Euler EVC Security Assessment

Report Version 1.1

January 10, 2024

Conducted by the **Hunter Security** team:

George Hunter, Lead Security Researcher
deadrosesxyz, Security Researcher

Table of Contents

1 About Hunter Security

Hunter Security is a duo team of independent smart contract security researchers. Having conducted over 50 security reviews and reported tens of live smart contract security vulnerabilities, our team always strives to deliver top-quality security services to DeFi protocols. For security review inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

The Hunter Security team was engaged by Euler Labs to review the Ethereum Vault Connector (EVC) smart contracts during the period from December 20, 2023, to January 2, 2024.

Overview

Project Name	Euler, EVC
Repository	https://github.com/euler-xyz/ethereum-vault-connector
Commit hash	89dcbd4d1ad149c060d0365c244746d9b160d5e5
Resolution	5cb1ebf53a1d537db6feabc5acf139891b37a39a
Documentation	https://github.com/euler-xyz/ethereum-vault-connector/tree/89dcbd4d1ad149c060d0365c244746d9b160d5e5/docs
Methods	Manual review

Timeline

-	December 20, 2023	Code freeze & audit kick-off
v0.1	January 3, 2024	Preliminary report
v1.0	January 6, 2024	Mitigation review
v1.1	January 10, 2024	Completion date

Scope

src/Errors.sol
src/EthereumVaultConnector.sol
src/Events.sol
src/ExecutionContext.sol
src/Set.sol
src/TransientStorage.sol
src/interfaces/

Issues Found

Critical risk	0
High risk	0
Medium risk	0
Low risk	1
Informational	2

5 Consultants

George Hunter - a proficient and reputable independent smart contract security researcher with over 50 solo and team security engagements contributing to the security of numerous smart contract protocols in the past 2 years. Previously held roles include Lead Smart Contract Auditor at Paladin Blockchain Security and Smart Contract Engineer at Nexo. He has audited smart contracts for clients such as LayerZero, Euler, TraderJoe, SmarDex, Ambire, and other leading protocols. George's extensive experience and meticulous attention to detail contribute to Hunter Security's reviews, ensuring comprehensive coverage and preventing vulnerabilities from slipping through.

deadrosesxyz - a proficient and reputable bug bounty hunter with over 10 live smart contract vulnerability reports, confirmed and effectively mitigated through Immunefi. He has made significant contributions to securing protocols such as Yearn, Velodrome, Euler, SPool, and other leading DeFi protocols. His creativity and experience in hunting vulnerabilities in live protocols provide unmatched value to Hunter Security's reviews, uncovering unique vulnerabilities and edge cases that most auditors overlook.

6 System overview

The Ethereum Vault Connector is a foundational layer designed to facilitate the core functionality required for a lending market. The EVC 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 scope consists of 1 main smart contract and 1 library - the `EthereumVaultConnector` and the `Set` data structure library. The `EthereumVaultConnector` implements the main methods used to interact with the system - `call`, `permit`, `batch` and `controlCollateral`. It inherits from the `TransientStorage` abstract contract that makes use of the `ExecutionContext` library. It implements a set of useful functionalities such as gasless transactions, batching, sub-accounts, operators, simulations, etc.

6.1 Codebase maturity

Code complexity - *Moderate*
Although the `EthereumVaultConnector` contract remains compact in size, its complexity is relatively high due to the various features and the project's attempt to abstract vaults implementations.

Security - *Excellent*
The codebase has already undergone 4 reviews from independent consultants as well as Certora FV.

Testing suite - *Excellent*
The test suite is comprehensive and adheres to all best practices.

Documentation - *Excellent*
Thorough documentation encompasses all functionalities and specifications of the EVC system.

Decentralization - *Excellent*
There is no central authority. However, collaterals should trust the enabled controllers, and vice versa.

Best practices - *Excellent*
The code complies with best practices and implements numerous gas optimizations.

6.2 Privileged actors

Accounts owners

- Description: An Ethereum address that controls a total of 256 EVC accounts.
- Permissions: `setNonce`, `setOperator`, `setAccountOperator`, `enableCollateral`, `disableCollateral`, `reorderCollaterals`, `enableController`, `disableController`, `permit`, `call`, `batch`.
- Trusted by: The corresponding 256 EVC accounts determined by the owner's 19 left-most bytes.

Account operators

- Description: An address approved to execute a set of functions on behalf of a given EVC account.
- Permissions: Same as account owners except `setNonce` and `setOperator`.
- Trusted by: EVC accounts.

Controller vaults

- Description: The borrowed-from (liability) vaults.
- Permissions: `controlCollateral`, `forgiveAccountStatusCheck`, `disableController`, and functions related to the account and vault status checks.
- Trusted by: Collateral vaults (and vice versa).

