EULER REWARD-STREAMS SECURITY AUDIT REPORT

May 16, 2024

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The `reward-streams` module of the Euler v2 protocol allows users to incentivize ANY token (including Euler v2 vault shares) with ANY token for a particular amount of time. The module facilitates a fully permissionless setup of rewards for any vault. Vault creators can choose between 2 options for vault share staking: shares can be staked manually by users by deposing shares to the protocol or by automatic hooks triggered for every balance update of a user in a vault.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Euler
Project name	reward-streams
Timeline	09 May 2024 - 16 May 2024
Number of Auditors	3

Project Log

Date	Commit Hash	Note
09.05.2024	95783173eff4668bb177eeefd9e8a9dc1a44669c	Commit for the audit
16.05.2024	92b946d4a089f03ac56a0735b92f1270bab215da	Commit for the reaudit

Project Scope

The audit covered the following files:

File name	Link
src/BaseRewardStreams.sol	BaseRewardStreams.sol
src/StakingRewardStreams.sol	StakingRewardStreams.sol
src/TrackingRewardStreams.sol	TrackingRewardStreams.sol

Deployments

Deployed contracts will be verified after the proposal is approved by the DAO.

1.5 Summary of findings

Severity	# of Findings
Critical	0
High	0
Medium	0
Low	7

ID	Name	Severity	Status
L-1	Unprotected hook	Low	Acknowledged
L-2	`claimSpilloverReward` is called without a state update	Low	Fixed
L-3	Transfer of poisoned tokens to any address	Low	Acknowledged
L-4	Strange reward accounting	Low	Acknowledged
L-5	DoS of popular tokens	Low	Fixed
L-6	Dust will get stuck on the contract	Low	Acknowledged
L-7	No events emitted inside the `balanceTrackerHook` function	Low	Fixed

1.6 Conclusion

In this audit, our primary objective was to assess the system's robustness and reliability from both security and operational perspectives. The main goals were to verify the fairness of the reward distribution and ensure that no vulnerabilities exist that could block dependent components of Euler v2 Vaults. We also focused on ensuring the accurate handling of edge cases that may occur during reward distribution.

The audit considered the following key aspects of the system, along with potential attack vectors and points of failure:

1. **Reward Distribution Accuracy**. When adding rewards, epoch boundaries are correctly set and checked. Users cannot add rewards for epochs that start too far in the future. Users receive rewards according to their share in eligible tokens. Users cannot claim their rewards multiple times as the `lastUpdated` variable is updated correctly. When the `block.timestamp` coincides with the epoch boundary, the distribution is carried out correctly. Before all operations that change the number of eligible tokens, an update is performed. Decimals are correctly accounted for in all calculations. The accuracy of rounding during token distribution is enhanced with the `SCALER` multiplier. However, some dust remains as a result of rounding. Recommendations have been given regarding this. Additionally, recommendations have been provided concerning `reward` tokens distributed without eligible tokens.

2. **Integration into the Euler ecosystem**. The project offers two reward distribution scenarios: via staking `rewarded` tokens and via invoking a hook by Euler Vault when the `rewarded` token balance changes. The second option implies a deeper integration of the reward stream. The audit found no possibility of blocking the hook call by the reward stream contract, even during position liquidation in the Vault.
   The hook call is public. Although this does not pose any issues in the current implementation, a recommendation has been made to restrict the caller. Interaction with EVC occurs correctly if the call happens on behalf of some address.

3. **Security of the trustless approach**. Users can add any `reward` token for any `rewarded` token. Both types of tokens can include poisoned tokens intended to disrupt the system's functionality. The system proved resilient to such tokens: the existing distribution of tokens cannot be blocked or broken by a malicious user.
   If a malicious user credits dirty money in a token with a blacklist (e.g., USDC) as a `reward`, the blacklisting of the contract will not block the entire protocol. Only the rewards of that token will be paused until a resolution from the token's governance.

