Euler, Reward Streams Security Audit

Report Version 1.0

May 9, 2024

Conducted by **Hunter Security**:

George Hunter, Lead Security Researcher
Windhustler, Senior Security Researcher
Stormy, Associate Security Researcher

Table of Contents

1 About Hunter Security

Hunter Security consists of multiple teams of leading smart contract security researchers. Having conducted over 100 security reviews and reported tens of live smart contract security vulnerabilities protecting over $1B of TVL, our team always strives to deliver top-quality security services to DeFi protocols. For security audit inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

The Hunter Security team was engaged by Euler to review the Reward Streams contracts during the period from May 1, 2024, to May 6, 2024.

Overview

Project Name	Euler, Reward Streams
Repository	https://github.com/euler-xyz/reward-streams
Commit hash	967c73de100620b83adce8897dc6b67bc74aac1c
Resolution	95783173eff4668bb177eeefd9e8a9dc1a44669c
Methods	Manual review & testing

Timeline

-	May 1, 2024	Audit kick-off
v0.1	May 7, 2024	Preliminary report
v1.0	May 8, 2024	Mitigation review

Scope

src/BaseRewardStreams.sol
src/StakingRewardStreams.sol
/src/TrackingRewardStreams.sol
src/interfaces/IBalanceTracker.sol
src/interfaces/IRewardStreams.sol

Issues Found

High risk	0
Medium risk	1
Low risk	2

5 Findings

5.1 Medium

5.1.1 BaseRewardsStreams does not support tokens with significantly different decimals and prices

Severity: Medium

Context: BaseRewardStreams.sol#L616

Description: The *BaseRewardStreams* contract allows to register any number of reward tokens per rewarded (staking) token. Different ERC20 tokens can have different decimal configurations and USD value per token. An extreme case would be if:

- Rewarded token is SHIB, that has 18 decimals and a price of $0.00002111
- Reward token is USDC, that has 6 decimals and a price of $1.00

By observing the *calculateRewards* function we can notice:

- *delta* (amount of reward tokens) is calculated by multiplying time elapsed per epoch, *rewardAmount* and *SCALER* (1e12)
- *accumulator* is updated by adding *delta* divided with the total eligible amount of rewarded tokens

$500 USD worth of *SHIB* is *500 / 0.00002111 * 1e18 = 2.3e25 SHIB* tokens.

On the other hand, $500 worth of *USDC* distributed as rewards during 15 seconds in an epoch 10 days long is: *1e12 * 15 * 500 * 10e6 / 864000 = 8.6e15 USDC* tokens.

This makes the division of *delta* with the total eligible amount of rewarded tokens to be zero, thus the accumulator will never get updated, and the rewards will never be distributed.

Even if we have two tokens with the same decimals, the accumulator can round to zero if their price difference is too big.

Recommendation: The *SCALER* should be higher than *1e12* to avoid such errors. Adjust the other storage variables with the change of the *SCALER* accordingly.

Resolution: Resolved.

5.2 Low

5.2.1 Setting rebasing tokens in a reward scheme can result in an internal logic error and potential denial of claiming the funds from the contract itself

Severity: *Low*

Context: BaseRewardStreams.sol#L503-L512

Description: Take as example that the stETH token was used in order to successfully create a reward scheme, the problem here is that the balance of the stETH token changes through time - it could lead to a greater balance once there is a positive rebase or to a less balance when a negative one occurs.

As a result the balance of the actual contract may be greater or lower than the actually balances recorded through the contract. In case of a negative rebase the stETH token will be less in the contract balance leading to a DoS of reward funds, this results as the system fully claims the recorded rewards and reset them to zero.

Recommendation: Consider whether rebasing tokens should be supported and implement proper accounting.

Resolution: Resolved.

5.2.2 Missing check for the maximum epoch duration

Severity: *Low*

Context: BaseRewardStreams.sol#L123

Description: During deployment, the *BaseRewardStreams* contract is initialized with the *EPOCH_DURATION*. The minimum duration is above 7 days, however there is no check for the maximum duration. As the constructor parameter *_epochDuration* has a type of *uint48*, that is the maximum value that can be passed to the contract. If the epoch duration were to be initialized with such a high value, the reward rate per epoch would be extremely low. Depending on the decimals configuration of reward and rewarded tokens, the rewards might round zero.

Recommendation: Consider adding a maximum duration check to prevent such scenarios.

Resolution: Resolved.