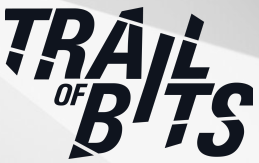

Euler Labs Ethereum Vault Connector

Security Assessment (Summary Report)

January 30, 2024

Prepared for:
Erik Arfvidson
Euler Labs

Prepared by: **Michael Colburn and Simone Monica**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Sam Greenup, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Simone Monica, Consultant
simone.monica@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
January 4, 2024	Pre-project kickoff call
January 17, 2024	Delivery of report draft
January 17, 2024	Report readout meeting
January 30, 2024	Delivery of summary report with fix review appendix

Executive Summary

Engagement Overview

Euler Labs engaged Trail of Bits to review the security of the smart contracts in the `ethereum-vault-connector` repository at commit `8e1b9a7`. The Ethereum Vault Connector (EVC) provides a base level of smart contract infrastructure designed to mediate interactions between ERC-4626 vaults so that one or more lending protocols can be built on top of the EVC.

A team of two consultants conducted the review from January 8 to January 12, 2024, for a total of two engineer-weeks of effort. With full access to source code, documentation, and examples in the `evc-playground` repository, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

The primary focus was a thorough review of the authentication and access control mechanisms and logic. We reviewed the authentication flows to identify ways that they could be bypassed or ways that the permissions could be granted or revoked unexpectedly. We also looked for ways that the account or vault status checks could be bypassed. Though we referenced the example applications in the `evc-playground` repository, these were considered out of scope and used only to improve our understanding of the system, not reviewed for security concerns.

In addition to the issues already present in the github repository, we observed two minor inconsistencies between the white paper, the specification, and the implementation. Though the codebase involves some complex authentication flows, the functionality and expected behavior are thoroughly documented, which helps make the system easier to understand. The Euler Labs team has also put significant effort into applying advanced testing techniques, such as fuzzing and formal verification, to the codebase, which helps demonstrate that it behaves as expected and will help prevent introducing regressions in future developments.

Recommendations

We encourage the Euler Labs team to correct the discrepancies that were identified in the white paper, the specification, and the implementation and to continue to develop fuzz tests and formal verification for important system invariants.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The codebase itself includes only simple math, primarily for looping and manipulating nonces. Trivial unchecked blocks are used for incrementing these values.	**Satisfactory**
Auditing	The contracts emit events for all critical state-changing operations. Since the `EthereumVaultConnector` contract is not upgradeable and does not include any global admin role, any monitoring or incident response efforts will likely be limited to communication with end users.	**Satisfactory**
Authentication / Access Controls	Implementing proper access controls on vault connections is core to the EVC's functionality. Proper access control checks appeared to be in place for any privileged functionality. Each account has different roles, such as the owner, various sub-accounts, and operators, with varying levels of access that are clearly documented in the system specification.	**Satisfactory**
Complexity Management	While the codebase is fairly complex for its size, it is accompanied by extensive documentation and a thorough specification that explains all the potential code paths.	**Satisfactory**
Decentralization	The `EthereumVaultConnector` contract itself has no global admin role with special privileges and is not upgradeable. However, this setup may not extend to protocols building on top of the EVC.	**Strong**

Documentation	The repository includes both a white paper that describes the system in plain language at a high level and a more formal specification that describes the requirements of the various components. The codebase includes thorough NatSpec comments and additional in-line comments when necessary to explain more complex logic.	**Strong**
Low-Level Manipulation	The `EthereumVaultConnector` contract uses assembly blocks in a few places. Two multiline blocks with explanatory comments are used as part of the permit function's validation, and three additional single-line statements are used elsewhere in the contract. These other lines are not overly complex but would benefit from comments for consistency. The return values of any low-level calls in the codebase are properly checked. The system relies heavily on bit manipulation operations (i.e., bit shifting and bit masking), but the bulk of these are abstracted away into the separate `ExecutionContext` library.	**Satisfactory**
Testing and Verification	The main `EthereumVaultConnector` contract has thorough test coverage. The Euler Labs team has also implemented advanced testing techniques like fuzzing, added Scribble annotations, and begun working on formal verification.	**Strong**
Transaction Ordering	We did not identify any risks related to transaction ordering.	**Satisfactory**

A. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

B. Custom Slither Lint

During the security review, we wrote a custom Slither script to check for some of the rules present in the Vault Specification that the vault implementations should adhere to.

The script in figure B.1 implements the following rules:

- The vault contract must inherit the `VaultBase` contract.
- The `doTakeVaultSnapshot`, `doCheckVaultStatus`, and `disableController` functions are implemented.
- The `msg.sender` variable is never used directly; instead the values returned by the `_msgSender` or `_msgSenderForBorrow` functions from the `EVCClient` contract should be used.

