assembly.*

## L-04 Collateral Removal Changes The Collaterals Order

The `reorderCollaterals` function allows users to reorder collateral according to their preferences. However, there is an issue when a user decides to remove one of the collaterals using the `disableCollateral` function. In such cases, the collateral list might be reordered using a common algorithm that swaps the last element with the removed one and then removes the last item from the list. This leads to a scenario where the `remove` function has unexpected side effects.

Consider either implementing logic to preserve the order of elements or documenting this behavior.

***Update:*** *Resolved in pull request #129. The Euler team stated:*

> *This is a known behavior of the system. The NatSpec of the `disableCollateral` function has been refined to indicate that it does not preserve the order of collaterals. Note that this behavior does not pose any risk to a user and may only cause inconvenience if the user needs to call `reorderCollateral` after previously calling `disableCollateral`. Considering that both operations can be batched in one EVC call or as a part of a bigger batch, it should never be an issue.*

## L-05 Deployment To Chains Other Than Ethereum

Below are some considerations to take into account when deploying this system to chains other than Ethereum.

- The `isSignerValid` function implements logic to prevent precompiles from being used as signers in the `permit` function. It assumes that the addresses of precompiles are values of `0xFF` or lower. This is true for Ethereum but may not be the case for other chains. For instance, on the zkSync Era chain system contract addresses start with `0x8000`.
- Typical address aliasing in chains like Arbitrum or Optimism might break the assumption of having the same prefix for different sub-accounts. Specifically, whenever an account is a contract and not an externally owned account (EOA), Layer 1 to Layer 2 (L1 -> L2) messages will have the `msg.sender` aliased with a mask. This might result in different aliased account prefixes. The EVC will need to integrate un-aliasing features if it is intended to support cross-chain messaging.
- Some chains, like zkSync Era, employ different memory growth mechanisms. In Ethereum, memory grows in words of 32 bytes each, whereas in zkSync Era, memory grows directly in bytes. Consequently, opcodes like `mstore` and `mload` may yield different results when utilized.

***Update:*** *Resolved in pull request #133 by defining the `isSignerValid` function as `virtual`. The Euler team stated:*

> *The `isSignerValid` function has been specifically added to the EVC in order to be overridden in case it is needed, depending on the chain to which the EVC is to be deployed. As you correctly noticed (i.e., for zkSync Era chain, the function will have to be overridden in order to invalidate signer addresses starting with `0x8000`), depending on the chain to which the EVC is to be deployed, other contract modifications might be necessary. Deployment to each chain will have to be closely evaluated.*

# Notes & Additional Information

## N-01 The Signature Can Be Used By Anyone

The `permit` function permits anyone to submit the signature created by the signer. While this behavior may be appropriate for many implementations, there may be cases where it is necessary to restrict the use of the signature to a specific account.

Consider whether it is worth including the submitter of the signature within the signature itself to enable the limitation of its use to the specific account.

***Update:*** *Resolved at commit ba286b2. The Euler team stated:*

> *This issue has been pointed out by other auditors as well and has been fixed under the mentioned commit.*

## N-02 Missing Converse Functions In `ExecutionContext`

In the `ExecutionContext` contract, there are functions to get values and insert values. Some functions also have a way to clear the corresponding bit of a flag but this is missing for the `checks deferred`, `check in progress`, `control collateral` and `simulation in progress` flags.

Consider providing these flags with the same functionality as the others.

***Update:*** *Acknowledged, not resolved. The Euler team stated:*

> *Acknowledged. `ExecutionContext` is a custom library and only contains functions that are directly used and needed by the EVC. No action is necessary.*

## N-03 Confusing `NonceUsed` Event

The user can utilize the `setNonce` function to invalidate a previously set nonce. This function permits setting the nonce only to a value higher than the current one. Once the new nonce is

set, the function emits a `NonceUsed` event, similar to the `permit` function. However, this process may be confusing since the user can set the nonce to a much higher value than the signature they issued, resulting in an event emission with a nonce that does not correspond to any signature.

Consider using a different event within the `setNonce` function which includes the current nonce and the value to which the nonce was increased, such as `nonceFrom` and `nonceTo`.

***Update:*** *Resolved in pull request #130.*

## N-04 Missing Input Validation

Throughout the codebase there are multiple instances of missing input validation.

