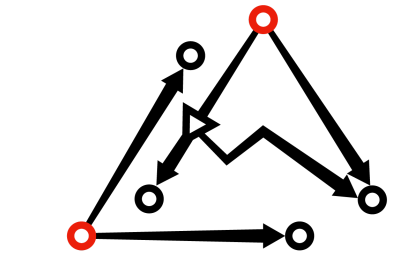

Least Authority

PRIVACY MATTERS

VM Module

Security Audit Report

# Conflux Network

Final Audit Report: 31 July 2025

# Table of Contents


- Overview
  - Background
  - Project Dates
  - Review Team
- Coverage
  - Target Code and Revision
  - Supporting Documentation
  - Areas of Concern
- Findings
  - General Comments
    - System Design
    - Code Quality
    - Documentation and Code Comments
    - Scope
  - Specific Issues & Suggestions
  - Suggestions
    - Suggestion 1: Clarify Specification and CIP Process
- About Least Authority
- Our Methodology

# Overview

## Background

Conflux Network has requested Least Authority perform a security audit of the VM Module for the Conflux Chain.

## Project Dates

- **June 27, 2025 - July 21, 2025:** Initial Code Review *(Completed)*
- **July 23, 2025:** Delivery of Initial Audit Report *(Completed)*
- **31 July, 2025:** Verification Review *(Completed)*
- **31 July, 2025:** Delivery of Final Audit Report *(Completed)*

## Review Team

- Will Sklenar, Security Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

# Coverage

## Target Code and Revision

For this audit, we performed research, investigation, and review of the VM Module for the Conflux Chain followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- conflux-rust:
  https://github.com/Conflux-Chain/conflux-rust
    - Only the VM Module
        - crates/execution:
            - vm-interpreter
            - vm-types
            - geth-tracer
            - executor

Specifically, we examined the Git revision for our initial review:

- `055fbb5e52b59560d8e34907f6ef78d5efc44a3f`

For the review, this repository was cloned for use during the audit and for reference in this report:

- conflux-rust:
  https://github.com/LeastAuthority/conflux-rust

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

## Supporting Documentation

The following documentation was available to the review team:

- Conflux documentation: https://doc.confluxnetwork.org
- Project's modular structure: https://github.com/Conflux-Chain/conflux-rust/blob/master/crates/client/src/common/mod.rs#L139
- Conflux Yellow Paper: https://confluxnetwork.org/files/Conflux_Protocol_Specification.pdf
- Summary of Code Migration for Conflux Execution Layer | Notion: https://gaudy-hub-f32.notion.site/Summary-of-Code-Migration-for-Conflux-Execution-Layer-1d87e025eade8064b546ef37a0659a3c
- Website: https://confluxnetwork.org

In addition, this audit report references the following documents:

- Conflux Improvement Proposals: https://github.com/Conflux-Chain/CIPs
- Ethereum Improvement Proposals: https://github.com/ethereum/EIPs

## Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

# Findings

## General Comments

Conflux is an EVM-compatible layer 1 blockchain. While Conflux retains full EVM bytecode compatibility in its Ethereum space, the protocol has been extended with collateral-based storage economics, dual account spaces, and sponsorship mechanics. In this audit, we reviewed Conflux's VM implementation.

Conflux operates in two modes: e-space (EVM bytecode compatible) and core-space, which includes further Conflux-specific functionality. The e-space section of the VM is largely a port of the open EVM

codebase. Our team checked the implementation of the EVM opcodes for parity with open EVM and did not identify any vulnerabilities, or errors related to stack handling or arithmetic operations.

## System Design

The Conflux VM consists of a three-phase execution model. The first phase (`FreshExecutive / PreCheckedExecutive`) verifies the transaction, including verifying the signature, nonce, and CIP-7702 authorizations. The next phase of the execution is a 256-bit stack interpreter with a journaling substate. Subsequently, the third phase is the finalization phase, which (among other responsibilities) handles gas refunds and updates token counters.

Each execution frame utilizes a checkpoint / revert API, to handle frame rollbacks inexpensively. Substate tracks logs, refunds, and accessed accounts, with substate merging occurring bottom-up. Our team audited the checkpoint-management subsystem for atomicity and consistency across success and failure modes and did not find any vulnerabilities or invariance violations.

Conflux provides an EVM implementation with a different Consensus design. The EVM implementation is based on a fork of OpenEthereum, which is no longer maintained. Conflux also incorporates code from Revm, an actively maintained Rust EVM implementation.

We conducted a comparison of the implementation against the specification, and found that Conflux inherits some specification limitations from Ethereum. The Ethereum specification (the “Yellow Paper”) is no longer updated, so in Ethereum, the specification must be understood as the Yellow Paper combined with the EIPs. A similar situation exists in Conflux, but instead of EIPs there are CIPs. The CIPs are a collection of numbered documents that describe changes to various components of the system. As a result, a full understanding of the system requires both the Yellow Paper and the complete set of finalized CIPs. Consequently, the process imposes a significant burden on an auditor to understand the entire system.