6.3 Threat model

What from the EVC has value in the market?

Funds held in collateral and controller vaults. Accounts' ownership.

What are the worst-case scenarios for the EVC system?

- Breaking transient storage variables.
- Account operator executing an action on behalf of the owner or another account.
- Skipping account and/or vault status checks.
- Read-only or any re-entrancy attack.
- Denial-of-Service liquidations.
- Abusing that the msg.sender is the EVC contract when `permit` is used.
- Using an invalid signer or malleable signatures (replay attack).
- Affecting a permit message execution due to a missing parameter in the message payload.
- Calling an arbitrary contract through `call` that can cause unexpected behavior in the EVC.
- Manipulating or forgetting to update the execution context.
- Stealing funds from users mid-transaction sent through `call`.
- Executing a simulation without reverting.
- Abusing the forgive-status check.
- Using the wrong message sender (`_msgSender()`) in a given context.

What are the main entry points and non-permissioned functions?

- `permit()` - Executes signed arbitrary data by self-calling into the EVC on behalf of an account owner or operator.
- `call()` - Calls into a target contract as per data encoded.
- `batch()` - Executes multiple calls into the target contracts while checks deferred as per batch items provided.
- `controlCollateral()` - Calls into one of the enabled collateral vaults from the currently enabled controller vault as per data encoded.

6.4 Observations

Several interesting design choices were observed during the review of the `EthereumVaultConnector`, some of which are listed below:

- The EVC implements a custom TransientStorage contract. If EIP-1153 gets accepted, and transient storage becomes natively supported by the EVM before the project launches, this code may be rewritten.
- Each Ethereum address has a total of 256 accounts in the EVC distinguished by the last 1 byte of the address. While this reduces the security of addresses by 8 bits, there remains a comfortable security margin. It's important to note that if a user's private key is compromised, all accounts owned by that user will also be compromised.
- Owners can set the operator address for any account, whereas operators can only unset themselves from accounts they've been assigned to.
- Sequentially signed permit messages can be invalidated simultaneously if the owner updates the nonce for the corresponding nonce namespace.
- Collateral vaults should trust controller vaults, and vice versa.
- Account status checks are required even in `reorderCollateral` and `enableCollateral`, which do not affect an account's solvency. This abstract approach allows controller vaults to craft more customized rules.
- Disabling a controller in the EVC requires the call to come from the controller itself, not the user.
- During an external call, any address can insert itself into the account and vault status check sets if checks are deferred, potentially forcing the transaction to revert.
- Precompiles cannot be used as addresses for permit, but the validation logic should change if the contract is deployed on other chains.
- The `getPermitHash` functions writes into the Free Memory Pointer slot but clears it right after.
- The EVC is written in a highly abstracted manner to allow flexibility and avoid making specific assumptions about vaults implementations.

6.5 Useful resources

The following resources were provided by the Euler team to expedite the understanding process and define requirements for developers and auditors working on Vault contracts that interact with the Ethereum Vault Connector:

- *Frequently Asked Questions*
- *Ethereum Vault Connector and Vault Specification*
- *Whitepaper*
- *Diagrams*
- *Playground repository*
- *Video walkthrough*

7 Findings

7.1 Low

7.1.1 Account cache not cleared before status checks when restoring the EC

Severity: *Low*

Context: EthereumVaultConnector.sol#L811

Description: When an operation is executed through the `permit`, `call`, `controlCollateral` or `batch` functions, the caller can specify an `onBehalfOfAccount` address that will be used as the `_msgSender` in the nested call.

If the `targetContract` is the EVC, the `onBehalfOfAccount` address will be retrieved by calling the `_msgSender()` function:

```
function _msgSender() internal view virtual returns (address) {
    return
        address(this) == msg.sender
            ? executionContext.getOnBehalfOfAccount()
            : msg.sender;
}
```

If the `targetContract` is a vault address, the vault should retrieve the `onBehalfOfAccount` address via the `getCurrentOnBehalfOfAccount` function:

```
// @audit An example implementation taken from the evc-playground repository.
function _msgSender() internal view returns (address sender) {
    address sender = msg.sender;

    if (sender == address(evc)) {
        (sender,) = evc.getCurrentOnBehalfOfAccount(address(0));
    }
}
```

It should not be relied on the `onBehalfOfAccount` address when account and vault status checks are in progress. For that reason, the `nonReentrantChecks` modifier temporary clears the `onBehalfOfAccount` address from the execution context while the checks are being performed.

```
// @audit This modifier is applied on all public functions performing status checks.
modifier nonReentrantChecks() virtual {
    EC contextCache = executionContext;

    if (contextCache.areChecksInProgress()) {
        revert EVC_ChecksReentrancy();
    }

    executionContext = contextCache.setChecksInProgress().setOnBehalfOfAccount(
        address(0));

    _;

    executionContext = contextCache;
}
```

However, if we pass through the `permit` function and execute a `call` or `batch`, the protection would be bypassed since the `restoreExecutionContext` does not clear the set `onBehalfOfAccount` address:

```
function restoreExecutionContext(EC contextCache) internal virtual {
    if (!contextCache.areChecksDeferred()) {
        // @audit Should also do `.setOnBehalfOfAccount(address(0))`.
        executionContext = contextCache.setChecksInProgress();

        checkStatusAll(SetType.Account);
        checkStatusAll(SetType.Vault);
    }

    executionContext = contextCache;
}
```

Recommendation: Consider implementing the following change on line 811:

```
        executionContext = contextCache.setChecksInProgress().setOnBehalfOfAccount(
            address(0));
```

Resolution: Resolved. The recommended fix was implemented.

7.2 Informational

7.2.1 Unhealthy borrowers may encounter issues when enabling collateral vaults

Severity: *Informational*

Context: EthereumVaultConnector.sol#L365

Description: Whenever an operation affects an account's solvency in a controller vault, the Ethereum Vault Connector triggers the `checkAccountStatus` function in the respective controller vault(s).

This verification follows all actions that alter the set of collateral vault addresses - `enableCollateral` , `disableCollateral`, `reorderCollaterals`. Only the `disableCollateral` method could negatively impact an account's solvency. Users might encounter situations where their account is marginally unhealthy, yet not below the liquidation threshold, and fail to add more collateral. The controller could revert during the `checkAccountStatus` invocation, despite these operations potentially improving an account's solvency.

To illustrate this issue, consider the following scenario:

1. Alice has a deposit in a WETH collateral vault.
2. Her controller vault has a 110% liquidation threshold but deems positions healthy only above 125% collateral.
3. WETH's price drops, bringing Alice closer to the liquidation threshold - 115%.
4. She decides to add a WBTC collateral vault with a small deposit to prevent liquidation.
5. The addition of WBTC would raise her ratio to 120%, below the 125% healthy ratio, but an improvement for Alice.
6. The controller vault would reject the `enableCollateral` operation due to this reason, despite its intent to enhance Alice's solvency.

Recommendation: Assess the potential impact during integration with different vault implementations. Removing the account status check from `enableCollateral` and `reorderCollateral` may lead to other side effects.

Resolution: Client's response: It has been correctly noted that only one of the methods, `disableCollateral`, can negatively affect the solvency of an account. However, we intentionally invoke the `checkAccountStatus` for every function that modifies the corresponding set. This approach, combined with the fact that the EVC does not make assumptions about how controller contracts implement their `checkAccountStatus` function, allows the account status to be more abstract than the traditional solvency of the account.

7.2.2 Checks-Effects-Interactions pattern violation in the Set.forEach methods

Severity: *Informational*

Context: Set.sol#L232, Set.sol#L266

Description: The `Set.forEachAndClear` and `Set.forEachAndClearWithResult` methods remove all addresses stored in the passed storage set reference one by one, executing the provided `callback` function on each address.

The `SetStorage` struct includes a field indicating the number of elements stored in a set's array. This number resets to 0 at the start of both `Set.forEachAndClear` methods.

However, this doesn't fully adhere to the Checks-Effects-Interactions pattern, potentially enabling an external account to re-enter from the callback (depending on the function implementation). This could lead to a read-only or standard re-entrancy attack since the count has been reset to 0 while the elements remain in storage.

It's important to note that this issue doesn't affect the EthereumVaultConnector contract but represents an inherent risk when using the Set library.

Recommendation: Developers using Set structures should consider this behavior and implement re-entrancy guards in their code. There are two potential fixes:

1. Decrease the `numElements` by 1 in storage on each iteration (this would incur a significant gas overhead).
2. Reset the count to 0 (as it currently does) but clear and store all elements in a memory array at the start, then execute the `callback` on each element.

Resolution: Acknowledged. Client's response: This is known and desired behavior. A @dev comment was added for both `Set.forEachAndClear` and `Set.forEachAndClearWithResult` to emphasize that the functions do not implement the CEI pattern, hence functions using them must prevent re-entrancy.

8 Final remarks

Hunter Security attests to the quality and professional approach to security that the Euler team ensures during the development of their smart contracts for the Ethereum Vault Connector protocol. The codebase has already undergone multiple security reviews conducted by some of the most reputable and talented security consultants in the space. The team implements all best practices in their smart contracts and testing suite. Specifications and requirements for vaults integrating the EVC have been thoroughly documented by the Euler team in their whitepaper and specifications, which can be found in the `docs` directory of the repository in scope. Our team is confident in the project and the team behind it.