4. **Potential DoS scenarios**. If an update has not been performed for a long period, the loop for each epoch will consume a lot of gas. However, this will not block the protocol. There are also no `require` or `assert` blocks that could lead to a contract DoS.

5. **Security of the optimizations**. The code is optimized for storage usage and contains many type-casting operations. All related invariants were found correct. Additionally, all unchecked blocks were verified. Underflow/overflow is impossible for all calculations.

This project demonstrates a high level of security and operational integrity. The audit identified only low-level problems. Addressing these issues with the recommended improvements will enhance the protocol's efficiency and reliability. Overall, the contracts are well-designed to provide secure and accurate reward distribution, ensuring the smooth operation of the Euler protocol.

2.FINDINGS REPORT

2.1 Critical

Not Found

2.2 High

Not Found

2.3 Medium

Not Found

2.4 Low

L-1	Unprotected hook
Severity	Low
Status	Acknowledged

Description

The `balanceTrackerHook` is currently not protected by a modifier and can be called by any actor: TrackingRewardStreams.sol#L34. Even though this vulnerability cannot be directly exploited, there remains a possibility that future integrations and protocol contracts can be susceptible to exploitation through this. Therefore, it is common practice to restrict the system as much as possible.

Recommendation

We recommend adding a check that the `msg.sender` is registered in the EVC market.

Client's commentary

> Acknowledged. Both the `TrackingRewardStreams` and the `StakingRewardStreams` distributor contracts are designed to allow anyone to incentivize any rewarded token (in both staking and non-staking manners) with any reward token. This design treats both contracts as public goods that can be utilized by other projects aiming to incentivize their users. For `StakingRewardStreams`, no integration effort is required, making it straightforward to use. However, `TrackingRewardStreams` requires integrating smart contracts to include a hook call when a user's balance changes. This design philosophy necessitates that there is no privileged list of contracts that can call the `balanceTrackerHook` function. It is important to note that the smart contract is designed to be safe for the `balanceTrackerHook` to be called by any actor, as the `msg.sender` can only affect the storage associated with its own address.

L-2	`claimSpilloverReward` is called without a state update
Severity	Low
Status	Fixed in 92b946d4

Description

`claimSpilloverReward` is called without a state update, which can leave some unclaimed amount on the zero address: BaseRewardStreams.sol#L257. Furthermore, the `earnedReward` function may return a larger amount for the zero address than what will be claimed by `claimSpilloverReward`, potentially leading to integration issues.

Recommendation

We recommend calling `updateRewardInternal` in the `claimSpilloverReward` function.

Client's commentary

> Fixed. The `claimSpilloverReward` function has been removed altogether and additional `recipient` parameter has been added to the `updateReward` function in order to allow user claim the spillover rewards virtually accrued to `address(0)`.
> PR-18

L-3	Transfer of poisoned tokens to any address
Severity	Low
Status	Acknowledged

Description

The permissionless nature of the protocol allows a malicious user to transfer any token to any address from the whitelisted address: BaseRewardStreams.sol#L523.

Recommendation

We recommend requiring an explicit approval from the recipient to receive any token.

Client's commentary

> Client: Acknowledged. It's important to note that tokens are only transferred to a user if `accountEarned.claimable > 0`, indicating that the user has willingly enabled that token as a reward. Given the permissionless nature of the distributor, it is the user's responsibility to avoid enabling tokens that could be considered malicious.
> MixBytes: `claimReward` allows a malicious actor to pass any `recipient` address, which can cause the transfer of a poisoned token to any address from the `StakingRewardStreams` and `TrakingRewardStreams` contracts. Because poisoned tokens can be directly sent to any address, this problem is not severe. Still, if some off-chain service tracks all transfers from the `StakingRewardStreams` and `TrakingRewardStreams` contracts (e.g., some CEX), then it can block the contract.
> Client: I acknowledge the scenario described, where a malicious actor could potentially use the `claimReward` function to send tokens to any recipient address. However, it's important to note that this does not pose a significant security threat, as poisoned tokens can be sent directly to users outside of the reward stream distributors. The use of reward stream distributors for this purpose would primarily constitute a griefing attack, aimed at causing inconvenience or damaging the reputation of the distributor contracts by associating them with malicious activities. Therefore, while the issue is worth noting, it does not necessitate immediate changes to the contract's functionality, given its limited impact and the nature of the attack.

