Euler Labs - EVC Security Review

Auditors

Christoph Michel, Lead Security Researcher

Emanuele Ricci, Lead Security Researcher

M4rio.eth, Security Researcher

Christos Pap, Associate Security Researcher

David Chaparro, Junior Security Researcher

Report prepared by: Lucas Goiriz

May 17, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Euler Labs is a team of developers and quantitative analysts building DeFi applications for the future of finance.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of ethereum-vault-connector according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**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 loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 30 days in total, Euler Labs engaged with Spearbit to review the ethereum-vault-connector protocol. In this period of time a total of **14** issues were found.

Summary

Project Name	Euler Labs
Repository	ethereum-vault-connector
Commit	c30606...cab775
Type of Project	DeFi
Audit Timeline	Apr 8 to May 10

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	14	13	1
Total	**14**	**13**	**1**

5 Findings

5.1 Informational

5.1.1 `isSignerValid` cannot be considered future-proof

Severity: Informational

Context: EthereumVaultConnector.sol#L978-L986

Description: The `isSignerValid` function is used by the `permit` function to validate if the `signer` can be considered a safe and valid signer. From the Natspec documentation and the inline comments, we can assume that:

- The function will be overridden with a custom logic given the chain on which the `EthereumVaultConnector` contract will be deployed to.
- If the `signer` and `precompiles` or `predeployes` share the same `owner` (fall into the same 19 bytes address prefix), the `signer` should be considered invalid.

The list of `precompiles` for a chain is not a static and fixed list and could change when a chain is upgraded. Because of this, the logic on which the `isSignerValid` is based on could end up not covering all the cases it should. This becomes a problem given that the `EthereumVaultConnector` is not an upgradable contract.

Recommendation: Euler should be aware that the current implementation of `isSignerValid` is not currently future-proof and should document such behavior in the Natspec.

Euler: Resolved in PR 148.

Spearbit: Verified.

5.1.2 Docs: Consider documenting explicitly that the number of collateral per account is restricted to an upper bound of `10`

Severity: Informational

Context: EthereumVaultConnector.sol#L86

Description: The `accountCollaterals` is a `mapping` state variable that maps and address to a `SetStorage` type. The `SetStorage` type can be seen as a "custom" array that at max can contain `10` elements.

Having the numbers of collateral enabled limited to a maximum number of `10` should be well documented inside the contract, interface and documentation because it could limit and influence the user's experience in certain scenarios.

Let's assume that `Alice` has enabled 10 collaterals and `Alice` has performed a borrow on `controllerVault` with the max possible LTV possible. Note that the `controllerVault` has its own logic and collateral whitelist, so it could decide to only recognize 1 of those collaterals and discard the rest (value of the discarded collaterals is 0 inside the HF).

Let's now assume that `Alice` becomes liquidable. `Alice` has 3 options:

1) Add a new collateral recognized by `controllerVault` (if not recognized, `controllerVault.checkAccountStatus` would revert).
2) Repay enough debt.
3) Supply enough collateral to the existing collaterals already recognized by `controllerVault` (if not recognized, `controller.checkAccountStatus` would revert).

Let's say that `Alice` is unable to perform action 2 and 3 because she can't repay the debt, or she can't supply existing collateral tokens.

The only possible option in such scenarios is to:

1) Perform `EVC.disableCollateral` for an existing collateral (preferably one that is not recognized by `controllerVault`).

2) Perform `EVC.enableCollateral` for a new collateral that is recognized by `controllerVault`.

Note that this operation can only be done inside an `EVC.batch` call, otherwise the `disableCollateral` operation would revert because `Alice` is already liquidable and would not be allowed to remove a collateral (it would trigger a non-deferred account-status-check).

There are additional edge-case scenarios that could create difficulties for the end user when he/she has enabled the max number of collaterals, and the limitations (of 10 collaterals) should be well documented.

