MiloTruck

Celo

Security Review

February 28, 2025

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher, primarily working as a Lead Security Researcher at Spearbit and Cantina. Previously, he was part of the team at Renascence Labs and a Lead Auditor at Trust Security.

For private audits or security consulting, please reach out to him on Twitter *@milotruck*.

1.2 Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

2 Risk Classification

Severity Level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality.
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality/availability.
- Low - Funds are **not** at risk.

2.2 Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

3 Executive Summary

3.1 About Celo

Celo is an emerging Ethereum Layer-2 designed to make blockchain technology accessible to all. With its focus on scalability, low fees, and ease of use, Celo is ideal for building blockchain products that reach millions of users around the globe.

To learn more about Celo, please visit https://celo.org/.

3.2 Overview

Project Name	Celo
Project Type	L2
Language	Solidity
Repository	optimism
Commit Hash	9526db529ded2781d43a22d66e6ff58efe6fcb59

3.3 Issues Found

High	1
Medium	1
Low	3
Informational	3

4 Findings

4.1 High Risk

4.1.1 CELO is permanently stuck on L2 and cannot be withdrawn to L1

Context:

- CeloTokenL1Permit.sol#L11-L15
- OptimismPortal2.sol#L137-L142
- OptimismPortal2.sol#L504-L505
- OptimismPortal2.sol#L438-L439

Description: `CeloTokenL1Permit`, which is the custom gas token (CGT) on L1, directly mints 1 billion CELO to `OptimismPortal2` on deployment:

```
contract CeloTokenL1Permit is ERC20, ERC20Permit {
    constructor(address portalProxyAddress) ERC20(NAME, SYMBOL) ERC20Permit(NAME) {
        _mint(portalProxyAddress, TOTAL_MARKET_CAP);
    }
}
```

On L2 genesis, 1 billion CELO will be distributed to addresses to mirror the current state on Celo L1.

However, tokens directly minted to `OptimismPortal2` will be permanently stuck as the portal tracks the amount of CGT minted on L2 internally with `_balance`:

```
/// @notice Represents the amount of native asset minted in L2. This may not
///         be 100% accurate due to the ability to send ether to the contract
///         without triggering a deposit transaction. It also is used to prevent
///         overflows for L2 account balances when custom gas tokens are used.
///         It is not safe to trust `ERC20.balanceOf` as it may lie.
uint256 internal _balance;
```

When `depositERC20Transaction()` is called to deposit CELO from L1 -> L2, `_balance` is increased:

```
// Gives overflow protection for L2 account balances.
_balance += _mint;
```

Likewise, `_balance` is decreased when CELO is withdrawn from L2 -> L1 in `finalizeWithdrawalTransactionExternalProof()`:

```
// Update the contracts internal accounting of the amount of native asset in L2.
_balance -= _tx.value;
```

Since `CeloTokenL1Permit` directly mints tokens to `OptimismPortal2`, `_balance` will remain at `0` after deployment. When any user attempts to bridge CELO from L2 back to L1, `finalizeWithdrawalTransactionExternalProof()` will revert due to an underflow when the bridged amount is subtracted `_balance`.

As a result, all CELO is stuck on L2 and cannot be bridged back to L1.

Recommendation: In `OptimismPortal2.initialize()`, initialize `_balance` as 1 billion tokens on deployment:

```
  // Set the `l2Sender` slot, only if it is currently empty. This signals the first initialization of
↪  the
  // contract.
  if (l2Sender == address(0)) {
      l2Sender = Constants.DEFAULT_L2_SENDER;

      // Set the `respectedGameTypeUpdatedAt` timestamp, to ignore all games of the respected type prior
      // to this operation.
      respectedGameTypeUpdatedAt = uint64(block.timestamp);

      // Set the initial respected game type
      respectedGameType = _initialRespectedGameType;

+     _balance = 1_000_000_000e18;
  }
```

Additionally, consider adding a test which ensures CELO withdrawals from L2 to L1 is possible post-genesis.

Celo: Acknowledged. This is being addressed in Deploy.s.sol#L462 as part of our deployment script since we want to minimize differences between Optimism's repository and our fork.

MiloTruck: Acknowledged.

4.2 Medium Risk

4.2.1 Inconsistent usage of `checkAndPauseIfSuperchainPaused()`/`paused()` in L1 contracts

Context:

- L1CrossDomainMessenger.sol#L98-L100, L1StandardBridge.sol#L116-L118
- L1ERC721Bridge.sol#L76
- OptimismPortal2.sol#L180-L183

Description: The functions in `CeloSuperchainConfig` which are called to determine if Celo is paused are different across L1 contracts.

