OpenZeppelin | security

OpenZeppelin Contracts Release v5.4 Diff Audit

July 14, 2025

Table of Contents

Summary

Type	Library
Timeline	From 2025-06-16 To 2025-07-02
Languages	Solidity

Total Issues	20 (5 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	8 (2 resolved)
Notes & Additional Information	11 (2 resolved, 2 partially resolved)
Client Reported Issues	1 (1 resolved)

Scope

OpenZeppelin audited the OpenZeppelin/openzeppelin-contracts repository at commit f6fea857 (release v5.4). This commit was compared with commit e4f7021 (release v5.3) and all new files were fully audited.

In scope were the following files:

```
contracts
├── access
│   ├── extensions
│   │   ├── AccessControlDefaultAdminRules.sol
│   │   └── AccessControlEnumerable.sol
│   ├── manager
│   │   ├── AccessManaged.sol
│   │   └── IAccessManager.sol
│   └── AccessControl.sol
├── account
│   ├── extensions
│   │   ├── draft-AccountERC7579.sol
│   │   ├── draft-AccountERC7579Hooked.sol
│   │   └── ERC7821.sol
│   ├── interfaces
│   │   └── IERC7913.sol
│   ├── utils
│   │   └── draft-ERC7579utils.sol
│   └── Account.sol
├── governance
│   ├── extensions
│   │   ├── GovernorCountingFractional.sol
│   │   ├── GovernorCountingOverridable.sol
│   │   ├── GovernorCountingSimple.sol
│   │   ├── GovernorNoncesKeyed.sol
│   │   ├── GovernorSequentialProposalId.sol
│   │   ├── GovernorSettings.sol
│   │   ├── GovernorSuperQuorum.sol
│   │   ├── GovernorTimelockAccess.sol
│   │   ├── GovernorTimelockCompound.sol
│   │   ├── GovernorTimelockControl.sol
│   │   └── GovernorVotes.sol
│   ├── Governor.sol
│   └── TimeLockController.sol
├── interfaces
│   └── draft-ERC7821.sol
├── token
│   ├── common
│   │   └── ERC2981.sol
│   ├── ERC20
```

```
|   |   ├── extensions
|   |   |   ├── ERC20capped.sol
|   |   |   ├── ERC20Permit.sol
|   |   |   ├── ERC20Wrapper.sol
|   |   |   ├── ERC1363.sol
|   |   |   └── ERC4626.sol
|   |   ├── utils
|   |   |   └── ERC1363Utils.sol
|   |   └── ERC20.sol
|   ├── ERC721
|   |   ├── extensions
|   |   |   ├── ERC721Enumerable.sol
|   |   |   ├── ERC721Royalty.sol
|   |   |   └── ERC721URIStorage.sol
|   |   ├── utils
|   |   |   └── ERC721Utils.sol
|   |   └── ERC721.sol
|   └── ERC1155
|       ├── extensions
|       |   └── ERC1155Supply.sol
|       ├── utils
|       |   ├── ERC1155Holder.sol
|       |   └── ERC1155Utils.sol
|       └── ERC1155.sol
└── utils
    ├── cryptography
    |   ├── signers
    |   |   └── draft-ERC7739.sol
    |   ├── AbstractSigner.sol
    |   ├── draft-ERC7739.sol
    |   ├── EIP712.sol
    |   ├── MultiSignerERC7913.sol
    |   ├── MultiSignerERC7913Weighted.sol
    |   ├── SignatureChecker.sol
    |   ├── SignerECDSA.sol
    |   ├── SignerERC7702.sol
    |   ├── SignerERC7913.sol
    |   ├── SignerP256.sol
    |   └── SignerRSA.sol
    ├── introspection
    |   └── ERC165.sol
    ├── structs
    |   ├── EnumerableMap.sol
    |   └── EnumerableSet.sol
    ├── Address.sol
    ├── Arrays.sol
    ├── Base64.sol
    ├── Blockhash.sol
    ├── Bytes.sol
    └── Strings.sol
```

System Overview

Version 5.4 of the OpenZeppelin Contracts library introduces new features, including support for ERC-7913, ERC-7821, EIP-2935, a keyed-nonces extension to the governance module, enhancements to the enumerable map and set libraries, and porting of several contracts from the OpenZeppelin/openzeppelin-community-contracts repository. In addition, multiple files were updated with trivial modifications, such as improved documentation, imports, and minor refactoring.