Recommendation: Euler should consider documenting explicitly that the number of collateral per account is restricted to an upper bound of `10`.

Euler: Fixed in pr 149.

Spearbit: Verified.

5.1.3 The `setAccountOwnerInternal` is redundant

Severity: Informational

Context: EthereumVaultConnector.sol#L1171

Description: The `setAccountOwnerInternal` is redundant as is only used in the `authenticateCaller` function.

Recommendation: Consider removing this function and just inline the logic in the `authenticateCaller` function.

Euler: Fixed in PR 147.

Spearbit: Verified.

5.1.4 Optimization on `onlyOwner` and `setOperator`

Severity: Informational

Context: EthereumVaultConnector.sol#L115:L122, EthereumVaultConnector.sol#L115:L122

Description: The `onlyOwner` modifier and the `setOperator` function uses the same logic to authenticate a call:

- Calculate the `phantomAccount`

```
address phantomAccount = address(uint160(uint152(addressPrefix)) << ACCOUNT_ID_OFFSET);
```

- Then authenticate the call

```
authenticateCaller({account: phantomAccount, allowOperator: false, checkLockdownMode: false});
```

The `setOperator` does not use the modifier because it needs to use the `msgSender` returned by the `authenticateCaller` but both use the same 2 operations to authenticate a call which results in duplicated code.

Recommendation: It would be much cleaner to just wrap these 2 operations in an internal function that returns `msgSender` and then call that internal function in the modifier and the `setOperator` function.

Euler: Fixed in PR 146.

Spearbit: Verified.

5.1.5 Consider using `ACCOUNT_ID_OFFSET` instead of `8`

Severity: Informational

Context: EthereumVaultConnector.sol#L1128

Description: The `ACCOUNT_ID_OFFSET` is a constant used to determine the account prefix or phantom accounts. This constant is used in various places instead of the number 8 but it's missing in the `getAddressPrefixInternal`.

```
function getAddressPrefixInternal(address account) internal pure returns (bytes19) {
    return bytes19(uint152(uint160(account) >> 8));
}
```

Recommendation: Consider using the `ACCOUNT_ID_OFFSET` variable instead of `8`.

Euler: Resolved in PR 135.

Spearbit: Verified.

5.1.6 Consider improving the comments in the `setLockdownMode` and `setPermitDisabledMode`, aligning them with the natspec documentation

Severity: Informational

Context: EthereumVaultConnector.sol#L289-L290, EthereumVaultConnector.sol#L307-L308

Description: In both the `setLockdownMode` and `setPermitDisabledMode`, the mode cannot be turned off (revert) when checks are deferred and when `msg.sender == adress(this)`. Those two requirements mean that you cannot turn off the mode when the functions are executed via a `EVC.call`, `EVC.batch` or `EVC.permit`.

The inline comment:

> "to disable this mode a direct call to the EVC must be made"

could be improved in clarity by being more explicit, like how it's documented inside the relative Natspec documentation written for the very same functions inside the `IEthereumVaultConnector` interface.

Recommendation: Consider improving the inline comment inside those functions, aligning/replacing it with the one used inside the `IEthereumVaultConnector` interface.

Euler: Fixed in PR 143.

Spearbit: Verified.

5.1.7 All the contracts, libraries and interfaces are declared with a floating pragma

Severity: Informational

Context: Global scope

Description: All the contracts, libraries and interfaces are declared with a floating pragma: `pragma solidity ^0.8.19;`. The floating pragma issue is mitigated by the fact that the Solidity version, which will be used at build time, is locked at `0.8.24` inside `foundry.toml`.

Nevertheless, it's both a good dev and security practice to specify the locked pragma that should be used directly in the contracts, libraries and interfaces.

Recommendation: Euler should lock the solidity pragma of the contracts, interfaces and libraries to the one specified in the `foundry.toml` configuration time.

Euler: Acknowledged. 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.

Spearbit: Acknowledged.