The script can be used as a base to possibly implement more detailed rules and/or remove false positives after an accurate evaluation (at the moment, the last rule can produce false positives).

```
import sys
from slither import Slither
from slither.core.declarations import Function, FunctionContract, FunctionTopLevel
from slither.core.declarations.solidity_variables import SolidityVariableComposed
from typing import List


def check_euler_vault():
    if len(sys.argv) != 3:
        print("Two arguments are needed: python check-euler-vault.py file.sol|.
vault_contract_name")
        exit()

    slither = Slither(sys.argv[1], ignore_compile=True)
    contracts = slither.get_contract_from_name(sys.argv[2])
    if len(contracts) == 0:
        print(f"Contract {sys.argv[2]} not found")
        exit()

    contract = contracts[0]

    # Check that the Vault inherits the VaultBase contract
    found = False
    for c_inherited in contract.inheritance:
        if c_inherited.name == "VaultBase":
            found = True
            break
```

```
    if found:
        print(f"[✓] {contract.name} inherits VaultBase")
    else:
        print(f"[] {contract.name} inherits VaultBase")

    # Check that doTakeVaultSnapshot, doCheckVaultStatus, disableController are
implemented
    doTakeVaultSnapshot = False
    doCheckVaultStatus = False
    disableController = False
    for f in contract.functions:
        if f.name == "doTakeVaultSnapshot" and not f.is_empty:
            doTakeVaultSnapshot = True
        elif f.name == "doCheckVaultStatus" and not f.is_empty:
            doCheckVaultStatus = True
        elif f.name == "disableController" and not f.is_empty:
            disableController = True

    if doTakeVaultSnapshot:
        print(f"[✓] {contract.name} implements doTakeVaultSnapshot function")
    else:
        print(f"[] {contract.name} implements doTakeVaultSnapshot function")

    if doCheckVaultStatus:
        print(f"[✓] {contract.name} implements doCheckVaultStatus function")
    else:
        print(f"[] {contract.name} implements doCheckVaultStatus function")

    if disableController:
        print(f"[✓] {contract.name} implements disableController function")
    else:
        print(f"[] {contract.name} implements disableController function")

    # Check that msg.sender is never used instead _msgSender or _msgSenderFromBorrow
from EVCClient should be used
    for f in contract.functions_entry_points:
        if not f.name == "constructor" and SolidityVariableComposed("msg.sender") in
_all_solidity_variables_read(f):
            print(f"[x] Function {f.name} is using msg.sender. Consider if it should
use _msgSender or _msgSenderFromBorrow from the EVCClient contract")


# Modified from
https://github.com/crytic/slither/blob/e3dcf1ecd3e9de60da046de471c5663ab637993a/slit
her/core/declarations/function.py#L1077-L1109
# Return all the solidity variables read from a function and does not consider
functions and modifiers from the EVCClient contract
def _all_solidity_variables_read(function) -> List:
    values = function.solidity_variables_read
    explored = [function]
    to_explore = [
        c for c in function.internal_calls if isinstance(c, FunctionTopLevel) or
(isinstance(c, FunctionContract) and c.contract_declarer.name != "EVCClient") and c
```

```
not in explored
    ]
    to_explore += [
        c for (_, c) in function.library_calls if isinstance(c, Function) and c not
in explored
    ]
    to_explore += [m for m in function.modifiers if m not in explored and
m.contract_declarer.name != "EVCClient"]

    while to_explore:
        f = to_explore[0]
        to_explore = to_explore[1:]
        if f in explored:
            continue
        explored.append(f)

        values += f.solidity_variables_read

        to_explore += [
            c
            for c in f.internal_calls
            if (isinstance(c, FunctionTopLevel) or isinstance(c, FunctionContract)
and c.contract_declarer.name != "EVCClient") and c not in explored and c not in
to_explore
        ]
        to_explore += [
            c
            for (_, c) in f.library_calls
            if isinstance(c, Function) and c not in explored and c not in to_explore
        ]
        to_explore += [m for m in f.modifiers if m not in explored and m not in
to_explore and m.contract_declarer.name != "EVCClient"]

    return list(set(values))


if __name__ == "__main__":
    check_euler_vault()
```

Figure B.1: A Slither script to check some vault-specific rules.

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On January 26, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Euler team in the following pull requests: PR #77, PR #79, and PR #81.

PR #77
Resolved. In the `restoreExecutionContext` function, when the checks are not deferred, the `onBehalfOfAccount` address is now correctly reset to `address(0)`.

PR #79
Resolved. The `requireAccountStatusCheckNow` and `requireAllAccountsStatusCheckNow` functions have been removed to reduce the possible attack surface. As a result, a vault cannot ask for an immediate account or vault check, but these will always be checked at the end.

PR #81
Resolved. Moving of the nonce update in the `permit` function after the signature has been validated to correctly implement the behavior expected from the specifications.