OpenZeppelin | security

gora Partial Delegation udit

June 24, 2024

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

ERC20VotesPartialDelegationUpgradeable 5

VotesPartialDelegationUpgradeable 5

IERC5805Modified and IVotesPartialDelegation 6

L2GovToken 6

Upgradeability and Access Control 6

EIP-712 and Permit 6

Security Model and Trust Assumptions 7

Risks and Mitigations 7

Trust Assumptions and Privileged Roles 7

Deployment Assumptions 8

Low Severity 9

L-01 Floating Pragma 9

L-02 Potential Non-Zero Voting Weight for Zero Address 9

L-03 Multiple Emissions of DelegateChanged Event 10

L-04 Misleading Documentation 10

Notes & Additional Information 11

N-01 Constant Not Using UPPER_CASE Format 11

N-02 Lack of Security Contact 11

N-03 Add Functionality to Cancel Signed Messages 12

N-04 Misleading Interface Name 12

N-05 Custom Errors Inconsistent with Solidity Style Guide 13

N-06 Gas Optimizations 13

Conclusion 15

Summary

Type	Governance
Timeline	From 2024-06-10 To 2024-06-19
Languages	Solidity

Total Issues	10 (10 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (4 resolved)
Notes & Additional Information	6 (6 resolved)

Scope

We audited the voteagora/ERC20VotesPartialDelegationUpgradeable repository at commit 92ffae9.

In scope were the following files:

```
src
├── ERC20VotesPartialDelegationUpgradeable.sol
├── IERC5805Modified.sol
├── IVotesPartialDelegation.sol
├── L2GovToken.sol
└── VotesPartialDelegationUpgradeable.sol
```

System Overview

Agora's ERC-20 voting contract is an extension of the standard ERC-20 token, incorporating advanced voting functionality with partial delegation capabilities. Built upon the OpenZeppelin library, this contract enables users to delegate a portion of their voting power to other addresses, while maintaining a historical record of each account's voting power through checkpoints.

`ERC20VotesPartialDelegationUpgradeable`

The `ERC20VotesPartialDelegationUpgradeable` contract is an extension of the ERC-20 token standard that incorporates partial delegation and vote tracking capabilities. It inherits from OpenZeppelin's `ERC20PermitUpgradeable`, which implements the ERC-20 permit extension (EIP-2612) for gasless token approvals.

The contract uses the `VotesPartialDelegationUpgradeable` contract to manage vote delegation and checkpoint recording. It handles the transfer of voting units when tokens are transferred, minted, or burned, and enforces a maximum token supply cap to prevent overflow risks in vote accounting.

`VotesPartialDelegationUpgradeable`

The `VotesPartialDelegationUpgradeable` contract provides the core functionality for partial delegation and vote tracking. It implements the `Checkpoints` library from OpenZeppelin to efficiently store and retrieve historical voting power balances.

The contract includes functions such as `delegate`, `delegatePartiallyOnBehalf`, and `delegateBySig` to allow token holders to delegate their voting power. Furthermore, it contains the internal functions used by child contracts to move voting power upon mint, transfer, and burn.

`IERC5805Modified` and `IVotesPartialDelegation`

The `IERC5805Modified` and `IVotesPartialDelegation` interfaces define the required functions and events for the partial delegation and vote tracking functionality. `IVotesPartialDelegation` defines the specific functions and events related to partial delegation. It attempts to extend the ERC-5805 functionality but makes breaking changes to include modifications for partial delegation by removing the requirement for the `delegates` function and changing the event signature for `DelegateChanged`. `IERC5805Modified` inherits from `IERC6372` and `IVotesPartialDelegation`, and thus inherits the aforementioned breaking changes.

`L2GovToken`

The `L2GovToken` is the main token contract that inherits from `AccessControlUpgradeable` and `ERC20VotesPartialDelegationUpgradeable`. It includes additional functionality for minting and burning tokens based on the assigned roles (`MINTER_ROLE` and `BURNER_ROLE`). The contract is upgradeable and uses an initializer function to set up the token's name, symbol, and admin role.

Upgradeability and ccess Control

The `L2GovToken` contract is designed to be upgradeable using the proxy pattern. It uses the OpenZeppelin upgradeable contracts to facilitate safe upgrades. Access control is implemented using OpenZeppelin's `AccessControlUpgradeable` contract, which allows defining and managing roles for specific actions like minting and burning tokens.

EIP-712 and Permit

The contract leverages the EIP712 standard for typed structured data hashing and signing, which is used in the permit functionality. The `ERC20PermitUpgradeable` contract is used to enable gasless token approvals using the permit mechanism.

