e

CD SECURITY

AUDIT REPORT

Euler

October 2024

Prepared by
GT_GSEC
tsvetanovv
MrPotatoMagic

Introduction

A time-boxed security review of the **Euler** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Euler**

The `ERC20WrapperLocked.sol` contract is a wrapper designed specifically for the `EUL` token, aiming to distribute and lock `rEUL` tokens as rewards with a specific unlock schedule.

Tokens are locked and gradually released based on this configurable schedule, providing controlled access to `rEUL` rewards over time.

The contract enforces transfer restrictions, ensuring that locked tokens cannot be moved freely, while only certain privileged accounts can bypass these locks.

Severity classification

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

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - f81de18be410e77176e2da247d8402576e99c40b**

Scope

The following smart contracts were in scope of the audit:

- `ERC20WrapperLocked.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 0 issues
- Low: 4 issues
- Informational: 4 issues

Findings Summary

ID	Title	Severity	Status
[L-01]	Approved distributor can reset user's lock by means of `transferFrom`	Low	Fixed
[L-02]	Distributor can degrade whitelist status to break invariant and unwrap tokens	Low	Acknowledged
[L-03]	Lock is created for address(0) when whitelisted address burns tokens	Low	Fixed
[L-04]	`Eul` tokens lockout may impact voting capabilities within Euler protocol	Low	Acknowledged
[I-01]	Whitelisting is not applied for allowance and spender	Informational	Fixed
[I-02]	Inaccurate NatSpec comments related to the distributor status	Informational	Fixed
[I-03]	Undocumented behaviour of admin allowed to degrade to distributor	Informational	Acknowledged
[I-04]	Risk of temporary `EUL` token lockup in contract due to centralization	Informational	Fixed

Detailed Findings

[L-01] Approved distributor can reset user's lock by means of `transferFrom`

Description

The NatSpec comments for the `_setWhitelistStatus` function state that the owner of protocol can modify the whitelist status and its effects on user's locks as it is essential protocol's functionality:

```
/// @dev If the account is being whitelisted, all the locked amounts
are removed resulting in all the tokens being /// unlocked. If the
account being removed from the whitelist, the current account balance
is locked. The side /// effect of this behavior is that the owner can
modify the lock schedule for users, i.e. by adding and then ///
removing an account from the whitelist, the owner can reset the
unlock schedule for that account. It must be /// noted though that
the ability to modify whitelist status and its effects on locks is a
core feature of this /// contract.
```

However, the lock can be reset also whenever approved, whitelisted user with `WHITELIST_STATUS_DISTRIBUTOR` status transfers tokens by means of `transferFrom()` function. As a result, not-whitelisted user can have lock period reset by other means that stated within the documentation.

PoC

1. As protocol owner, deposit some amount of tokens to user1.
2. Forward blockchain time for 90 days.
3. As a user1, approve an amount of tokens for user2 who has `WHITELIST_STATUS_DISTRIBUTOR` status assigned.
4. As a user2, by means of the `transferFrom()` transfer tokens to self.
5. As a user2, transfer tokens back to the user1.
6. Observe the user1's locks status. Note that initial lock is reset to 180 days.

Recommendations

It is recommended to review the implementation and decide whenever the aforementioned scenario is valid by design, or additional conditions must be implemented to prevent not intended lock period reset.

[L-02] Distributor can degrade whitelist status to break invariant and unwrap tokens

Description

According to the invariant here, the distributor is not allowed to unwrap tokens it holds and can only transfer them to whitelisted or non-whitelisted addresses. A potential oversight that could become a privilege escalation issue is that once the distributor receives its allocation from the admin, the distributor can degrade its whitelist status, which allows the distributor to unwrap the reward tokens.