5.1.8 Natspec: the `operator` natspec for `setOperator` and `setAccountOperator` is outdated

Severity: Informational

Context: IEthereumVaultConnector.sol#L182-L183, IEthereumVaultConnector.sol#L193-L194

Description: In both the functions Natspec documentation, it's stated that the `operator` cannot be equal to `address(0)`. This check is not implemented in the actual code, and the Euler team has indeed confirmed that the documentation is outdated.

Recommendation: Euler should update the outdated Natspec documentation relative to the `operator` parameter for the `setOperator` and `setAccountOperator` functions. Both functions allow the `operator` to be equal to `address(0)`.

Euler should also consider homogenizing the documentation about the `operator` parameter in both functions to clear any confusion. For example, in the `setAccountOperator` the `operator` is defined as "The address of the operator that is being installed or uninstalled". This definition seems like the `operator` is required to be a smart contract and not an EOA while in the codebase such requirement is not enforced.

Euler: Fixed in PR 144.

Spearbit: Verified.

5.1.9 Natspec: improve and clarify the natspec documentation about the `enabled` parameter for the `setPermitDisabledMode` function

Severity: Informational

Context: IEthereumVaultConnector.sol#L157-L165

Description: The `setPermitDisabledMode` allows the owner of an `addressPrefix` to **disable** the `permit` functionality. When `enabled=TRUE` the `permit` function will be disabled (user is enabling the disable-mode), when `enabled=FALSE` the `permit` function will be enabled (user is disabling the disable-mode).

The current Natspec for the `enabled` parameter states:

> `/// @param enabled A boolean indicating whether to enable or disable permit functionality.`

By just reading such documentation, an external actor could assume that passing `enabled=true` would enable the `permit` function, while in reality, it's totally the opposite.

Recommendation: Euler should consider rewriting and improving the Natspec documentation for the `enabled` parameter of the `setPermitDisabledMode` function.

Euler: Fixed in PR 145.

Spearbit: Verified.

5.1.10 `callThroughEVC` ABI encoding does not right-pad `data` parameter with zeroes

Severity: Informational

Context: EVCUtil.sol#L43

Description: The `callThroughEVC` modifier encodes the following call using custom code:

```
function call(address targetContract, address onBehalfOfAccount, uint256 value, bytes calldata data)
        external
        returns (bytes memory result);
```

According to the ABI argument encoding, it should right-pad the `bytes data` parameter with zeroes until it's a multiple of 32.

> followed by the minimum number of zero-bytes such that len(enc(X)) is a multiple of 32.

However, the custom assembly code does not do this, in contrast to what `abi.encodeCall(EVC.call, (address(this), msg.sender, 0x0, data))` would do.

Recommendation: This violation of the ABI specification does currently not interfere with Solidity's calldata decoding. Fixing it is therefore not required and might add unnecessary complexity. If the gas optimization of a custom assembly encoding is preferred to `abi.encodeCall`, we recommend integration testing this function for the Solidity version that will be used in deployment to ensure compatibility.

Euler: Fixed in PR 142.

Spearbit: Verified.

5.1.11 Lockdown mode is checked before authenticating caller

Severity: Informational

Context: EthereumVaultConnector.sol#L763-L786

Description: The `authenticateCaller({ account, checkLockdownMode: true })` function checks the lockdown mode of the provided `account` even before checking if the caller is allowed to operate on behalf of `account`. The code will revert with `EVC_LockdownMode` if the account is in lockdown mode, instead of `EVC_NotAuthorized` if the caller is not authorized.

Recommendation: The proper error in this scenario should be the `EVC_NotAuthorized` because if the account is in lockdown mode should only be relevant if the caller is allowed to operate on it.

Euler: Fixed in PR 141.

Spearbit: Verified.

