SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Eco Protocol
Prepared by:	Sherlock
Lead Security Expert:	stopthecap
Dates Audited:	May 22 - May 25, 2023
Prepared on:	June 12, 2023

Introduction

The Eco Association is a non-profit organization dedicated to supporting the growth of the Eco Currency protocol.

Scope

Repository: eco-association/op-eco

Branch: main

Commit: 39f205fb46eea3df770f0119a890ffdc1ac8f382

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
0	2

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0xDjango	toshii	J4de
ecexit	GimelSec	Dug
0xdeadbeef	n33k	
0xRobocop	nobody2018	

Issue H-1: Stale inflationMultiplier in L1ECOBridge

Source:
https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/126

Found by

0xRobocop, Dug, GimelSec, J4de, n33k, nobody2018, toshii

Summary

`L1ECOBridge::inflationMultiplier` is updated through `L1ECOBridge::rebase` on Ethereum, and it is used in `_initiateERC20Deposit` and `finalizeERC20Withdrawal` to convert between token amount and `_gonsAmount`. However, if `rebase` is not called in a timely manner, the `inflationMultiplier` value can be stale and inconsistent with the value of L1 ECO token during transfer, leading to incorrect token amounts in deposit and withdraw.

Vulnerability Detail

The `inflationMultiplier` value is updated in `rebase` with an independent transaction on L1 as shown below:

```
function rebase(uint32 _l2Gas) external {
    inflationMultiplier = IECO(l1Eco).getPastLinearInflation(block.number);
```

However, in both `_initiateERC20Deposit`, `transferFrom` is called before the `inflationMultiplier` is used, which can lead to inconsistent results if `rebase` is not called on time for the `inflationMultiplier` to be updated. The code snippet for `_initiateERC20Deposit` is as follows:

```
IECO(_l1Token).transferFrom(_from, address(this), _amount);
_amount = _amount * inflationMultiplier;
```

`finalizeERC20Withdrawal` has the same problem.

```
uint256 _amount = _gonsAmount / inflationMultiplier;
bytes memory _ecoTransferMessage =
↪   abi.encodeWithSelector(IERC20.transfer.selector,_to,_amount);
```

The same problem does not exist in L2ECOBridge. Because the L2 rebase function updates inflationMultiplier and rebase l2Eco token synchronously.

```
function rebase(uint256 _inflationMultiplier)
    external
    virtual
    onlyFromCrossDomainAccount(l1TokenBridge)
    validRebaseMultiplier(_inflationMultiplier)
{
    inflationMultiplier = _inflationMultiplier;
    l2Eco.rebase(_inflationMultiplier);
    emit RebaseInitiated(_inflationMultiplier);
}
```

Impact

The attacker can steal tokens with this.

He can deposit to L1 bridge when he observes a stale larger value and he will receive more tokens on L2.

Code Snippet

https://github.com/sherlock-audit/2023-05-ecoprotocol/blob/main/op-eco/contracts/bridge/L1ECOBridge.sol#L244-L251

https://github.com/sherlock-audit/2023-05-ecoprotocol/blob/main/op-eco/contracts/bridge/L1ECOBridge.sol#L333-L335

Tool used

Manual Review

Recommendation

Calling `IECO(l1Eco).getPastLinearInflation(block.number)` instead of using `inflationMultiplier`.

Discussion

syjcnss

Escalate for 10 USDC.

This is different from #52 which is about desynced inflationMultiplier between L1&L2 bridges.

#17, #83, #110 and #126 are about desynced inflationMultiplier between the L1ECOBridge and the L1 ECO token.

In L1, the actually transfered gos amount will be different from the gos amount deposited if the inflationMultiplier is stale and not updated by calling rebase. This has nothing to do with the L2 inflationMultiplier.

```
bridge/L1ECOBridge.sol:_initiateERC20Deposit

IECO(_l1Token).transferFrom(_from, address(this), _amount);    // <- transfer
↪  underlying gos amount(_amount * 'inflationMultiplier' inside the L1 ECO
↪  token) of tokens to deposit
_amount = _amount * inflationMultiplier;  // <- record a wrong gos deposited
↪  token amount if inflationMultiplier is stale
```

In the PoC, @albertnbrown proves that a desynced inflationMultiplier between L1&L2 bridges does not cause a problem. The PoC has a synced value between L1ECOBridge and the L1 ECO token at desposit because the constructor initiated inflationMultiplier with IECO(l1Eco).getPastLinearInflation(block.number). So it doesn't invalidate this issue.