Security Model and Trust ssumptions

Risks and Mitigations

The L2GovToken contract is upgradeable which introduces the risk of potentially malicious upgrades. The contract uses OpenZeppelin's upgradeable contracts, which provide a secure upgradeability mechanism. The team has disclosed that the end goal is to replace this upgradeable token with a non-upgradeable one.

The contract relies on the `AccessControlUpgradeable` contract for role-based access control. If the admin role is compromised or mismanaged, it could lead to unauthorized actions such as minting or burning tokens. It is crucial to establish a secure and decentralized governance process for managing roles. The contract enforces a maximum token supply cap to prevent integer overflow vulnerabilities in the vote accounting. However, it is important to ensure that the cap is set appropriately and cannot be bypassed through contract upgrades.

Trust ssumptions and Privileged Roles

The `L2GovToken` contract defines the following privileged roles:

1. `DEFAULT_ADMIN_ROLE`: This role is assigned to the specified `_admin` address during contract initialization. It can grant and revoke other roles and should be controlled by a trusted governance system.
2. `MINTER_ROLE`: This role allows the assigned addresses to mint new tokens using the `mint` function. It should be granted only to trusted entities responsible for token minting according to the defined rules and governance decisions.
3. `BURNER_ROLE`: This role allows the assigned addresses to burn tokens using the `burn` function. It should be granted only to trusted entities responsible for token burning according to the defined rules and governance decisions.

The contract assumes that the addresses assigned the `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, and `BURNER_ROLE` during initialization are trustworthy and will manage the contract responsibly. Misuse of these roles could lead to token inflation or improper token

destruction. The contract relies on external libraries for various functionalities. It is important to keep an eye on any security updates or vulnerabilities reported in these dependencies.

Deployment ssumptions

This set of smart contracts is designed for usage on L2 networks and is not intended for deployment on L1 networks. The contracts are built to leverage the unique characteristics and benefits of L2 environments such as faster transaction speeds, lower gas costs, and improved scalability. When deploying these contracts on an L2 network, it is essential to consider the specific features and limitations of the chosen L2 solution. Attempting to use them on an L1 network may result in suboptimal performance and increased gas costs.

Low Severity

L-01 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, there are multiple floating pragma directives:

- `ERC20VotesPartialDelegationUpgradeable.sol` has the `solidity ^0.8.20` floating pragma directive.
- `IERC5805Modified.sol` has the `solidity ^0.8.20` floating pragma directive.
- `IVotesPartialDelegation.sol` has the `solidity ^0.8.20` floating pragma directive.
- `L2GovToken.sol` has the `solidity ^0.8.20` floating pragma directive.
- `VotesPartialDelegationUpgradeable.sol` has the `solidity ^0.8.20` floating pragma directive.

Consider using a fixed pragma directive for improved code clarity.

Update: *Resolved in pull request #58 at commit 6676441.*

L-02 Potential Non-Zero Voting Weight for Zero ddress

A user with tokens can explicitly call the `delegate` function passing in the zero address with a certain voting weight. This would create a checkpoint confirming the delegation of the voting weight to the zero address. This is also relevant for other functions in this contract that perform delegations as well such as `delegateBySig` and `delegatePartiallyOnBehalf`.

Therefore, when retrieving this value through `getVotes` or `getPastVotes`, the returned value could be non-zero. This can be unexpected for code built on top of this contract, especially as EIP-5805 states that these two functions should not count the tokens delegated to the zero address and should always return 0. Therefore, code that is reliant on the fact that this would always return 0 could be vulnerable to attacks.

Consider always returning 0 for the zero address or performing a no-op when updating the `_delegateCheckpoints` array when a user explicitly delegates to the zero address.

Update: *Resolved in pull request #57 at commit ac5f5e5.*

L-03 Multiple Emissions of `DelegateChanged` Event

In the OpenZeppelin voting implementation, the emission of the `DelegateChanged` event occurs once per call to `delegate` and contains information about the delegator, the previous delegatee, and the new delegatee. This codebase seeks to build upon full delegation with the additional feature of partial delegation. Here, the emission of the `DelegateChanged` event replaces the information about the previous delegatee with a numerator which denotes the percentage of votes going to that specific delegatee. One of these events is emitted per each change of the delegatee.

With `MAX_PARTIAL_DELEGATIONS` set to 100, this could result in 200 events being emitted for one call to `delegate`. Aggregating these events to figure out basic information, such as which delegatees were changed, could be challenging for any nodes reading these events. Furthermore, with the absence of information regarding the previous delegatee, this could be even more difficult.

Consider emitting one event that takes in an array of the previous partial delegations and the new array of partial delegations. While this would break the signature for the `DelegateChanged` in EIP-5805, the current event already breaks this since the last parameter is of type `uint96` instead of `address`.