`L1CrossDomainMessenger` and `L1StandardBridge` call `paused()`:

```
function paused() public view override returns (bool) {
    return superchainConfig.paused();
}
```

In contrast, `L1ERC721Bridge` and `OptimismPortal` call `checkAndPauseIfSuperchainPaused()`:

```
require(superchainConfig.checkAndPauseIfSuperchainPaused() == false, "L1ERC721Bridge: paused");
```

```
modifier whenNotPaused() {
    if (superchainConfig.checkAndPauseIfSuperchainPaused()) revert CallPaused();
    _;
}
```

Therefore, if a user calls into `L1ERC721Bridge` or `OptimismPortal` while the superchain is paused, `PAUSED_SLOT` will be set and Celo will remain paused after the superchain is unpaused.

In contrast, calls to `L1CrossDomainMessenger` or `L1StandardBridge` do not set `PAUSED_SLOT`, causing Celo to automatically unpause with the superchain.

Recommendation: Consider modifying `L1ERC721Bridge` and `OptimismPortal` to use `pause()` as well.

Celo: Fixed in PR 332. We realized it is not possible for those functions to change Celo's paused state, since in every case, the function would revert if it's paused.

MiloTruck: Verified, `L1ERC721Bridge` and `OptimsimPortal` call `pause()` instead of `checkAndPauseIfSuperchainPaused()` now. Since Celo's paused state no longer changes when L1 contracts are called, there should be some monitoring in place to regularly check if the Superchain is paused and call `checkAndPauseIfSuperchainPaused()` if so.

4.3 Low Risk

4.3.1 `ISuperchainConfig` is not updated for `DelayedWETH`

Context:

- DelayedWETH.sol#L44-L45
- DelayedWETH.sol#L91-L92

Description: `DelayedWETH` calls the `superchainConfig` contract to determine if `withdraw()` should be paused:

```
/// @notice Address of the SuperchainConfig contract.
ISuperchainConfig public config;
```

```
function withdraw(address _guy, uint256 _wad) public {
    require(!config.paused(), "DelayedWETH: contract is paused");
```

However, unlike the other L1 contracts, it has not been modified to use the `ICeloSuperchainConfig` interface or call `checkAndPauseIfSuperchainPaused()`.

Recommendation: Modify `DelayedWETH` to use `ICeloSuperchainConfig`.

Celo: Fixed in PR 329.

MiloTruck: Verified, the recommendation has been implemented.

4.3.2 Celo is not automatically unpaused when superchain unpauses

Context: CeloSuperchainConfig.sol#L55-L62

Description: In `CeloSuperchainConfig`, `checkAndPauseIfSuperchainPaused()` can be called to set `PAUSED_SLOT` to `true` whenever the superchain is paused:

```
function checkAndPauseIfSuperchainPaused() public returns (bool paused_) {
    if (ISuperchainConfig(superchainConfig()).paused()) {
        _pause("Superchain paused");
        return true;
    }

    return paused();
}
```

This allows `PAUSED_SLOT` to be automatically set whenever the superchain is paused.

However, when the superchain is unpaused by Optimism's guardian, `PAUSED_SLOT` will still be set to `true`. Therefore, Celo L2 will still be paused and Celo's guardian will be forced to manually unpause the chain.

This design is not optimal as Celo should automatically unpause if it was not paused by Celo's guardian.

Recommendation: Consider removing `checkAndPauseIfSuperchainPaused()`. There is no need to set `PAUSED_SLOT` to `true` when the superchain is paused, since `paused()` will always return `true` when the superchain is paused.

Celo: Acknowledged, this is the intended behavior. Since Celo is not a part of the Superchain, we may have a different attack surface than other Superchain chains and we might not be 100% coordinated with them. So a fix for Superchain chains being deployed does not guarantee that it fixes Celo as well.

MiloTruck: Acknowledged.

4.3.3 Recommendations for `CeloSuperchainConfig`

Context:

1. CeloSuperchainConfig.sol#L24-L25
2. SuperchainConfig.sol#L50-L55, CeloSuperchainConfig.sol#L42-L50
3. CeloSuperchainConfig.sol#L55-L62

Description/Recommendation:

(1) `CeloSuperchainConfig` uses the `superchainConfig` namespace for `SUPERCHAIN_CONFIG_SLOT`:

```
bytes32 public constant SUPERCHAIN_CONFIG_SLOT =
    bytes32(uint256(keccak256("superchainConfig.superchainConfig")) - 1);
```

However, Optimism's `SuperchainConfig` uses the same namespace for its variables. Consider using a different namespace, such as `celoSuperchainConfig.superchainConfig`, to avoid storage collisions if `CeloSuperchainConfig` is upgraded with a different `SuperchainConfig` implementation in the future.