- Missing zero address check for `operator` parameter in `setAccountOperator` function.
- Missing zero address check for `evc` parameter in `EVCUtil`'s contract constructor.

Consider implementing input validation for the instances above to prevent unexpected behavior.

***Update:*** *Partially resolved at commit 4118c0f. The Euler team stated:*

> *Acknowledged. Considering that the lack of input validation for the EVC functions like `setAccountOperator`, `enableCollateral` or `enableController` poses neither any risk to the user nor causes any unexpected behavior, we decided not to validate those functions inputs. No action is necessary. It lies in the user's best interest to provide correct parameters. As for the lack of address validation in the `EVCUtil` constructor, it was pointed out in the other audit and fixed under this commit.*

## N-05 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice proves beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to lack of knowledge on how to do so. Additionally, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the maintainers of those libraries to make contact with the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact. For instance:

- The `EVCUtil` abstract contract.
- The `Errors` contract.
- The `EthereumVaultConnector` contract.
- The `Events` contract.
- The `ExecutionContext` library.
- The `IERC1271` interface.
- The `IEVC` interface.
- The `IVault` interface.
- The `Set` library.
- The `TransientStorage` abstract contract.

Consider adding a NatSpec comment containing a security contact to the top of the contract's definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

***Update:*** *Resolved in pull request #131.*

## N-06 Inconsistent Use of Named Returns

The `EthereumVaultConnector` contract has inconsistent usage of named returns in its functions. To improve the readability of the contract, use the same return style in all functions.

Consider being consistent with the use of named returns throughout the codebase.

***Update:*** *Acknowledged, not resolved. The Euler team stated:*

> *Acknowledged. No action necessary.*

## N-07 File and Contract Names Mismatch

The `IEthereumVaultConnector` file name does not match the `IEVC` contract name.

To make the codebase easier to understand for developers and reviewers, consider renaming this file to match the contract name.

***Update:*** *Acknowledged, not resolved. The Euler team stated:*

> *Acknowledged. Despite the inconsistency, naming the interface differently from the file name is convenient. The file name does not use the abbreviation and clearly indicates that the Ethereum Vault Connector interface can be found in it. On the other hand, the interface contained in that file uses the abbreviation, making it more convenient to use in other smart contracts due to its shortness. No action is necessary.*

## N-08 Repeated Code in `Set` Library

Within the `Set` library, the logic to retrieve the first element in the set, and the number of elements, is repeated in many functions.

Consider encapsulating this logic into a single internal function and using it when needed. This way, if anything changes, only the internal function needs to be modified, thereby improving the overall readability and quality of the codebase.

***Update:*** *Acknowledged, not resolved. The Euler team stated:*

> *Acknowledged. The `Set` library has already been audited multiple times. Hence, unless absolutely necessary, we would not like to make any modifications in it. No action is necessary.*

## N-09 Non-explicit Imports Are Used

The use of non-explicit imports in the codebase can decrease the clarity of the code, and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity files or when inheritance chains are long.

Throughout the codebase, global imports are being used. For instance:

- The `import "../interfaces/IEthereumVaultConnector.sol";` import in `EVCUtil.sol`.
- The `import "./interfaces/IEthereumVaultConnector.sol";` import in `Errors.sol`.
- The `import "./Set.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./Events.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./Errors.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./TransientStorage.sol";` import in `EthereumVaultConnector.sol`.

- The `import "./interfaces/IEthereumVaultConnector.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./interfaces/IVault.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./interfaces/IERC1271.sol";` import in `EthereumVaultConnector.sol`.
- The `import "./ExecutionContext.sol";` import in `TransientStorage.sol`.
- The `import "./Set.sol";` import in `TransientStorage.sol`.

Following the principle that clearer code is better code, consider using named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

***Update:*** *Resolved in pull request #132.*

# Conclusions

The Ethereum Vault Connector (*EVC*) coordinates interactions between vaults using the ERC-4626 standard and provides a framework to securely participate in a lending protocol. This audit was conducted over the course of two weeks and did not reveal any significant issues. Various recommendations have been provided to enhance the quality and documentation of the codebase. The well-implemented test suite, along with the fuzzing and formal verification tests, reflects a mature codebase that the audit team appreciated. The Euler team was very supportive and answered all questions in a timely manner.