ERC-7913 (Signatures by Address-Less Keys)

The `SignerERC7913` contract inherits `AbstractSigner` and implements signature verification for address-less keys following the ERC-7913 standard. A multi-signature signer system using multiple ERC-7913 signers is implemented in the `MultiSignerERC7913` contract, providing the following functionality:

- Manage a dynamic set of signers (e.g., EOA or smart contract signers).
- Require a threshold number of valid signatures for authorizing operations.
- Be compatible with both ECDSA (EOA) signatures and ERC-1271 smart contract signatures.

The new functionality allows clients to build accounts or wallets where actions must be jointly authorized by multiple parties.

The `MultiSignerERC7913` contract is further extended by `MultiSignerERC7913Weighted`, which adds support for weighted signatures. Specifically, it assigns different positive weights to each signer, enabling more flexible governance schemes. For example, some signers could have a higher weight than others, allowing for weighted voting or prioritized authorization. The threshold for validation is reached when the sum of the weights of the unique signers of a message crosses the threshold value.

Another relevant new contract is `SignatureChecker`, which is a helper for streamlining the verification of ECDSA, ERC-1271, and ERC-7913 signatures.

ERC-7821 (Batch Executor Interface)

The `ERC7821` contract (and its associated interface `IERC7821`) implements the ERC-7821 standard for batch execution. The latter defines a standardized interface for executing one or more function calls atomically and is intended for use by smart contract wallets, EIP-7702 EOAs, and ERC-4337 smart accounts.

EIP-2935 (Access to Historical Block Hashes)

The `Blockhash` contract implements block hash access beyond the 256-block limit following the EIP-2935 standard. It preserves support for the native `BLOCKHASH` opcode as a fallback. Specifically, the ability to retrieve block hashes is extended to 8191 blocks in the past. In case the block is within the last 256 blocks, the native `BLOCKHASH` EVM opcode is used. In case the block is between 256 and 8191 blocks ago, an EVM-reserved history storage contract at a hard-coded address is called, which keeps a record of the last 8191 block hashes in its state. For all other blocks (including future blocks), the return value is zero.

Keyed Nonces

The `GovernorNoncesKeyed` contract inherits `Governor` and `NoncesKeyed` and implements the *keyed nonce* abstraction when voting by signature. Traditional, un-keyed nonces prevent the replaying of votes on proposals. On the other hand, keyed nonces provide a kind of domain separation by proposal ID, so that many nonces can be incremented in parallel for different proposals. In the `GovernorNoncesKeyed` contract, this is achieved by using the first 192 bits of the `proposalID` as the key.

Enhancements to `EnumerableMap` and `EnumerableSet`

Enhancements were made to the `EnumerableMap` and `EnumerableSet` libraries. These implement an enumerable version of Solidity's native mapping and an enumerable set data structure, respectively. Specifically, in `EnumerableMap`, for all supported maps, a new `keys` function has been added that returns a slice of the keys stored in the map, as opposed to the full list of keys. In addition, a new map, `BytesToBytesMap`, along with all associated methods supported by the other maps, has been implemented. In `EnumerableSet`, to all supported sets, a new `values` function has been added for sliced access to the values in the set, along with two new sets, `Bytes` and `StringSet`, and their associated methods.

Ports from Community Contracts

Multiple contracts were moved from the OpenZeppelin/openzeppelin-community-contracts repository after minor modifications. These include verification of ECDSA, RSA, and P-256 signatures, libraries providing utility functions for ERC-7739 and ERC-7579, and a contract implementing EIP-712.

Trivial Modifications

All files in scope that are not related to any of the new functionalities mentioned in the preceding sections contain only trivial modifications such as documentation updates, imports, and minor refactoring.

Security Model and Trust Assumptions

During the audit, a trust assumption was made that the `HISTORY_STORAGE_ADDRESS` hardcoded value in the `Blockhash` library points to the correct EVM precompile.

Low Severity

L-01 Possible Overflow when Computing the Total Weight of ERC-7913 Multisigners

The `totalWeight` function returns a `uint64` value by casting `getSignerCount() + _totalExtraWeight` to `uint64`. However, `getSignerCount` returns a `uint256` value. Therefore, theoretically, an overflow may occur before the cast operation. This will happen if the number of signers or their total weight `_totalExtraWeight` (or both) is close to or larger than `2^64-1`, which is not likely to occur in practice.

Regardless of the low risk, consider reverting in case of an overflow.

Update: *Resolved in pull request #5790. The team stated:*

> *The totalWeight() does the addition of :*
>
> *- `getSignerCount()` (an uint256)*
>
> *- `_totalExtraWeight` (an uint64)*
>
> *This add operation is done on the "uint256 space". If `getSignerCount()` is close to `2^256-1`, and if `_totalExtraWeight` makes up for the difference, then that addition could technically overflow.*
>
> *However, we are using solidity ^0.8.27 for this file. Since solidity 0.8.0 all arithmetic operations are "checked" by default. Here there is no `unchecked` block, so any overflow (prior to the casting) will revert. Solidity takes care of that.*
>
> *Said otherwise: the compiler already performs overflow detection here. Doing it ourselves would just duplicate the verification cost with no upside.*
>
> *The only way we see having `totalWeight()` overflow though is by first setting the signer weights so that `totalWeight()` is almost `2^64` and then add new signers (with a default weight of 1). We address this scenario with the fix in the provided pull request.*

L-02 `_setSignerWeights` Does Not Check if the Weight Has Changed

When calling the `_setSignerWeights` function in `MultiSignerERC7913Weighted.sol`, it is possible to pass in a list of signers and weights where one or more entries are equal to the current signer and weight. For example, suppose `signer1` is a `bytes` object and `_extraWeights[signer1] == 1` so that `signer1` has a weight of 2, and we call `_setSignerWeights([signer1], [2])`. Then, the following happens:

1. `extraWeightRemoved` is incremented by 1.
2. `extraWeightAdded` is incremented by 1.
3. `ERC7913SignerWeightChanged(signer1, 2)` is emitted.
4. `_totalExtraWeight` is updated to the same value it was before.
5. `_validateReachableThreshold()` is called.

In the above scenario, unnecessary computations are performed and events are emitted without a state change.

To save gas and avoid unnecessary event emissions, consider validating that for each `i < signers.length`, the weight to be set for the signer is equal to the signer's current weight.

Update: *Resolved in pull request #5775.*

L-03 Name for Elements in Set Struct is Potentially Confusing

Within the `Set` struct of the `EnumerableSet` library, the elements of the set are stored as a `bytes32[]` array named `_values`. The corresponding `private` getter functions `_values(set)` and `values(set, start, end)`, along with the `internal` getter functions `values(set)` and `values(set, start, end)` for structs that wrap the `Set` struct (e.g., `Bytes32Set`), are also named similarly.

While this hews closely to the terminology of mappings, it has the unfortunate consequence that, in the keys function for the `Bytes32ToBytes32Map` map in `EnumerableMap.sol` and throughout the file, there is phrasing such as `return map._keys.values(start, end);`. However, this does not relate to the values of the enumerable map. Instead, it returns a slice of the set of keys. This is likely to cause some confusion between the values of the outer enumerable map with the "values" of the inner enumerable set.

Since "element" is the standard mathematical term for a member of a set, adopting this terminology will help emphasize the abstraction of the `Set` data structure. In addition, this can help differentiate between `Set`s from Solidity mappings and each of the enumerable map structs given in `EnumerableMap.sol`. As such, consider renaming the `_values` field of the `Set` struct, as well as the associated getter functions of `Set` and other structs that wrap it, to `elements`.

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

> *Changing from `values()` to `elements()` would be a breaking API change affecting all EnumerableSet types, requiring ecosystem-wide updates.*
>
> *However, the current naming is already reasonably clear, `map._keys.values()` indicates we're accessing values from the keys collection. Since users shouldn't directly access internal structures like `_keys` anyway, optimizing clarity for discouraged usage patterns isn't a priority.*
>
> *The change would also require upgradeable plugin configuration and impose significant migration costs for minimal practical benefit. We'll keep the current naming for now, since the theoretical improvement doesn't justify the breaking change impact.*

L-04 Missing External Call Failure Check

To satisfy EEA EthTrust Security Level [S], code that makes external calls using the low-level call functions (i.e., `call`, `delegatecall`, `staticcall`, `send`, and `transfer`) MUST check the returned value from each usage to determine whether the call failed. Normally, exceptions in subcalls 'bubble up', unless they are handled in a `try-catch` block. However, Solidity defines a set of low-level call functions that do not have built-in safety checks: `call`, `delegatecall`, `staticcall`, `send`, and `transfer`. Calls made using these functions behave differently. Specifically, they return a boolean indicating whether the call completed successfully. As such, not explicitly testing the return values of these calls for failure may lead to unexpected behavior in the caller contract.

The `staticcall(gas(), HISTORY_STORAGE_ADDRESS, 0x00, 0x20, 0x20, 0x20)` call within the `_historyStorageCall` contract in `Blockhash.sol` is missing a failure check.

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

> *The `staticcall` to `HISTORY_STORAGE_ADDRESS` has guaranteed success semantics by design:*

> *- If EIP-2935 code exists at the address, it follows the specification and never reverts*
>
> *- If no code exists, the call succeeds with empty return data*
>
> *In both cases, the success boolean is always `true`. Adding a redundant check would consume gas without providing any safety benefit. This is an acceptable exception to the general rule given the well-defined behavior of EIP-2935.*

L-05 Variable Names Too Similar

Similar variable names make the code challenging to read and maintain, and can introduce confusion during the auditing process. Within `MultiSignerERC7913.sol`, `signature` is similar to `signatures`. Other than the similarity, another argument in favor of renaming is that the `signature` array does not contain only signatures. In fact it encodes two arrays - a `signatures` and a `signers` array.

Consider renaming the variables using clear and descriptive variable names and adhering to a naming convention.

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

> *We're keeping the naming, since the team agrees that its distinction is semantically meaningful: `signature` represents the composite multisig signature inspired by ERC-1271, while `signatures` are the individual signatures that compose it.*
>
> *This naming conveys the conceptual relationship between the aggregated signature and its parts. The variables serve distinctly different purposes in the multisig verification process, making the current naming both logical and maintainable.*

L-06 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified:

- In `draft-ERC7821.sol`, in the `execute` function, the `mode` and `executionData` parameters are not documented.
- In `IERC7913.sol`, in the `verify` function:
  - the `key`, `hash`, `signature` parameters are not documented
  - not all return values are documented
- In `draft-IERC7821.sol`, in the `execute` function, the `mode` and `executionData` parameters are not documented.

- In `draft-IERC7821.sol`, in the `supportsExecutionMode` function:
  - the `mode` parameter is not documented
  - not all return values are documented
- In `MultiSignerERC7913.sol`, in the `ERC7913SignerAdded` event, the `signers` parameter is not documented.
- In `MultiSignerERC7913.sol`, in the `ERC7913SignerRemoved` event, the `signers` parameter is not documented.
- In `MultiSignerERC7913.sol`, in the `ERC7913ThresholdSet` event, the `threshold` parameter is not documented.
- In `MultiSignerERC7913.sol`, in the `getSigners` function:
  - the `start` and `end` parameters are not documented
  - not all return values are documented
- In `MultiSignerERC7913.sol`, in the `getSignerCount` function, not all return values are documented.
- In `MultiSignerERC7913.sol`, in the `isSigner` function:
  - the `signer` parameter is not documented
  - not all return values are documented
- In `MultiSignerERC7913.sol`, in the `threshold` function, not all return values are documented.
- In `MultiSignerERC7913Weighted.sol`, in the `ERC7913SignerWeightChanged` event, the `signer` and `weight` parameters are not documented.
- In `MultiSignerERC7913Weighted.sol`, in the `signerWeight` function:
  - the `signer` parameter is not documented
  - not all return values are documented
- In `MultiSignerERC7913Weighted.sol`, in the `totalWeight` function, not all return values are documented.
- In `SignerERC7913.sol`, in the `signer` function, not all return values are documented.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

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

> *The documentation pattern is intentionally consistent across the entire OpenZeppelin library. Parameter and return value documentation is omitted because our documentation engine doesn't render these NatSpec elements.*

> *Adding parameter/return documentation would create inconsistency with the established codebase style without providing user-facing benefits. The current approach prioritizes meaningful function descriptions over unused documentation elements.*

L-07 Possible Duplicate Event Emission

When a setter function does not check if the incoming value is different from the existing one, it opens up the possibility of spamming events that indicate a change even when no actual change has occurred. Spamming identical values may confuse off-chain clients that rely on event data to track state changes.

Within `MultiSignerERC7913.sol`, the `_setThreshold` function sets the `_threshold` value and emits an event without checking if the value has changed.

Consider adding a check statement to revert the transaction if the incoming value is identical to the existing one.

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

> *OpenZeppelin contracts consistently allow no-ops that emit events (e.g., zero-value transfers, unchanged approvals). Adding threshold-specific checks would break this design pattern.*
>
> *The current behavior maintains consistency across the codebase.*

L-08 Different Pragma Directives

In order to clearly identify the Solidity version with which the contracts will be compiled, pragma directives should be fixed and consistent across file imports.

Throughout the codebase, multiple instances of different pragma directives were identified:

- `draft-ERC7821.sol` has the pragma directive `pragma solidity ^0.8.20;` and imports the file `draft-IERC7821.sol`, which has a different pragma directive.
- `GovernorNoncesKeyed.sol` has the pragma directive `pragma solidity ^0.8.24;` and imports the file `NoncesKeyed.sol`, which has a different pragma directive.
- `MultiSignerERC7913Weighted.sol` has the pragma directive `pragma solidity ^0.8.27;` and imports the file `MultiSignerERC7913.sol`, which has a different pragma directive.

Consider using the same fixed pragma version across all the files.

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

> *We try to minimise the pragma each file uses. If a contract only doesn't need any feature introduced after 0.8.20, we have no reason to not mark it ^0.8.20. We use 0.8.20 as our minimum (for non-interface files), because it introduced push0 that we believe to be a great optimisation.*
>
> *Some files will depend on a file that uses ^0.8.20 but wil also use additional feature that were added to the language later. In that case we use a pragma that reflect this requirement.*
>
> *We have test in place to ensure that: all contract can be compile with the pragma being used (for example not contract that uses ^0.8.20 will be compilable with 0.8.20, and won't use any feature introduced after that)*
>
> *A consequence of this test, is that if B imports A, B's pragma will be equal or more restrictive than A's pragma.*

Notes & Additional Information

N-01 Minor Inconsistencies in the Implementations of Map and Set

Multiple instances of minor inconsistencies in the implementation of the methods for `BytesToBytesMap` and other maps (e.g., `Bytes32ToBytes32Map`) were identified:

- The at function for `BytesToBytesMap` differs from the at function for `Bytes32ToBytes32Map` in that it has an implicit `return`.
- The `tryGet` function for `BytesToBytesMap` differs from the `tryGet` for `Bytes32ToBytes32Map` in that it avoids the explicit `if-else` branches.
- The get function for `BytesToBytesMap` uses `tryGet` while the `get` function for `Bytes32ToBytes32Map` duplicates the logic of the corresponding `tryGet` for the type.

Similarly, in `EnumerableSet.sol`, the different methods for the `StringSet` and `ByteSet` types name the input set variable either as `set` or `self` (e.g., self vs. set and self vs. set). In contrast, the methods for the other types consistently use `set`.

Consider making the implementations consistent across the different map and set types.

Update: *Partially resolved in pull request #5776. The team stated:*

> *We partially addressed the issue by standardizing parameter naming (`self` → `set`) but kept other inconsistencies.*
>
> *BytesToBytesMap handles memory types differently than value-type maps, requiring distinct patterns like implicit returns and specialized memory handling. These differences are functional requirements, not stylistic inconsistencies that need correction.*

N-02 Inner Mapping Only Has One Named Parameter

In the `Bytes32ToBytes32Map` and `BytesToBytesMap` structs, the "key" parameter of the inner mapping is named `key`, but the "value" parameter is unnamed. Similarly, in the `Set`, `StringSet`, and `BytesSet` structs, the "key" parameter of the `_positions` mapping is named `value`, but the "value" parameter is unnamed.

For consistency, consider naming both parameters in each mapping.

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

> *Acknowledged. The naming convention is intentional and follows established patterns across OpenZeppelin contracts (e.g., ERC20's `_balances`, `_allowances`).*
>
> *Key parameters are named for clarity since they're often non-obvious. Value parameters remain unnamed when the mapping name itself clearly indicates the stored type (e.g., `_values` mapping obviously stores values). Adding redundant naming like `mapping(bytes32 key => bytes32 value) _values` would be unnecessarily verbose.*

N-03 Functions Updating State Without Event Emissions

The `_setSigner` function in `SignerERC7913.sol` updates the state without an event emission.

Consider emitting events whenever there are state changes to improve the clarity of the codebase and make it less error-prone.

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

> *This is true for all `Signer*.sol` contracts (`SignerECDSA`, `SignerP256`, `SignerRSA`). All have a function `_setSigner` to update the key that doesn't emit an event.*
>
> *These are low level abstractions, that should interfere as little as possible with the contracts that inherit from, them. In many cases, the key will be set at construction and never updated.*
>
> *If a user decides to expose the internal `_setSigner` function, they should emit an event themselves.*

N-04 Redundant `return` Statements

Functions that have named returns do not require explicit `return` statements.

Throughout the codebase, multiple instances of redundant `return` statements were identified:

- The `return callType == ERC7579Utils.CALLTYPE_BATCH && execType == ERC7579Utils.EXECTYPE_DEFAULT && modeSelector == ModeSelector.wrap(0x00000000);` statement in `draft-ERC7821.sol`
- The `return false;` statement in `MultiSignerERC7913.sol`
- The `return hash.areValidSignaturesNow(signers, signatures);` statement in `MultiSignerERC7913.sol`

To improve code clarity, consider removing `return` statements from functions that have named returns.

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

> *Not going to fix.*

> *Note that in the case of `MultiSignerERC7913.sol`, the `return false` is used to break the execution of the loop and force an early return. Writing false to the return variable and continuing the execution would just not work.*

N-05 Missing 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, multiple instances of contracts missing a security contact were identified:

- The `ERC7821` abstract contract
- The `GovernorNoncesKeyed` abstract contract
- The `IERC7913SignatureVerifier` interface
- The `IERC7821` interface
- The `Blockhash` library
- The `MultiSignerERC7913` abstract contract
- The `MultiSignerERC7913Weighted` abstract contract
- The `SignerERC7913` 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: *Acknowledged, not resolved. The team stated:*

> *This is not what the security contact comment is designed for.*
>
> *The security contact is something deployers / and users add to there code so that we (OZ) know how to contact them in case we ever identify their contract as vulnerable. When we identify issues with our code, we scan the chain for affected instances. For each instance, we want to be able to notify the admin (to tell them to pause/upgrade/...)*

> *of these apps. It's a way for people that import our code to get notification from us if there is an issue.*
>
> *In those circumstances, we don't want to end up with "contact openzeppelin".*

N-06 Lack of Indexed Event Parameter

Within `MultiSignerERC7913.sol`, the `ERC7913ThresholdSet` event does not have indexed parameters.

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

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

> *Indexing a parameter is expensive, and is useful only in specific circumstances.*
>
> *For example, in ERC20 transfer, indexing the from and to address is great for someone to easily filter through a large number of event, figuring the transfers that affect a particular account he/she is interested in. On the other hand, the value is not indexed, because there the cost is not really worth it as we don't expect anyone to say "I want all events that have this exact amount transfered, regardless of the sender and receiver".*
>
> *In the case of MultiSignerERC7913, we imagine that someone may want to listen to all the threshold updates, but we don't imagine that person would really want to filter these by value. It would mean "I want to know about threshold updates that set the value to X, but I don't care about threshold updates to any other values"*

N-07 File and Contract Names Mismatch

The `IERC7913.sol` file name does not match the `IERC7913SignatureVerifier` contract name.

To make the codebase easier to understand for developers and reviewers, consider renaming files to match the contract names.

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

> *With a few historical exceptions (that we keep for backward compatibility), the files in the interface folder usually refer to the ERC that standardize the interfaces. For example,*

> *IERC4337 is only one file even though it defines many interfaces (none of which are names IERC4337). The same is true of IERC7579.sol.*
>
> *Similarly IERC6909.sol defines multiple interfaces that fall in the scope or ERC-6909.*
>
> *For ERC-7913, we feel that, just like for ERC-7579:*
>
> *- the name of the file should only reflect the ERC name (like most other files in the folder)*
>
> *- the name of the contract has to be a bit more explicit as to what it does.*

N-08 Custom Errors in `require` Statements

Since Solidity version `0.8.26`, custom error support has been added to `require` statements. Initially, this feature was only available through the IR pipeline. However, Solidity `0.8.27` extended support to the legacy pipeline as well.

The `draft-ERC7821.sol` contains multiple instances of `if-revert` statements that could be replaced with `require` statements:

- The `if (!_erc7821AuthorizedExecutor(msg.sender, mode, executionData)) revert Account.AccountUnauthorized(msg.sender)` statement
- The `if (!supportsExecutionMode(mode)) revert UnsupportedExecutionMode()` statement

For conciseness and gas savings, consider replacing `if-revert` statements with `require` statements.

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

> *using `require(consition, customError(args));` may be more readable than `if(!condition) revert customError(args);` but it is actually more expensive.*
>
> *The reason is that in the case of the if, the custom error code (with the argument) is only encoded if the condition is not met. Using the require forces the custom error to always be encoded, even if its not going to be emitted.*
>
> *While we sometimes do use the new require format, it is not a requirement of our guidelines. We also feel it's sometimes better to leave more room to the user and have a more relaxed pragma.*

> *I don't think the arguments in favor of require are strong enough to justify this change as a fix.*

N-09 Documentation Improvements

Throughout the codebase, multiple opportunities for improving the documentation were identified:

1. The name of the `_validateThreshold` function suggests that the value of the multisig `_threshold` is validated. However, what is being validated is whether the number of collected signatures meets the threshold. Consider renaming the function to better reflect its purpose. Some suggestions would be `_validateNumberOfSignatures` or `_validateSufficientSignatures`.

2. In the `_validateVoteSig` function, an invalid keyed nonce signature will be validated a second time as well in the standard nonce scenario, which may seem redundant. However, this redundancy is well-motivated though due to the following reasons:

    - On the one hand, the library must support both standard and keyed nonce signatures, while on the other, it has no way of knowing which of the two is supported by the client. Therefore, both scenarios must be checked.
    - In the case of keyed nonces, the nonce is explicitly incremented only for valid signatures. In contrast, standard nonces are always incremented, regardless of whether the signature is valid or not. Therefore, by falling back to the standard nonce scenario for invalid keyed nonce signatures, an implicit increase of the nonce is forced even for invalid keyed nonce signatures. This is also a reason why the order in which standard vs. keyed nonce signatures are checked is important and should not be switched.

    Consider documenting the specifics of the above-discussed behavior.

3. In the ERC-7913 multisig setting, there may be cases in which a `_removeSigners` operation must be preceded by a respective threshold readjustment via `_setThreshold`. Indeed, consider a scenario of 10 signers with a threshold of 5 and a need to remove 6 signers. If the threshold is not readjusted first (to a value at most 4), the call to `_removeSigners` will revert due to the original threshold (5) being unreachable for 4 signers.

    However, the threshold must be lowered and the signers removed in the *same* transaction. Otherwise, a group of to-be-removed signers could take advantage of the

lowered threshold to approve a transaction before they are removed. Not having a combined "lower threshold and remove signers" function is a very reasonable design choice and improves modularity of the code. Yet, extra care should be exercised so that whoever imports the library does not make mistakes in using it. Consider documenting the fact that the threshold must be lowered, then the signers removed, in that order, in the same transaction.

4. In `MultiSignerERC7913.sol`, the documentation for `_validateSignatures` states that "*The signatures arrays must be at least as large as the signers arrays. Panics otherwise.*" However, `_validateSignatures` calls `areValidSignaturesNow(signers, signatures)`, which returns `false` if `signers.length != signatures.length`. Consider amending this comment to state that the two arrays must be equal in length.

5. In `EnumerableSet.sol` and `EnumerableMap.sol`, new structs (`StringSet`, `BytesSet`, and `BytesToBytesMap`) have been added that represent sets and mappings of elements of variable length. The documentation for the `add`, `remove`, `contains`, and `at` methods for `StringSet` and `BytesSet`, and the documentation for the `set`, `remove`, `contains`, `at`, `tryGet`, and `get` methods of `BytesToBytesMap`, describe these as O(1) operations. This is correct when there is a uniform upper bound on the length of the elements in the set.

   If the element lengths grow asymptotically with the size of the set, the time bound is O(m), where m is the length of the element being added, removed, or queried. This is true for queries and removals instead of just additions, because the input element must be read and hashed to find its position in the `_positions` mapping. Similarly, the time bound on previously O(n) operations `values` and `clear` of `StringSet`, `BytesSet`, and `BytesToBytesMap` is O(mn) instead of O(n). Consider checking the time bounds and updating the documentation for enumerable sets and mappings with variable length elements.

6. In `EnumerableSet.sol`, the warning is a run-on sentence. For clarity, consider breaking this into multiple sentences and updating each instance of this warning accordingly.

7. In the _validateVoteSig and _validateExtendedVoteSig functions, the key for the nonce is derived by casting (resp. here) the `proposalId` to a `uint192` using `uint192(proposalId)`. In Solidity, this operation retains the lower 192 bits. According to the documentation, the key corresponds to the "first 192 bits" of the `proposalId`, which may be ambiguous and may potentially be interpreted as the

higher-order bits rather than the lower-order bits. The ambiguity could lead to nonce collisions or unintended replay vulnerabilities if the `proposalId` exceeds 192 bits, because the intended high-order bits (which carry different significance) are ignored. Consider editing the documentation to state "low-order" bits and preferably giving an example.

Consider improving the documentation as per the aforementioned recommendations.

Update: *Partially resolved in pull request #5779. The team stated:*

> *Some of the recommendations are addressed in the linked PR. We however decided to not address everything raise here.*
>
> *On `_validateThreshold` naming: Generic function names are appropriate in abstract library contexts where the validation logic is meant to be overridden. The name describes the contract (threshold validation) rather than the implementation detail (signature counting). I'd suggest keeping it as is.*
>
> *On `_validateVoteSig` redundancy: The "redundant" validation is intentional architectural design. It enables graceful degradation from keyed nonces to standard nonces, allowing incremental adoption without breaking existing integrations. This provides better UX than requiring explicit mode selection. I'd say we keep it as is.*
>
> *On Internal API documentation level: Internal functions like `_setThreshold` and `_removeSigners` are building blocks for smart contract developers extending OpenZeppelin contracts, not end-user APIs. These developers create public wrapper functions that combine internal calls correctly and will undergo security audits regardless of documentation completeness. I'd say adding implementation guidance to internal functions doesn't add much value and is not an immediate security concern.*
>
> *On Time complexity as O(1): I would argue that the noted complexity still describes the correct relationship between the set size and the operation cost. In short, assuming constant size of the data inserted, then then it's correct to indicate O(1).*

N-10 Typographical Errors

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

- Here, "Returns whether whether" should be "Returns whether".
- Here, "Return the an array containing a slice of the keys" should be "Returns an array containing a slice of the keys". Also here and possibly at other places.

- Here, "Tries to returns the value associated with `key`." should be "Tries to return the value associated with `key`". Also here and possibly at other places.

Consider fixing all occurrences of typographical errors to improve code clarity and avoid misinterpretation.

Update: *Resolved in pull request #5777 at commit e1277f7.*

N-11 Missing `immediate` Parameter in `canCall` Function

In `IAccessManager.sol`, in the comment to the `canCall` function, the documentation refers to an `immediate` parameter that should allow bypassing the delay when set to true. However, the function signature only includes the parameters `caller`, `target` and `selector`. This discrepancy between the documentation and the actual function signature could lead to confusion about how the function is expected to behave and indicates a potential logical inconsistency.

Consider fixing the documentation or the implementation, so the two are consistent.

Update: *Resolved in pull request #5795.*

Client Reported

CR-01 Allowing Zero Threshold in ERC-7913 Multisigner

The `_setThreshold` function sets the value for the signature `_threshold`. However, there is no check enforcing the threshold to be non-zero. With a threshold of zero, `_validateThreshold` will always return `true` regardless of the number of signers. The risk is mitigated during signature verification in `_rawSignatureValidation` which forces the number of signatures to be non-zero and consequently calls `_validateSignatures` on at least one signature. In addition, the risk is further minimized by the fact that the value of threshold can only be set during through the internal function `_setThreshold`.

As a result of the aforementioned mitigating factors, there is little risk of any practical attack. However, having a zero threshold effectively defeats the purpose of any multisig scheme.

Therefore, consider enforcing the value of the threshold to be strictly larger than zero. Alternatively, consider documenting this behavior.

Update: *Resolved in pull request #5772 at commit d7930da.*

Conclusion

The v5.4 release of OpenZeppelin Contracts library introduces support for ERC-7913, ERC-7821, and EIP-2935, a keyed-nonces extension to the governance module, enhancements to the enumerable map and set libraries, and porting of several contracts from the Community Contracts repository, among other minor updates. This improves the versatility of the library and addresses user needs dictated by the latest standards.

The code quality is of a high standard, with multiple safety mechanisms in place to prevent malicious use, unintentional or otherwise. It is also accompanied by detailed documentation and usage instructions. Several low-severity issues were reported, along with multiple notes recommending various improvements.

The OpenZeppelin Contracts team is commended for the high quality of their work and is appreciated for their active engagement throughout the audit in addressing all the questions posed by the audit team promptly and in great detail.