



Euler Labs EthereumVaultConnector

Security Assessment (Summary Report)

January 17, 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

© 2024 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be business confidential information; it is licensed to Euler Labs under the terms of the project statement of work and intended solely for internal use by Euler Labs. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the [Trail of Bits Publications page](#). Reports accessed through any source other than that page may have been modified and should not be considered authentic.

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

About Trail of Bits	1
Notices and Remarks	2
Table of Contents	3
Project Summary	4
Executive Summary	5
Codebase Maturity Evaluation	6
A. Code Maturity Categories	8
B. Custom Slither Lint	10

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 **Simone Monica**, Consultant
michael.colburn@trailofbits.com 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

Executive Summary

Engagement Overview

Euler Labs engaged Trail of Bits to review the security of its **Ethereum Vault Connector smart contracts** at commit 8e1b9a7. The Ethereum Vault Connector (EVC) contract is a base level of smart contract infrastructure designed to mediate interactions between ERC-4626 vaults such that one or more lending protocols can be built on top of it.

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 thorough review of the authentication and access control mechanisms and logic. We reviewed the authentication flows to try to identify any ways that they could be bypassed or that 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 only used to improve our understanding of the system and were not reviewed for security concerns.

We did not identify any issues as a result of our review. We did note two minor inconsistencies between the whitepaper, specification, and implementation. Though the codebase involves some complex authentication flows, functionality and expected behavior is thoroughly documented which does help make the system easier to understand. The Euler Labs team has also put a lot of effort into applying advanced testing techniques, such as fuzzing and formal verification, to the codebase, which helps to demonstrate that it behaves as expected and to prevent introducing regressions in future development.

Recommendations

We encourage the Euler Labs team to correct the discrepancies that were identified in the specification and implementation and to continue to work on developing fuzz tests and formal verification of 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 only includes 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. As the EVC 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 which 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 of the potential code paths.	Satisfactory
Decentralization	The EthereumVaultConnector contract itself has no global admin or owner role with special privileges and is not upgradeable. Note that this may not extend to protocols building on top of EVC.	Strong
Documentation	The repository includes both a whitepaper that describes the system in plain language at a high level, as well as a more formal specification that describes the requirements of the various components. The codebase	Strong

	includes thorough NatSpec comments as well as additional in-line comments when necessary to explain more complex logic.	
Low-Level Manipulation	<p>The EVC contract uses assembly blocks in a few places. Two multi-line 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 too complex, but would benefit from comments for consistency.</p> <p>The return values of any low-level calls in the codebase are properly checked. The system relies heavily on various bit manipulation operations (i.e., bit shifting and bit masking), but the bulk of this is abstracted away into the separate <code>ExecutionContext</code> library.</p>	Satisfactory
Testing and Verification	The main EVC contract has thorough test coverage. The Euler Labs team has also implemented advanced testing techniques like fuzzing, Scribble annotations, and is 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 specs](#) that the vault implementations should adhere to.

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

- It must inherit `VaultBase`
- the `doTakeVaultSnapshot`, `doCheckVaultStatus`, `disableController` functions are implemented
- `msg.sender` is never used directly, instead `_msgSender` or `_msgSenderFromBorrow` from `EVCCClient` should be used

This script can be used as a base to possibly implement more detailed rules and/or remove false positives where present after an accurate evaluation (at the moment the last rule implemented 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 EVCCClient 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 EVCCClient contract")

# Modified from
https://github.com/crytic/slither/blob/e3dcf1ecd3e9de60da046de471c5663ab637993a/slither/core/declarations/function.py#L1077-L1109
# Return all the solidity variables read from a function and does not consider
functions and modifiers from the EVCCClient 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 != "EVCCClient") 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 != "EVCCClient"]

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 != "EVCCClient") 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 != "EVCCClient"]

    return list(set(values))

if __name__ == "__main__":
    check_euler_vault()

```

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