Update: *Resolved in pull request #60 at commit b617591 and in pull request #67 at commit c881700.*

L-04 Misleading Documentation

In the codebase, there are a few instances of misleading documentation:

- In line 305 of `VotesPartialDelegationUpgradeable`, the comment states that this function delegates "all of the voting units to delegates". However, this function can be called when only delegating some of the voting units.

- In line 315 of `VotesPartialDelegationUpgradeable`, the comment states that this portion of the code "subtracts votes from the old delegatee set". However, this portion of the code does not do any subtraction.

Consider revising the comments to provide more clarity to developers, auditors, and any other entities that will be reading the codebase.

Update: *Resolved in pull request #59 at commit e5c64c9.*

Notes & Additional Information

N-01 Constant Not Using `UPPER_CASE` Format

In `VotesPartialDelegationUpgradeable.sol`, the `VotesPartialDelegationStorageLocation` constant is not declared using the `UPPER_CASE` format.

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved in pull request #61 at commit ed61553.*

N-02 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `ERC20VotesPartialDelegationUpgradeable` abstract contract.

- The `IERC5805Modified` interface.
- The `IVotesPartialDelegation` interface.
- The `L2GovToken` contract.
- The `VotesPartialDelegationUpgradeable` abstract contract.

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved in pull request #65 at commit 826e0d4.*

N-03 dd Functionality to Cancel Signed Messages

By calling the `delegateBySig` or `delegatePartiallyOnBehalf` functions of the `VotesPartialDelegationUpgradeable` contract, an account can receive delegates from a token holder, which are used for voting.

Given that this functionality is highly sensitive, consider adding a `cancelDelegateSig` function which the user could call to cancel the signed message (e.g., by increasing their nonce).

Update: *Resolved in pull request #66 at commit f21bc39.*

N-04 Misleading Interface Name

The name of the `IERC5805Modified` interface suggests that while it has been modified, it still adheres to ERC-5805. However, this interface inherits from `IVotesPartialDelegation`, which has removed the `delegates` function. The specification states that the `delegates` function MUST be implemented. Therefore, this interface does not enforce ERC-5805.

While the comments in `IERC5805Modified` do clarify that this interface mostly only supports the ERC-5805 standard, the interface name can still be misleading.

Consider renaming this interface to avoid causing any confusion.

Update: *Resolved in pull request #62 at commit d3edd01.*

N-05 Custom Errors Inconsistent with Solidity Style Guide

In the `VotesPartialDelegationUpgradeable` abstract contract, there are custom errors located in these and these lines of code. These custom errors sandwich other parts of the code, such as enum and struct definitions, constant declarations, and a function.

Consider following the Solidity Style Guide layout and placing all custom errors at the appropriate location.

Update: *Resolved in pull request #63 at commit 1bd4d23.*

N-06 Gas Optimizations

Throughout the codebase, there are some instances of code that can be optimized to improve gas usage:

- The following variables in `VotesPartialDelegationUpgradeable.sol` are initialized to their default values. Consider leaving them uninitialized.
  - The `i` variable
  - The `i` variable
  - The `i` variable
  - The `i` variable
  - The `i` variable
  - The `j` variable
  - The `_totalNumerator` variable
  - The `i` variable
- The postfix increment operator is used in `VotesPartialDelegationUpgradeable.sol` for the following variables. Consider using the prefix increment operator `++i` instead. Alternatively, consider using Solidity version 0.8.22 or greater.
  - The `i++`
  - The `i++`
  - The `i++`
  - The `i++`
  - The `i++`
  - The `i++`

- Array lengths in exit conditions are read out on each loop iteration in the following examples. Consider saving the respective array lengths to a local variable beforehand, thereby preventing repeated `length` reads on each iteration.
  - The `_partialDelegations.length`
  - The `_newDelegations.length`
  - The `_from.length`
  - The `_to.length`
  - The `_delegations.length`
- The call to `_transferVotingUnits` occurs before the `if` block. Consider placing this call after the `if` block so that whenever the supply is greater than the cap, the revert would happen earlier in the execution.
- The old delegatees are obtained in line 316 of `VotesPartialDelegationUpgradeable` and again in line 334. Consider reusing the same local variable.

Consider the above recommendations to reduce gas usage.

Update: *Resolved in pull request #64 at commit 526d4ca.*

Conclusion

Agora's innovation has been aimed at providing more flexibility in vote delegation to allow a token holder to delegate to multiple addresses. The audit uncovered several low-severity issues while various recommendations aimed at improving code clarity and gas consumption have been made. We appreciate the Agora team's support and willingness to answer our questions in a timely manner throughout the audit period.