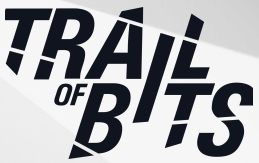

Offchain Security Council Rotation Update

Security Assessment (Summary Report)

March 12, 2025

Prepared for:
Harry Kalodner, Lee Bousfield, Steven Goldfeder, and Ed Felten
Offchain Labs

Prepared by: **Gustavo Grieco and Tarun Bansal**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@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:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Tarun Bansal, Consultant
tarun.bansal@trailofbits.com

Project Timeline

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

Date	Event
February 20, 2025	Delivery of report draft
February 28, 2025	Report readout meeting
March 12, 2025	Delivery of final summary report

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of the changes made to the governance contracts to allow council members to rotate their own keys without a proposal. These changes correspond to PR #322 (`57f495c`).

The commits in scope involve changes that resolve previously detected issues in an earlier version of the implementation, along with a new feature to allow upcoming council members to set a new key when the cohort is replaced. Only the production smart contract changes were in scope.

A team of two consultants conducted the review from February 13 to February 19, 2025, for a total of eight engineer-days of effort. With full access to source code and documentation, we performed a manual review of the code in scope.

Observations and Impact

This engagement revealed an issue related to the removal of an upcoming council member. Additionally, we provide some recommendations for improving the code quality in the Code Quality Recommendations appendix.

Recommendations

Based on the security review, Trail of Bits recommends that Offchain Labs take the following steps:

- Remediate the finding disclosed in this report as part of a direct remediation or any refactor that may occur when addressing other recommendations.
- Review the items in the Code Quality Recommendations appendix and consider taking action on each one.

1. Member removal does not offer protection when the cohort has already been elected

Severity: **Low**	Difficulty: **High**
Type: Timing	Finding ID: TOB-SC-ROT-1
Target: `SecurityCouncilManager.sol`	

Description

The security council member removal code does not correctly handle the case where the election has already ended and the cohort is queued for changing, allowing malicious members to temporarily evade removal.

Security council members can perform key rotation using two different options. The first one allows members to rotate the key directly:

```
   function rotateMember(
        address newMemberAddress,
        address memberElectionGovernor,
        bytes calldata signature
    ) external {
        uint256 lastRotatedTimestamp = lastRotated[msg.sender];
        if (lastRotatedTimestamp != 0 && block.timestamp < lastRotatedTimestamp +
minRotationPeriod)
        {
            revert RotationTooSoon(msg.sender, lastRotatedTimestamp +
minRotationPeriod);
        }
        // we enforce that a the new address is an eoa in the same way do
        // in NomineeGovernor.addContender by requiring a signature
        uint256 currentRotationNonce = rotationNonce[msg.sender];
        address newAddress = ECDSAUpgradeable.recover(
            getRotateMemberHash(msg.sender, currentRotationNonce), signature
        );
        // we safety check the new member address is the one that we expect to
replace here
        // this isn't strictly necessary but it guards agains the case where the
wrong sig is accidentally used
        if (newAddress != newMemberAddress) {
            revert InvalidNewAddress(newAddress);
        }
```

```
        ..
```

Figure 1.1: The `rotateMember` function

The second option offers the opportunity to execute the rotation indirectly when the cohort is changed, storing the address to rotate to in the `rotatingTo` mapping. This is executed some time after the election is resolved, when the cohort is replaced:

```
   function replaceCohort(address[] memory _newCohort, Cohort _cohort)
        external
        onlyRole(COHORT_REPLACER_ROLE)
    {
        if (_newCohort.length != cohortSize) {
            revert InvalidNewCohortLength({cohort: _newCohort, cohortSize:
cohortSize});
        }

        // delete the old cohort
        _cohort == Cohort.FIRST ? delete firstCohort : delete secondCohort;
        address[] storage otherCohort = _cohort == Cohort.FIRST ? secondCohort :
firstCohort;

        for (uint256 i = 0; i < _newCohort.length; i++) {
            // we have to change the array so correct _newCohort can be emitted
            address rotatingAddress = rotatingTo[_newCohort[i]];
            if (rotatingAddress != address(0)) {
                // only replace if there is no clash
                if (
                    !SecurityCouncilMgmtUtils.isInArray(rotatingAddress, _newCohort)
                        && !SecurityCouncilMgmtUtils.isInArray(rotatingAddress,
otherCohort)
                ) {
                    _newCohort[i] = rotatingAddress;
                }
            }
            _addMemberToCohortArray(_newCohort[i], _cohort);
        }

        _scheduleUpdate();
        emit CohortReplaced(_newCohort, _cohort);
    }
```

Figure 1.2: The `replaceCohort` function

Address rotation needs to be implemented correctly to avoid allowing a potentially malicious member to evade removal. The process for removing a security council address includes some code to specifically prevent this evasion. This includes keeping track of the

previously rotated address, in order to correctly remove it when the `removeMember` function is called:

```
function removeMember(address _member) external onlyRole(MEMBER_REMOVER_ROLE) {
    if (_member == address(0)) {
        revert ZeroAddress();
    }
    address memberIfRotated = memberRotatedTo(_member);

    Cohort cohort = _removeMemberFromCohortArray(memberIfRotated);
    _scheduleUpdate();
    emit MemberRemoved({member: memberIfRotated, cohort: cohort});
}
```

Figure 1.3: The `removeMember` function

However, when the address is rotated indirectly using `setRotatingTo`, there is no mapping to track who rotated that key, nor any way to avoid the key removal if a security member key is compromised.

Exploit Scenario

Alice's address is approved for the security council election. The voting takes place, and Alice is elected. The call to `replaceCohort` is scheduled for execution. Eve compromises Alice's private key. The current security council or DAO decides to remove Alice from the security council. They deploy an action to remove Alice's address, to be executed once the cohort is changed. Eve calls `setRotatingTo` at the very last minute, before the execution of `replaceCohort`. The security council / DAO need to take action on the new Eve address.

Recommendations

Short term, document this behavior to make sure the relevant actors (e.g., the security council or the DAO itself) are aware of the removal process during the process of the cohort change.

Long term, review the incident response plan and procedures across components to make sure all the relevant situations are covered and there is adequate infrastructure to perform them.

A. Code Quality Recommendations

The following is a list of findings that were not identified as immediate security issues but may warrant further investigation.

- The `rotatingTo` mapping entries are never removed. This can introduce vulnerabilities with future updates that use the `rotatingTo` mapping.
- Be careful when using the `addMember` and `replaceMember` functions, as the `_addMemberToCohortArray` internal function removes the `rotatedTo` mapping entry for the newly added member. This can lead to unexpected behavior if a member is added again after being rotated away.

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.