OpenZeppelin | security

OpenZeppelin Contracts Release v5.3 Diff Audit

April 3, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-03-03 To 2025-03-12
Languages	Solidity

Total Issues	5 (3 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (1 resolved, 1 partially resolved)
Notes & Additional Information	2 (2 resolved)

Scope

We audited the OpenZeppelin/openzeppelin-contracts repository at commit d4b2e98. This commit was diffed against commit acd4ff7 and all new files were fully audited. In scope were the following files:

```
contracts
├── access/manager/AuthorityUtils.sol
├── account/utils/draft-ERC4337Utils.sol
├── governance
│   ├── Governor.sol
│   ├── IGovernor.sol
│   └── extensions
│       ├── GovernorProposalGuardian.sol
│       ├── GovernorSequentialProposalId.sol
│       ├── GovernorSuperQuorum.sol
│       ├── GovernorVotesQuorumFraction.sol
│       └── GovernorVotesSuperQuorumFraction.sol
├── interfaces/draft-IERC6909.sol
├── metatx/ERC2771Forwarder.sol
├── proxy/utils/Initializable.sol
├── token
│   ├── ERC20/extensions/ERC4626.sol
│   ├── ERC20/utils/SafeERC20.sol
│   └── ERC6909
│       ├── draft-ERC6909.sol
│       └── extensions
│           ├── draft-ERC6909ContentURI.sol
│           ├── draft-ERC6909Metadata.sol
│           └── draft-ERC6909TokenSupply.sol
└── utils
    ├── Calldata.sol
    ├── Pausable.sol
    ├── Strings.sol
    ├── cryptography
    │   ├── Hashes.sol
    │   └── MessageHashUtils.sol
    ├── math/Math.sol
    └── structs
        ├── EnumerableMap.sol
        ├── EnumerableSet.sol
        └── MerkleTree.sol
```

Update: *All resolutions and the final state of the audited codebase mentioned in this report are contained at commit f3f0a64 on branch release-v5.3. In addition to the fixes identified in this report, this commit also includes the following changes:*

- *Fix to signature verification in P256.sol introduced at version 5.1.0 due to RIP-7212 specifications for non-existent precompiles.*
- *Minor codebase changes introduced via PR5605 related to addition of Entrypoint v0.8.*

Please note that changes introduced to contracts not in-scope for this audit were not part of the additional reviews.

System Overview

Version 5.3 of the OpenZeppelin Contracts library introduces new features, including support for ERC-6909, optional enhancements for governance, and updated functionality for certain utilities.

ERC-6909

The ERC-6909 standard strives to create a minimal specification for multi-token contracts, which removes some of the features of ERC-1155. Specifically, callbacks and batching have been removed from the interface. This new release follows the ERC specification to provide developers with the fundamental smart contracts necessary for this standard, as an alternative to the more complex ERC-1155.

Governance

This release introduces new governance features, including the ability to add guardians, track proposal IDs sequentially, and implement a super quorum. The first feature allows for the addition of a guardian, which is a privileged entity that can cancel proposals at any time. The second feature is the introduction of a sequential proposal ID system, which includes a new helper function that assists with the transition from a hash-based proposal identification system to a sequential one. Lastly, the super quorum feature allows proposals with sufficient "for" votes to bypass the waiting period and move directly to the "succeeded" state. These features have been implemented in new contracts that inherit from the existing governance ones.

Utilities

In this release, changes have been made to three libraries: `EnumerableMap`, `EnumerableSet`, and `MerkleTree`. For the first two, functionality to clear the existing data structure was added. Furthermore, a new struct named `Bytes32x2Set` was added in order to store tuples of values. For the Merkle tree, update functionality was added so that the caller can update a leaf of the Merkle tree.