5.1.12 `batchSimulation`'s `result` array's length increased

Severity: Informational

Context: EthereumVaultConnector.sol#L668-L673

Description: The `batchSimulation` function calls `batchRevert` that returns `EVC_RevertedBatchResult(batchItemsResult, accountsStatusCheckResult, vaultsStatusCheckResult);`. This return data is copied to the `result` array and the inner arrays are being decoded via:

```
(batchItemsResult, accountsStatusCheckResult, vaultsStatusCheckResult) =
            abi.decode(result, (BatchItemResult[], StatusCheckResult[], StatusCheckResult[]));
```

For this to work, the 4-byte selector needs to be skipped. The code does this by simply pointing the `result` array pointer 4 bytes ahead:

```
assembly {
    result := add(result, 4)
}
```

However, this changes the `result` array's length as it now includes the selector in addition to its initial length. The array length will be `> 2**32`. The`abi.decode` function currently still works as it only checks that, while decoding, the read memory offsets are not out of the array bounds (using the wrong result length). As `batchRevert` always correctly encodes its return data and the new `result` length is always greater than the original result length, this check does not cause issues.

Recommendation: We recommend setting the proper array length as the compiler does not give any guarantees on how its `abi.decode` implementation will use the array length. The only guarantee is that it won't read out-of-bounds elements, but if it would, for example, access the last element, the memory expansion cost with the inflated length would be too high and revert the call.

```
if (success) {
    revert EVC_BatchPanic();
} else if (result.length < 4 || bytes4(result) != EVC_RevertedBatchResult.selector) {
    revertBytes(result);
}

assembly {
    let length := mload(result)
    // skip 4-byte EVC_RevertedBatchResult selector
    result := add(result, 4)
    // write new array length = original length - 4-byte selector
    // cannot underflow as we require result.length >= 4 above
    mstore(result, sub(length, 4))
}
```

Euler: Fixed in PR 140.

Spearbit: Verified.

5.1.13 Docs: Lockdown mode specification violations

Severity: Informational

Context: EthereumVaultConnector.sol#L74-L75

Description: The Lockdown mode specification currently reads:

> Once Lockdown Mode activated, the EVC MUST significantly reduce its functionality across the Owner's Accounts. In this mode, the Owner is restricted to managing Operators and Nonces, while authorized Operators are restricted to revoking their own permissions. With Lockdown Mode active, neither the Owner nor the Operators can carry any other operations on the EVC. Notably, calling external smart contracts on behalf of the Owner or any of their Accounts is prohibited. However, enabled Controllers can still control collaterals for the Accounts, even under lockdown.

To summarize, only `setNonce`, `setOperator`, `setAccountOperator` (and `controlCollateral`/`forgiveAccountStatusCheck`/`disableController` as it's the controller triggering these) should be callable.

However, it's currently possible to use `call`/`batch` functions to call back to `msg.sender` as it skips the `authenticateCaller(checkLockdown=true)` checks. This technically violates "Notably, calling external smart contracts on behalf of the Owner or any of their Accounts is prohibited".

The owner can also call `setLockdownMode` to turn it off and `setPermitDisabledMode`.

Recommendation: Consider adding the exceptions to the specification. Consider explicitly listing the functions that should still be callable by a specific role in lockdown mode.

Euler: Fixed in docs.

Spearbit: Verified.

5.1.14 Docs: Improve reentrancy specification

Severity: Informational

Context: EthereumVaultConnector.sol#L155-L188

Description: The reentrancy specification currently reads:

> Only the Checks-Deferrable Call functions and Permit function MUST be allowed to re-enter the EVC.

Recommendation: Consider providing more background / context for EVC reentrancy. Generally, one can always reenter any function on the EVC as long as no checks or control collateral is in progress. This is, for example, because of nested batch calls, the way `permit` is implemented, and (nested) vaults that need to be able to call back to the EVC. One is mostly concerned about reentering the EVC while an important account state is being accessed. Therefore, all the "checks-deferrable call functions + permit" and `require*Check` functions will revert if checks or control collateral are in progress.

Euler: Fixed in docs.

Spearbit: Verified.