sherlock-admin

> Escalate for 10 USDC.
>
> This is different from #52 which is about desynced inflationMultiplier between L1&L2 bridges.
>
> #17, #83, #110 and #126 are about desynced inflationMultiplier between the L1ECOBridge and the L1 ECO token.
>
> In L1, the actually transfered gos amount will be different from the gos amount deposited if the inflationMultiplier is stale and not updated by calling rebase. This has nothing to do with the L2 inflationMultiplier.
>
> ```
> bridge/L1ECOBridge.sol:_initiateERC20Deposit
>
> IECO(_l1Token).transferFrom(_from, address(this), _amount);    // <-
> ↪  transfer underlying gos amount(_amount * 'inflationMultiplier' inside
> ↪  the L1 ECO token) of tokens to deposit
> _amount = _amount * inflationMultiplier;  // <- record a wrong gos
> ↪  deposited token amount if inflationMultiplier is stale
> ```
>
> In the PoC, @albertnbrown proves that a desynced inflationMultiplier between L1&L2 bridges does not cause a problem. The PoC has a synced value between L1ECOBridge and the L1 ECO token at desposit because the constructor initiated inflationMultiplier with IECO(l1Eco).getPastLinearInflation(block.number). So it doesn't invalidate this issue.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

albertnbrown

This is a legitimate high issue and we found it and addressed it on our own already, but were having trouble looking through the duplicates for #52 for issues like this to correctly reward them. Thanks for escalating it for visibility.

albertnbrown

We fixed this here https://github.com/eco-association/op-eco/pull/40

ctf-sec

I agree with

> This is different from https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/52 which is about desynced inflationMultiplier between L1&L2 bridges.

https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/17, https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/83, https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/110 and https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/126 are about desynced inflationMultiplier between the L1ECOBridge and the L1 ECO token.

while this is the context read me

#17, #83, #110 and #126 and #13

are all duplicate of #126, can mark as high severity

hrishibhat

Escalation accepted

Valid high

sherlock-admin

> Escalation accepted
>
> Valid high

```
This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on
↪ this issue.
```

0xffff11

Approved fix in: https://github.com/eco-association/op-eco/pull/40

0xffff11

Fixed confirmed. Using the token inflationMultiplier instead of the bridge inflationMultiplier

Issue H-2: Malicious actor cause rebase to an old inflation multiplier

Source:
https://github.com/sherlock-audit/2023-05-ecoprotocol-judging/issues/142

Found by

0xDjango, 0xdeadbeef, ecexit

Summary

The protocol has a rebasing mechanism that allows to sync the inflation multiplier between both L1 and L2 chains. The call to rebase is permissionless (anyone can trigger it). Insufficant checks allow a malicious actor to rebase to an old value.

Vulnerability Detail

Rebasing from L1 to L2 is through the `L1ECOBridge` rebase function. It collects the inflation multiplier from the ECO token and sends a message to `L2ECOBridge` to update the L2 ECO token inflation multiplier. https://github.com/sherlock-audit/2023-05-ecoprotocol/blob/main/op-eco/contracts/bridge/L1ECOBridge.sol#L296

```
function rebase(uint32 _l2Gas) external {
    inflationMultiplier = IECO(l1Eco).getPastLinearInflation(
        block.number
    );

    bytes memory message = abi.encodeWithSelector(
        IL2ECOBridge.rebase.selector,
        inflationMultiplier
    );

    sendCrossDomainMessage(l2TokenBridge, _l2Gas, message);
}
```

A malicious actor can call this function a large amount of times to queue messages on `L2CrossDomainMessenger`. Since it is expensive to execute so much messages from `L2CrossDomainMessenger` (especially if the malicious actor sets `_l2Gas` to a high value) there will be a rebase message that will not be relayed through `L2CrossDomainMessenger` (or in failedMessages array).

Some time passes and other legitimate rebase transactions get executed.

One day the malicious actor can execute one of his old rebase messages and set the value to the old value. The attacker will debalance the scales between L1 and

L2 and can profit from it.

Impact

debalance the scales between L1 and L2 ECO token

Code Snippet

Tool used

Manual Review

Recommendation

When sending a rebase from L1, include in the message the L1 block number. In L2 rebase, validate that the new rebase block number is above previous block number

Discussion

albertnbrown

This is legitimate because unlike upgrade functions, the `rebase` function has no auth guards. We have added fixes to this to this PR:

https://github.com/eco-association/op-eco/pull/33

0xdeadbeef0x

Escalate for 10 USDC

Escalating to verify that this gets the the reward tag as it was confirmed and fixed by the sponsor.

sherlock-admin

> Escalate for 10 USDC
>
> Escalating to verify that this gets the the reward tag as it was confirmed and fixed by the sponsor.

You've created a valid escalation for 10 USDC!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation accepted

Valid high

sherlock-admin

> Escalation accepted
>
> Valid high

```
This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on
↪  this issue.
```

0xffff11

Added fix in: https://github.com/eco-association/op-eco/pull/33

0xffff11

Fix confirmed, added block number to L1 calls to prevent any replay attacks using failed cross-bridge calls