Security Model and Trust Assumptions

Auditing libraries requires a shift in focus due to their composability within blockchain protocols. While the scope of an audit is typically limited to the code itself, the scope expands when it comes to libraries because of their potential internal and external integrations. Libraries act as foundational components for many protocols. This means that their security is influenced not just by their internal robustness, but also by how they are utilized by integrators. As a result, ensuring a library's security involves reviewing the code as well as anticipating its various use cases and integration scenarios.

In addition to the above, the complexity grows because, while a library must accommodate a wide range of potential use cases, the responsibility for secure implementation often falls on developers who integrate it into their projects. These developers must carefully review the security considerations when extending contracts from the library. A library's security risks can multiply depending on how well developers understand and utilize its contracts. Therefore, extra care is necessary to identify and address all potential threats, both direct and indirect, or to document them so that developers are fully aware of the associated security risks.

Privileged Roles

With the introduction of the `GovernorProposalGuardian` governance extension, a new guardian role has been added. As outlined in the System Overview section, this role can cancel a proposal in any state if it has not been executed, expired, or canceled already. However, if the guardian is not set, this privilege is passed to the person who made the proposal that is being canceled. Note that without this extension, the proposer can only cancel their proposal during the "pending" state.

Low Severity

L-01 `canCallWithDelay` Change Not Functionally Equal

The `canCallWithDelay` function of the `AuthorityUtils` library helps call `canCall` on an `authority` address, and can handle both a `(bool)` and `(bool, uint32)` response. The 5.3 release introduced a code change in how the response is decoded. Previously, the call's return data was `abi.decode`d depending on the data length. This causes reverts if the second value exceeds the `uint32` size.

In the new version, the return values are simply cast to their expected type. This means that a value exceeding the `uint32` size would wrap around to a value modulo 2^{32} and not revert. This wrapped `uint32` value, which represents the delay of a call, could then cause unintended behavior in `AccessManaged` contracts.

Consider checking the return size of the `uint32` value before casting to enforce the expected interface and to be consistent with the behavior of the previous version.

Update: *Resolved in pull request #5584 at commit d1efb23. The Solidity Contracts team stated:*

> *The fix is not going back to the previous behavior (that reverted) but instead making sure that the modulo cannot cause a high delay to be interpreted as a small delay. We changed it so that if the value does not fit in a uint32, then we use a 0 delay. This as the effect of disabling any delay that is not a uint32.*

L-02 ERC-6909 Total Supply Assumption Can Be Overridden

The ERC-6909 total supply extension assumes the following to perform an `unchecked` total supply decrement when burning a token:

> `amount <= _balances[id][from] <= _totalSupplies[id]`

This assumption is enforced in the base contract by reverting with the `ERC6909InsufficientBalance` error if the amount exceeds the `from` address balance. Furthermore, the total supply per ID is incremented on each mint such that the total supply must be at least equal or larger than any of the users' balances. However, a custom token can potentially override the `_update` logic such that this assumption no longer holds. This could lead to the total supply underflowing upon a burn.

Consider removing the `unchecked` block around the total supply decrement on burn.

Update: *Acknowledged, not resolved. The Solidity Contracts team stated:*

> *This behavior is simmilar to the one we have in `ERC1155` and `ERC1155Supply`.*
>
> *While an overriden version of `_update` could selectivelly execute `ERC6909._update` and `ERC6909TokenSupply._update` to try to create an inconsistency, we beleive this is unlikelly to happen in real usage of the contract.*
>
> *In this case we prefer optimizing gas usage of the huge majority of "clean" inheritance (including overrides that simply call `super._update` as we recomand in our good practices). Complex (and malicious) overrides fall under the responsability of the developper.*

L-03 Missing and Misleading Documentation

Throughout the codebase, multiple instances of misleading comments were identified:

- Both the `approve` function in `draft-ERC6909.sol` and the corresponding interface lack documentation on how to make infinite approvals, which is to set the allowance to `type(uint256).max`.
- The docstring on the `_transfer` function in `draft-ERC6909.sol` is contradictory as it first suggests an override, but later states that `_update` should be overridden for custom functionality instead.
- The ERC-6909 spec does not specify whether allowance should be consumed despite the operator status. The implementation is opinionated by *not* consuming the allowance for operators, which could be documented.
- In `GovernorVotesSuperQuorumFraction.sol`, it is not explicitly mentioned that only "for" votes count towards the super quorum. While this is mentioned in the `GovernorSuperQuorum` parent contract, consider explicitly mentioning this in `GovernorVotesSuperQuorumFraction` as well for clarity.