Conflux aims for practical compatibility with the EVM. However, some differences are unavoidable due to a different Consensus system and block time. Arbitrary Ethereum contracts are not guaranteed to run on Conflux, but it is likely that Ethereum contracts can be ported to Conflux. Several CIPs (150, 151, 152, 154, 156, 645, 7702, 165) target compatibility between Conflux eSpace and the EVM. We paid particular attention to verifying the implementation of these CIPs.

Of note is CIP-7702, for which EIP-7702 is the Ethereum counterpart. CIP-7702 allows externally owned accounts to execute smart contract code, enabling them to behave similarly to contract accounts. This is achieved by allowing EOAs to delegate to a smart contract. We reviewed CIP-7702 and related CIPs and did not identify any vulnerabilities with the implementation.

One aspect of CIP-7702 is that one account can initialize many new “empty” EOAs by setting the authority for several uninitialized accounts. While the new accounts would be essentially empty in that they would contain no funds, they nevertheless take up storage with their account headers. As a result, CIP-7702 could introduce a vulnerability by allowing an inexpensive way to bloat the state trie by creating many empty “auth accounts.” To address this issue, CIP-7702 leverages Conflux’s system of storage collateral. Any transaction that would result in a new empty account being created is required to provide storage collateral as dictated by the value `PER_EMPTY_ACCOUNT_COST`. By charging collateral through this approach, the attack vector is mitigated by making the attack expensive to carry out.

As part of this evaluation, we also checked that the gas calculations matched CIP-465, and found them to be consistent with the specification.

#### Dependencies

We examined the dependencies implemented in the codebase and noted that some cryptographic components have been forked from external repositories, which may complicate maintenance or the ability to apply future updates. We generally recommend using libraries directly to facilitate integration of future improvements and security patches. Additionally, if it is necessary to use custom cryptographic implementations or forked or otherwise nonstandard libraries, we strongly recommend performing an audit specifically focusing on the custom cryptographic libraries.

### Code Quality

We performed a manual review of the repositories in scope and found that Conflux's codebase is well-structured and modular, with core logic organized into clearly named modules. Additionally, the code is generally well-written and is idiomatic Rust.

#### Tests

The repositories in scope include substantial test coverage, comprising both internal unit tests written in Rust and integration tests written in Python, which evaluate the implementation via the RPC interface.

### Documentation and Code Comments

Conflux has a specification (forked from the Ethereum Yellow Paper) and detailed CIPs for each additional feature added. In many parts of the codebase, the code references the CIPs (or EIPs) through variable names, feature flags, and comments. This is especially valuable for understanding where particular CIPs are implemented. However, it could be further improved by maintaining alignment between the specification and the CIPs (Suggestion 1). Additionally, the codebase is moderately well-commented, which was helpful in understanding the intended functionality of most components.

### Scope

The scope of this audit was limited to the correctness of the VM, conducted in preparation for the upcoming hardfork. As part of this assessment, we paid particular attention to the ongoing CIPs. Other components such as the Consensus protocol, P2P protocol, and cryptographic libraries, were out of scope for this review.

## Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

| ISSUE / SUGGESTION | STATUS |
|---|---|
| Suggestion 1: Clarify Specification and CIP Process | Planned |

# Suggestions

## Suggestion 1: Clarify Specification and CIP Process

**Location**

Conflux-Chain/CIPs/blob/master/README.md

**Synopsis**

Conflux is primarily a fork of Ethereum, intended to allow contracts developed for Ethereum to operate within its environment. The Conflux specification is likewise derived from the Ethereum specification, and, as a result, inherits the issue that the specification (the "Yellow Paper") has not been kept up to date with the implementation. Therefore, to fully understand the Conflux system, it is necessary to consult the Yellow Paper and the Conflux Improvement Proposals (CIPs) together. However, the CIPs exist only as a collection of numbered documents. It is not possible to determine the specific CIPs that pertain to a particular aspect of the system. For example, if one wishes to verify that all the opcodes have been correctly implemented, ideally there would be a single, comprehensive list containing every opcode along with the complete description of their behavior. In practice, however, this information is distributed across individual CIPs, requiring each to be consulted. The Conflux Improvement Proposal `README` describes an approval process by which the specification is updated to include accepted CIPs, but this process has not been consistently followed.

**Mitigation**

We recommend updating the specification to include changes introduced by CIPs, as described in the CIP process, or at a minimum referencing the relevant CIPs in the corresponding sections of the specification.

**Status**

The Conflux Network team has stated that they will schedule time to gradually improve the Specification document, and noted that even the Ethereum Specification document (Yellow Paper), despite showing a recent date, has not incorporated updates from the past two years.

**Verification**

Planned.

# About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

## Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

## Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

## Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

## Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

## Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.