L-4	Strange reward accounting
Severity	Low
Status	Acknowledged

Description

Currently, if a reward token was deposited before any user enabled it, it will be lost (claimed by any user): BaseRewardStreams.sol#L562. This behavior is strange because it would be more beneficial to redistribute such an amount of tokens to further epochs.

Recommendation

We recommend redistributing the tokens that couldn't be claimed to further epochs.

Client's commentary

> Acknowledged. The behavior described is indeed an edge case. However, addressing it would significantly increase the complexity of the codebase and disrupt several existing assumptions. One key assumption is that rewards are only registered through the `registerReward` function and do not automatically roll over to subsequent epochs. Another is that the amount distributed in a current epoch remains constant. Addressing this edge case would necessitate substantial modifications to the current, straightforward code structure. Additionally, it's important to note that the `registerReward` function mandates that any registered reward stream must start in a future epoch, at least in the subsequent one, which practically allows users time to enable the reward before its distribution begins.

L-5	DoS of popular tokens
Severity	Low
Status	Fixed in 92b946d4

Description

Well-known tokens can be maliciously deposited to newly created vaults as a reward, so when vault managers decide to set rewards, they will need to pay a lot of gas to skip empty epochs: BaseRewardStreams.sol#L171.

Recommendation

We recommend adding information about such behavior to the documentation.

Client's commentary

> Fixed. The documentation has been updated to sufficiently describe this behavior.
> PR-18

L-6	Dust will get stuck on the contract
Severity	Low
Status	Acknowledged

Description

If the decimals of `reward` and `rewarded` tokens differ (e.g., 18 decimals and 6 decimals), the calculation of the `accumulator` will always lead to some dust being stuck on the contract without the ability to claim it: BaseRewardStreams.sol#L620. The magnitude of this problem will increase over time as the dust accumulates on the contract.

Recommendation

We recommend redistributing dust to further epochs. This can be done by comparing the token's balance to the amount that should be distributed to users in further epochs. This approach will also distribute possible token donations to users.

Client's commentary

> Client: Acknowledged and partially fixed. The documentation has been updated to clearly describe this behavior, and calculations have been refined to minimize precision loss under certain conditions. However, it's important to note that precision loss is inherent in Solidity due to its use of fixed-point arithmetic. The suggested solution of redistributing dust to future epochs is not feasible. Implementing this would require iterating over all users who have interacted with a specific rewarded-reward pair, comparing the total registered reward amount against the sum of individual earned amounts. Such an implementation is impractical in smart contract environments due to the extensive computational and gas cost implications.
> PR-18
> MixBytes: Indeed, the fix presented in PR-18 makes this problem even less severe, but it doesn't solve it completely, so the status is ACKNOWLEDGED.

L-7	No events emitted inside the `balanceTrackerHook` function
Severity	Low
Status	Fixed in 92b946d4

Description

There is a `balanceTrackerHook` function defined at the line TrackingRewardStreams.sol#L30. It updates rewards data, `distributionStorage.totalEligible` and `accountStorage.balance` storage variables but doesn't emit any corresponding events.

Recommendation

We recommend emitting an event at the end of the function mentioned. It can log the updated `distributionStorage.totalEligible` and `accountStorage.balance` variables.

Client's commentary

> Fixed. The `Staked` and `Unstaked` events have been removed and replaced by a unified event called `BalanceUpdated`. This event is now emitted from the `stake`, `unstake`, and, as recommended, the `balanceTrackerHook` function. However, contrary to the recommendation, information about the total eligible amount is not logged because it depends on the enabled rewards set of the account.
> PR-18

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts


https://github.com/mixbytes/audits_public


https://mixbytes.io/


hello@mixbytes.io


https://twitter.com/mixbytes