- The comment in line 29 of `draft-ERC6909TokenSupply.sol` states "_balances[id][from]", but it should be changed to state "_balances[from][id]".

Consider adding comments and revising the aforementioned ones to improve consistency and more accurately reflect the implemented logic, making it easier for auditors and other parties examining the code to understand what each section of code is designed to do.

Update: *Partially resolved in pull request #5584 at commit 803dff6. Bullet point 3 regarding additional ERC-6909 allowance documentation was not addressed.*

Notes & Additional Information

N-01 Typography Errors

Throughout the codebase, multiple instances of typography errors were identified:

- The documentation of the `Bytes32x2Set` struct and functions refers to itself as "in the self" [1, 2, 3], "to a self", "from a self", and "on the self". Consider removing "a" or "the", or changing "self" to "set".
- The comment in line 181 of `MerkleTree.sol` states "for the leaf being update". Consider changing it to "for the leaf being updated".

Consider correcting the above typographical errors to improve the clarity and readability of the codebase.

Update: *Resolved in pull request #5584 at commit 24ee8a6 as well as commit bfdbb67. The Solidity Contracts team stated:*

> *`Bytes32x2Set` was removed in commit bfdbb67.*

N-02 `escapeJSON` Escapes All Unicode Characters

In order to prevent JSON injection, the `escapeJSON` function of the `Strings` library takes a JSON string as the input to then escape certain characters (e.g., in NFT metadata). Thus,

control characters, backslashes, and quotation marks are escaped by prepending a backslash. This prevents breaking out of the value's string context and adding additional key-value pairs of the JSON object.

Regarding Unicode characters specifically, defined as `\uXXXX` (where `XXXX` is a 2-byte hexadecimal value), section 2.5 of RFC-4627 requires escaping for specific ranges:

- Control Characters: U+0000 to U+001F
- Quotation Marks: U+0022
- Backslash ("reverse solidus"): U+005C

The `escapeJSON` function does not handle these cases separately but instead escapes all backslashes and, therefore, also **all** Unicode characters. This behavior, while compliant with the spec, can be inconvenient for front-end applications that rely on native Unicode representations. Extra escaping forces these systems to perform additional decoding steps, which can lead to rendering inconsistencies or increased processing overhead. In contexts where precise handling and display of Unicode characters are critical - such as in languages that require specific symbols - the automatic doubling of backslashes may introduce challenges for correct interpretation and localization.

While the existing approach is secure, consider relaxing the escape rule by not escaping Unicode characters for the allowed ranges. Alternatively, consider documenting this escape rule in the codebase for improved clarity.

Update: *Resolved in pull request #5584 at commit 02a6c25. The Solidity Contracts team decided to add a comment about this behavior.*

Conclusion

The v5.3 release of OpenZeppelin Contracts introduces support for ERC-6909, optional enhancements for governance, and updated functionality for certain utilities. We commend the Solidity Contracts team for addressing user needs by incorporating new standards, enhancing existing features, and adding new utilities.

During the audit, particular care was taken to document edge cases, ensuring that integrators are informed of potential risks when interacting with these contracts. Such efforts aim to create a more resilient codebase, recognizing the library's critical role as a foundational component within the blockchain ecosystem. The Contracts team has demonstrated a strong commitment to maximizing the library's security, and we are glad to have collaborated with them on this milestone.