MixBytes()

Euler EVK Periphery Security Audit Report

APRIL 22, 2025

Table of Contents

1. Introduction

1.1 Disclamer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

The CapRiskSteward contract is a risk management contract designed to provide controlled and safe adjustments to vault parameters in the Euler protocol. It allows authorized users to modify supply and borrow caps within predefined safety limits, implementing a time-based cooldown system where adjustments are restricted by a maximum factor (1.5x) that gradually recharges over a 3-day period. The contract also includes functionality to update interest rate models, but only accepts models deployed by a recognized factory.

This security audit was conducted over a 2-day period.

The primary objective of the audit was to ensure that this contract indeed provides an additional layer of protection when modifying critical vault parameters. Apart from the findings presented in the report, we examined the following attack vectors and potential issues.

Key aspects of the risk management contract. Necessary checks are in place for the new supplyCap and borrowCap values. The adjustment factor recharges correctly according to CHARGE_INTERVAL, and allowedAdjustFactor is calculated without precision loss. Authentication for the setCaps and setInterestRateModel functions is properly enforced.

Integration of the contract into the overall project architecture. We verified that functions from the CapRiskSteward contract can be correctly invoked via the EVC when called from an account owner's address. The AmountCap library is correctly used to unwrap the currentCap and nextCap parameters passed to the _validateCap function. However, special cases related to the absence of cap limits are not handled correctly, as outlined in the findings section. The IRM newModel is correctly validated using the isValidDeployment function, ensuring that it was previously deployed and configured.

User security. We confirmed that updating the IRM model does not negatively impact users' positions in the protocol or cause an immediate deterioration in health metrics.

Additionally, the audit highlighted the following system behaviors:

- The _validateCap() function enforces an imbalanced range check [66%; 150%], meaning it allows cap values to increase more rapidly than they decrease.
- The checkVaultStatus() function of the RiskManagerModule may be affected by updated cap values. If caps are reduced, this function could start reverting. However, this does not appear to pose a significant risk to the protocol's stability.

The audit identified only low-severity issues. Addressing these findings with the recommended improvements will enhance the protocol's efficiency and reliability.

The audit was focused specifically on:
- The contract's access control mechanisms
- The cap adjustment logic and validation
- The time-based cooldown system
- The IRM update validation
- The message data handling and contract interaction
- Any potential reentrancy vulnerabilities
- The error handling and revert conditions
- The assembly code usage in _targetContract()
- The mathematical operations in cap validation

1.3 Project Overview

Summary

Title	Description
Client Name	Euler
Project Name	EVK Periphery
Type	Solidity
Platform	EVM
Timeline	02.04.2025 - 14.04.2025

Scope of Audit

File	Link
src/Governor/CapRiskSteward.sol	CapRiskSteward.sol

Versions Log

Date	Commit Hash	Note
02.04.2025	a438c1e910f7292adbd527b4ea72742d3f736f23	Initial Commit

Date	Commit Hash	Note
04.04.2025	dfaf4e9499c78d4eee42437f0e9a6e70fcf2cc3b	Commit for the Re-audit
14.04.2025	f311fa49f4442643c4473ac46b520b1f40e99dcb	Commit with updates

Mainnet Deployments

File	Address	Blockchain
CapRiskSteward.sol	0xe934ff...Ad27B048	Ethereum Mainnet
CapRiskSteward.sol	0xB6D8eb...8d6601D4	Base
CapRiskSteward.sol	0x0A6aB7...7Ccefe5A	Swell

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: • Cross-check findings among auditors • Discuss identified issues • Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** • The client addresses the identified issues and provides feedback • Auditors verify the fixes and update their statuses with supporting evidence • A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** • Verify the final code version against recommendations and their statuses • Check deployed contracts for correct initialization parameters • Confirm that the deployed code matches the audited version • Issue a public audit report, published on our official GitHub repository • Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50-60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10-20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	0
Medium	0
Low	3

Findings Statuses

ID	Finding	Severity	Status
L-1	Incomplete Validation of Calldata Length in _targetContract	Low	Acknowledged
L-2	Incomplete Cap Safety Check Against MAX_SANE_AMOUNT	Low	Fixed
L-3	Cap Validation Failure for Vaults With No Limits	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Incomplete Validation of Calldata Length in `_targetContract`		
Severity	Low	**Status**	Acknowledged

Description
The `_targetContract` function extracts the `address` of the vault contract from the calldata by assuming the last 20 bytes represent the address. While there's a check that `msg.data.length > 20`, this check allows possible manipulations of the calldata which can bring additional attack vectors in the future. This function requires exactly 88 bytes to process the call (4 bytes (selector) + 32 (uint16) + 32 (uint16) + 20 (address)).
This opens a path for potential calldata manipulation or malformed call attempts, especially when relying on calldata decoding for internal logic.

Recommendation
Enhance the validation logic by enforcing strict size:

```
if (msg.data.length != 88) revert MsgDataInvalid();
```

Client's Commentary:
Acknowledge, won't fix. This attack vector is theoretical and can only be realized in a highly specific setup. Furthermore, state-changing functions in CapRiskManager are behind access control mechanisms, so the caller is assumed to be trusted.

L-2	Incomplete Cap Safety Check Against MAX_SANE_AMOUNT		
Severity	Low	**Status**	Fixed in dfaf4e94

Description
Within _validateCap, caps are validated only based on a dynamic allowedAdjustFactor. However, **hard sanity limits** (MAX_SANE_AMOUNT) will still be enforced in downstream contracts like the **GovernanceModule**, leading to **unexpected reverts** even if the steward's validation passes.

Recommendation
We recommend adding explicit validation for absolute sanity limits:

```
// For supply case
if (nextCapResolved > maxCap ||
    nextCapResolved < minCap ||
    nextCapResolved > 2 * MAX_SANE_AMOUNT) {
    revert CapAdjustmentInvalid(vault);
}

...

// For borrow case
if (nextCapResolved > maxCap ||
    nextCapResolved < minCap ||
    nextCapResolved > MAX_SANE_AMOUNT) {
    revert CapAdjustmentInvalid(vault);
}
```

Client's Commentary:
Will fix by limiting new caps to MAX_SANE_AMOUNT.

L-3	Cap Validation Failure for Vaults With No Limits		
Severity	Low	**Status**	Fixed in dfaf4e94

Description

This issue has been identified within the _validateCap function of the CapRiskSteward contract.
If a vault currently does not have a limit, the function AmountCap.wrap(currentCap).resolve() returns type(uint256).max. Consequently, when calculating currentCapResolved * allowedAdjustFactor / 1e18, the operation CapRiskSteward.sol#L134 due to an overflow.
The issue is classified as **Low** severity because it can cause unintended revert, preventing valid cap adjustment.

Recommendation

We recommend implementing a check to handle scenarios where currentCap is not set.

Client's Commentary:

In the no limit case, there could be no valid cap adjustment, because the domain of caps is [0, uint112.max] U {uint256.max}. Any sane adjustment factor ($2^{60-2}70$) would not allow adjusting from uint256.max to the sane range regardless.
Anyway, we are going to fix this by handling the unlimited scenario specifically, for the purposes of code cleanliness.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes