yAudit Euler ERC20 Wrapper Locked - Review

Review Resources:

- Codebase (PR)

Auditors:

- HHK
- Panda

Table of Contents

Review Summary

**Euler ERC20 Wrapper Locked **

The Euler EVK periphery ERC20 Wrapper Locked provides a mechanism for wrapping ERC20 tokens with a time-based unlocking schedule, depending on whether accounts are whitelisted. When whitelisted, users can freely move and withdraw their tokens, while when they're not, their tokens unlock as per the unlocking schedule.

The contracts of the Euler EVK periphery ERC20 Wrapper Locked PR were reviewed over two days. Two auditors performed the code review between 2 October and 4 October 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit a6a8024b90b334806c0d99b7bab3b10b45a74bc5 for the Euler EVK periphery ERC20 Wrapper Locked PR.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/ERC20/deployed/RewardToken.sol
src/ERC20/implementation/ERC20WrapperLocked.sol
```

After the findings were presented to the Euler team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Euler and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Implements proper access control with owner and whitelist mechanisms.
Mathematics	Good	No complex math formulas. Usage of unchecked math without risks.
Complexity	Good	Contract logic is straightforward and easy to follow.
Libraries	Good	Utilizes well-audited OpenZeppelin libraries for core functionality.
Decentralization	Average	The code allows significant control by the contract owner, who can modify user lock schedules. While this provides flexibility in management, it introduces centralized points of control over user's assets.
Code stability	Good	Code appears stable.
Documentation	Good	Comprehensive NatSpec comments.
Monitoring	Good	Emits events for key actions. One event was missing.
Testing and verification	Good	Code is properly tested.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions

that are outside the scope of the requirements.

- Gas savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

1. Low - Potential loss of funds in early withdrawals

Users may unintentionally forfeit a portion of their funds by withdrawing tokens before the full unlock period has elapsed, as the contract does not prevent early withdrawals or warn users of potential losses.

Technical Details

The `withdrawToByLockTimestamp` and `withdrawToByLockTimestamps` functions in the `ERC20WrapperLocked` contract allow users to withdraw tokens based on specific lock timestamps. However, these functions do not prevent withdrawals before the entire unlock period has passed. As a result, users who withdraw early will receive fewer tokens than they would if they waited for the whole of the unlock period, effectively losing a portion of their funds.

Impact

Low.

Recommendation

Add a boolean parameter defaulted to false that allows for a withdrawal with loss. This would require users to explicitly acknowledge and accept the potential loss of funds when withdrawing early.

Developer Response

Fixed in https://github.com/euler-xyz/evk-periphery/pull/110/commits/371cea6eea51b81b38470d922fd1f929999e1ddd.

Gas Saving Findings

None.

Informational Findings

1. Informational - Add a `LockModified()` event

Technical Details

The contract emits events when a lock is created or removed, but it doesn't emit events when a lock is modified. For example, in the _update() function, if tokens were already transferred to the address less than a day ago the lock will be updated, but no events will be emitted.

Impact

Informational.

Recommendation

Add a `LockModified()` event.

Developer Response

Acknowledged. We do not think that it is justified to add this event. `LockCreated` event does not emit the amount information on purpose. It only emits the `account` and `lockTimestamp` because the lock creation/removal for the account seems to be the only useful information from the off-chain monitoring perspective. The fact that the lock was modified does not convey much additional information. In fact, it seems to be redundant as in can always be paired with a traditional ERC20 `Transfer` event.

2. Informational - `EVCAccountOwner` can reschedule unlocks for locked users.

The `setWhitelistStatus` function in the `ERC20WrapperLocked` contract allows the EVC Account Owner to modify the lock schedule for users. This capability can potentially be used to reschedule unlocks for locked users, which may not align with the expected behavior of the locking mechanism.

Technical Details

The `setWhitelistStatus` function is callable by the EVC Account Owner (via the `onlyEVCAccountOwner` modifier) and the contract owner (via the `onlyOwner` modifier). When changing a user's whitelist status, the function performs the following actions:

1. When whitelisting an account (setting status to `true`):
   - Removes all existing locks for the account
   - Effectively unlocks all tokens for the account
2. When removing an account from the whitelist (setting status to `false`):
   - Creates a new lock for the account's entire balance
   - Sets the lock timestamp to the current normalized timestamp

The owner can reset the unlock schedule for an account by adding and then removing it from the whitelist.

Impact

Informational.

Recommendation

There is no clear recommendation that would address this issue without potentially compromising the contract's intended functionality. The ability to modify whitelist status and its effects on locks appears to be a core feature of the contract design. It is important to ensure that users are fully aware of this functionality and its implications.

Developer Response

We acknowledge this behavior, it is by design. Additional natspec comments were added to highlight it in https://github.com/euler-xyz/evk-periphery/pull/110/commits/63074f118b11bc3cf9cd8d1a821bdb882950e4cc.

3. Informational - `onlyEVCAccountOwner()` only used once

Technical Details

The modifier `onlyEVCAccountOwner()` checks that when the protocol is called through the EVC, the original caller is the account owner, and no checks are in progress. It is only used in the function `setWhitelistStatus`.

However, the `setRemainderReceiver()` doesn't implement this modifier. This could allow an operator set for the contract owner to change the `remainderReceiver` address.

Impact

Informational.

Recommendation

Add the modifier to the other functions or remove it.

Developer Response

Fixed in https://github.com/euler-xyz/evk-periphery/commit/ae28cab7ff73e0d90a0efd017bbc104f7a9ec327.

Final remarks

Auditors found the Euler ERC20 Wrapper Locked codebase to be simple and well-structured. While it is primarily intended for internal use by the Euler team with the EUL token, it appears flexible enough for adoption by other teams and projects within the ecosystem. However, they must ensure their tokens do not exhibit unexpected behaviors, such as rebasing or callbacks.