How to use this POC:

- Add it at the bottom of the RewardToken.t.sol test contract in the `test` folder
- To view the console.log statements, add console to the namespace as shown in the POC
- Run the test using `forge test --mt testDistributorIssue -vvvvv`
- The traces have been added below the POC

```
import {Vm, Test, console} from "forge-std/Test.sol";

address distributor = makeAddr("distributor");

    function testDistributorIssue() public {
        vm.prank(owner);
        rewardToken.setWhitelistStatus(distributor, 2);

        vm.prank(owner);
        rewardToken.setWhitelistStatus(owner, 1);

        vm.prank(owner);
        erc20Mintable.mint(owner, 10e18);

        vm.prank(owner);
        erc20Mintable.approve(address(rewardToken), 10e18);

        vm.prank(owner);
        rewardToken.depositFor(distributor, 10e18);
        console.log("Reward Token Balance of distributor:",
rewardToken.balanceOf(distributor));

        // Distributor degrades its whitelist status
        console.log("Distributor degrades its whitelist status");
        vm.prank(distributor);
        rewardToken.setWhitelistStatus(0);

        console.log("Distributor attempts to unwrap");
        skip(180 days);
        vm.prank(distributor);
        rewardToken.withdrawToByLockTimestamp(distributor, 0, false);

        console.log("Underlying token Balance of distributor after
unwrap:", erc20Mintable.balanceOf(distributor));
        console.log("Reward Token Balance of distributor after unwrap:",
rewardToken.balanceOf(distributor));
    }
```

Traces/Logs

```
[PASS] testDistributorIssue() (gas: 275536)
Logs:
  Reward Token Balance of distributor: 10000000000000000000
  Distributor degrades its whitelist status
  Distributor attempts to unwrap
  Underlying token Balance of distributor after unwrap:
10000000000000000000
  Reward Token Balance of distributor after unwrap: 0
```

Recommendations

Disallow distributor from degrading its whitelist status. Even if degrading whitelist status needs to be kept for distributor, consider auto-transferring a distributor's balance to the remainderReceiver or owner of the contract.

[L-03] Lock is created for address(0) when whitelisted address burns tokens

Description

In `update()` here, the team expects no special handling to occur when a whitelisted address burns tokens.

When a whitelisted address burns tokens though, we enter the if block here since the from address is a whitelisted address (the address burning the tokens) and the to address is the zero address, which is non-whitelisted. The `EnumerableMap` is updated for address(0) as if tokens were minted even though they were actually burnt. This behaviour is odd since the comments linked here mention there should not be any special handling present. Nonetheless, this does not pose a risk but the team should be aware of this behaviour as it could unnecessarily emit the `LockCreated` event for address(0) here every time a whitelisted address burns tokens.

How to use this POC:

- Add it at the bottom of the RewardToken.t.sol test contract in the test folder
- To view the console.log statements, add console to the namespace as shown in the POC
- Run the test using forge test --mt testWhitelistedBurnIssue -vvvvv

Traces/Logs:

```
[PASS] testWhitelistedBurnIssue() (gas: 354570)
Logs:
  Address(0) amount: 10000000000000000000
```

```
    function testWhitelistedBurnIssue() public {
        vm.prank(owner);
        rewardToken.setWhitelistStatus(owner, 1);

        vm.prank(owner);
        erc20Mintable.mint(owner, 20e18);

        vm.prank(owner);
        erc20Mintable.approve(address(rewardToken), 20e18);

        vm.prank(owner);
        rewardToken.depositFor(distributor, 10e18);

        vm.prank(owner);
```

```
        rewardToken.depositFor(owner, 10e18);

        // Whitelisted owner burns tokens
        vm.prank(owner);
        rewardToken.withdrawTo(distributor, 10e18);

        uint256 amount =
rewardToken.getLockedAmountByLockTimestamp(address(0), 0);
        console.log("Address(0) amount:", amount);
    }
```

[L-04] `Eul` Tokens lockout may impact voting capabilities within Euler protocol

Description

The `ERC20WrapperLocked` contract is meant to be used with the `Eul` token. This token implements OpenZeppelin's ERC20Votes library. Effectively, the EUL tokens represent governance voting powers to effect change over the Euler Protocol code. However, there is no limit set regarding the amount of tokens that can be locked within the wrapper. Thus, depending on the wrapper usage, a significant amount of tokens can be locked within the wrapper. These tokens cannot be used for voting unless released. Ultimately, uncontrolled tokens lockout may negatively impact the protocol's governance mechanism.