(2) `CeloSuperchainConfig` inherits `SuperchainConfig`, which contains the following `initialize()` function:

```
function initialize(address _guardian, bool _paused) public initializer {
    _setGuardian(_guardian);
    if (_paused) {
        _pause("Initializer paused");
    }
}
```

This function is not overridden as the `initialize()` function declared in `CeloSuperchainConfig` has different parameters. Therefore, it is possible to call the wrong `initialize()` function during deployment. Consider modifying `SuperchainConfig` to remove this `initialize()` function.

(3) `checkAndPauseIfSuperchainPaused()` calls `superchainConfig.paused()` without checking if `superchainConfig()` is the zero address:

```
function checkAndPauseIfSuperchainPaused() public returns (bool paused_) {
    if (ISuperchainConfig(superchainConfig()).paused()) {
        _pause("Superchain paused");
        return true;
    }

    return paused();
}
```

Therefore, if `superchainConfig()` was initialized as `address(0)`, `checkAndPauseIfSuperchainPaused()` will always revert when called. Consider modifying the check as such:

```
- if (ISuperchainConfig(superchainConfig()).paused()) {
+ if (superchainConfig() != address(0) && ISuperchainConfig(superchainConfig()).paused()) {
        _pause("Superchain paused");
      return true;
  }
```

Celo: Fixed in PR 323 and PR 326.

MiloTruck: Verified. Issues 1 and 3 were fixed as recommended, while issue 2 was fixed by implementing a `CeloSuperchainConfig` that does not inherit `SuperchainConfig`.

4.4 Informational

4.4.1 `OptimismMintableERC20` tokens deployed on L1 also inherit `AbstractFeeCurrency`

Context: OptimismMintableERC20.sol#L13-L19

Description: `OptimismMintableERC20` was modified to inherit `AbstractFeeCurrency`:

```
/// @title OptimismMintableERC20
/// @notice OptimismMintableERC20 is a standard extension of the base ERC20 token contract designed
///         to allow the StandardBridge contracts to mint and burn tokens. This makes it possible to
///         use an OptimismMintablERC20 as the L2 representation of an L1 token, or vice-versa.
///         Designed to be backwards compatible with the older StandardL2ERC20 token which was only
///         meant for use on L2.
contract OptimismMintableERC20 is ERC20Permit, ISemver, AbstractFeeCurrency {
```

This allows Celo's VM on L2 to mint/burn `OptimismMintableERC20` tokens through the zero address, described in detail here.

However, `OptimismMintableERC20Factory` (the factory which deploys `OptimismMintableERC20` tokens) is deployed on both L1 and L2. Therefore, `OptimismMintableERC20` tokens created on L1 will also inherit `AbstractFeeCurrency`, although its functions cannot be called on Ethereum.

Note that there is no impact; the `debitGasFees()` and `creditGasFees()` functions will simply be un-callable on L1.

Celo: Acknowledged.

MiloTruck: Acknowledged.

4.4.2 Missing call to `_disableInitializers()` in `CeloTokenL1`

Context: CeloTokenL1.sol

Description: `CeloTokenL1` is an upgradeable contract with an `initialize()` function:

```
contract CeloTokenL1 is ERC20Upgradeable {
    function initialize(address portalProxyAddress) public initializer {
        __ERC20_init(NAME, SYMBOL);
        _mint(portalProxyAddress, TOTAL_MARKET_CAP);
    }
}
```

However, there is no constructor with a call to `_disableInitializers()`. This allows the implementation contract of `CeloTokenL1` to be initialized by an attacker.

Recommendation: Consider adding the following constructor:

```
constructor() {
    _disableInitializers();
}
```

Celo: Fixed in PR 325.

MiloTruck: Verified, the recommendation has been implemented.

4.4.3 Minor improvements to code

Context: See below.

Description/Recommendation:

1. CeloSuperchainConfig.sol#L61 - `checkAndPauseIfSuperchainPaused()` can return `super.paused()` instead:

```
- return paused();
+ return super.paused();
```

2. CeloSuperchainConfig.sol#L71-L82 - `paused()` can be simplified as such:

```
function paused() public view override returns (bool paused_) {
    return super.paused() || (
        superchainConfig() != address(0) && ISuperchainConfig(superchainConfig()).paused()
    );
}
```

3. CeloTokenL1.sol#L6-L8, CeloTokenL1Permit.sol#L7-L9 - These constants should be moved into their respective contracts.
4. CeloTokenL1.sol#L11 - `initialize()` can be declared `external` instead.
5. CeloTokenL1Permit.sol#L11 - `CeloTokenL1Permit` does not need to inherit `ERC20` as `ERC20Permit` already inherits it:

```
- contract CeloTokenL1Permit is ERC20, ERC20Permit {
+ contract CeloTokenL1Permit is ERC20Permit {
```

6. AbstractFeeCurrency.sol#L17, AbstractFeeCurrency.sol#L26-L38 - Both instances of `creditGasFees()` can be declared external.

Celo: Partially fixed in PR 325 and PR 336.

MiloTruck: Verified.