Recommendations

It is recommended to consider implementing upper bound limit to prevent wrapping and locking significant amounts of `EUL` tokens, that could impact the governance mechanism.

[I-01] Whitelisting is not applied for allowance and spender

Description

Within the `ERC20WrapperLocked` contract a whitelisting access control is applied to control the transfer of wrapped tokens functionality. However, this security control is not applied for the `approve` function. Thus, any user can approve any amount of tokens to any whitelisted and non-whitelisted account. Furthermore, within the `transferFrom` function only the `from` and `to` addresses are being checked against the whitelist, but not the `spender`. Thus, whenever approved, a not whitelisted user can transfer tokens whenever one from the `from` and `to` addresses is whitelisted. The finding is reported as a deviation from leading security practices.

Recommendations

It is recommended to review the implementation and decide whenever the aforementioned scenario is valid by design, or additional security controls should be applied for `approve()` and `transferFrom` functions.

[I-02] Inaccurate NatSpec comments related to the distributor status

Description

The NatSpec comments for the `ERC20WrapperLocked` contract states that an account with the `WHITELIST_STATUS_DISTRIBUTOR` whitelist status can transfer tokens only to not-whitelisted account or to the account with the `WHITELIST_STATUS_ADMIN` status.

> ... If the account has a `DISTRIBUTOR` whitelist status, their tokens /// cannot be unwrapped by them, but can only be transferred to the account that is not whitelisted and become a subject /// to the locking schedule or transferred to the account with an ADMIN whitelist status.

However, in fact, such user can transfer tokens to any whitelisted account with `WHITELIST_STATUS_DISTRIBUTOR` whitelist status, including self. The finding is reported as a deviation from leading security practices.

Recommendations

It is recommended to update the NatSpec comments with accurate information.

[I-03] Undocumented behaviour of admin allowed to degrade to distributor

Description

Admins can call `setWhitelistStatus()` to degrade their role to `NONE` or `DISTRIBUTOR`. The NatSpec here only mentions that the whitelisted accounts can degrade their whitelist status but it does not document whether this includes an admin being able to degrade to a distributor. Depending on the intention, consider implementing the necessary fixes or documenting the behaviour.

```
    function setWhitelistStatus(uint256 status) public
onlyWhitelisted(_msgSender()) {
        address account = _msgSender();
        if (whitelistStatus[account] != status) {
            if (status == WHITELIST_STATUS_ADMIN) {
                revert NotAuthorized();
            } else {
                _setWhitelistStatus(account, status);
            }
        }
    }
```

[I-04] Risk of temporary EUL token lockup in contract due to centralization

Description

There is a potential centralization risk which could lock up EUL tokens in the contract. If `remainderReceiver` is address(0) and owner renounces ownership, the `EUL` ERC20 transfer will not be able to occur (due to ERC20 transfer disallowing transfers to the zero address). Users can wait till 180 days though to have no remainder amount so tokens will then be releasable. Case to be aware of though does not pose risk other than the `EUL` market price affecting users while their `EUL` tokens are locked in for 6 months.

If the owner role is never expected to be renounced, consider overriding the `renounceOwnership()` and revert in its body.

```
if (totalRemainderAmount != 0) {
            if (!allowRemainderLoss) revert RemainderLossNotAllowed();

            address receiver = remainderReceiver;
            asset.safeTransfer(receiver == address(0) ? owner() :
receiver, totalRemainderAmount);
        }
```