OpenZeppelin | security

Contracts 4.9 Release Audit

OpenZeppelin

May 9, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	Lib a
Timeline	F m 2023-03-10 T 2023-05-03
Languages	S lidi

Total Issues	26 (18 e l ed, 1 a iall e l ed)
Critical Severity Issues	0 (0 e l ed)
High Severity Issues	0 (0 e l ed)
Medium Severity Issues	3 (2 e l ed)
Low Severity Issues	3 (3 e l ed)
Notes & Additional Information	20 (13 e l ed, 1 a iall e l ed)

Scope

We audited the OpenZeppelin/openzeppelin-contracts repository at two commits: - `fa112be6826debe8848223888b3d23746a6ede8f` commit for the contracts under the `access` subfolder - `ca822213f2275a14c26167bd387ac3522da67fe9` commit for all of the other contracts

The `7f5e91062e10fe7f715eab7e46154e5d445add78` commit, featured in this report, is a merge of the two commits mentioned above.

In scope were the following contracts:

```
contracts
├── utils
│   └── StorageSlot.sol
├── governance
│   ├── compatibility
│   │   ├── GovernorCompatibilityBravo.sol
│   │   └── IGovernorCompatibilityBravo.sol
│   ├── extensions
│   │   ├── GovernorPreventLateQuorum.sol!
│   │   ├── GovernorTimelockCompound.sol
│   │   ├── GovernorVotesComp.sol
│   │   ├── GovernorVotesQuorumFraction.sol
│   │   └── GovernorVotes.sol
│   ├── Governor.sol
│   ├── IGovernor.sol
│   ├── TimelockController.sol
│   └── utils
│       ├── IVotes.sol
│       └── Votes.sol
├── token
│   ├── ERC20
│   │   ├── extensions
│   │   │   ├── ERC20Votes.sol
│   │   │   ├── ERC4626.sol
│   └── ERC721
│       └── extensions
│          └─── ERC721Votes.sol
└── access
    ├── AccessControlDefaultAdminRules.sol
    ├── IAccessControlDefaultAdminRules.sol
    └── manager
       ├── AccessManaged.sol
       ├── AccessManagerAdapter.sol
       ├── AccessManager.sol
       └── IAuthority.sol
```

Additionally, the following contracts were audited at the 91df66c4a9dfd0425ff923cbeb3a20155f1355ea commit:

```
contracts
├── proxy
│   └── transparent
│       └── TransparentUpgradeableProxy.sol
├── token
│   └── ERC721
│       └── extensions
│           └── ERC721Wrapper.sol
└── utils
    ├── ShortStrings.sol
    ├── Strings.sol
    └── cryptography
        └── EIP712.sol
```

and the following contracts were audited at the 91df66c4a9dfd0425ff923cbeb3a20155f1355ea commit but only the changes between the versions 4.8 and 4.9 were in-scope:

```
contracts
├── token
│   ├── ERC20
│   │   └── utils
│   │       └── SafeERC20.sol
│   └── ERC721
│       └── extensions
│           └── ERC721URIStorage.sol
└── utils
    ├── Checkpoints.sol
    └── cryptography
        ├── ECDSA.sol
        └── SignatureChecker.sol
```

System Overview

The OpenZeppelin contracts team asked us to review two different parts of their libraries, the governance and access contracts.

Governance contracts

These contracts are designed to manage the decision-making processes of organizations such as DAOs. These contracts allow token holders to propose and vote on changes to the system, such as modifying the protocol, changing the governance rules, or allocating funds.

- *GovernorCompatibilityBravo:* designed to be used as a governance contract that can be used to replace an existing governance contract with minimal disruption to the existing system. It is designed to be compatible with the Bravo protocol, and it allows token holders to propose and vote on changes to the system.
- *GovernorPreventLateQuorum:* designed to prevent proposals from being passed too shortly before the voting period ends, even if they have achieved a quorum.
- *GovernorVotesQuorumFraction:* designed to add flexibility to the quorum by expressing it as a fraction of the total number of tokens, rather than a fixed number, which helps to adjust the quorum requirement if the size of the token holder community changes.
- *GovernorVotes:* basic governance contract that allows token holders (e.g., ERC721 and ERC20) to propose and vote changes in the system. Includes a voting mechanism that requires a minimum quorum and a majority vote to pass proposals. It is meant to be used with the *ERC721Votes* and *ERC20Votes* contracts.
- *GovernorVotesComp:* designed specifically for COMP token holders, this contract works similar to *GovernorVotes*.
- *ERC721Votes* and *ERC20Votes*: designed to allow ERC721 and ERC20 token holders to use the voting power of their tokens in voting systems.
- *TimelockController:* designed to provide time-based control over proposals. Allows proposals to be submitted, but they cannot be executed until a specified amount of time passed.

Most of these contracts implement a delegation functionality, which allows a token holder to delegate their voting power to other accounts to vote on their behalf.

Access control contracts

Contracts that provide a way to manage access control in smart contracts. These contracts allow users to define different roles within a system, assign, revoke and renounce permissions to these roles, and control who has access to certain functions or resources within the system. The OpenZeppelin contracts team is planning to introduce the `manager` contracts, which are meant to be used to better manage permissions in complex systems, and the `AccessControlDefaultAdminRules` contract, which specifies certain rules to manage the holder of the default `admin` role of a system.

Manager contracts

If a contract is meant to have permissioned functions, it should inherit from the `AccessManaged` contract, which provides the `restricted` modifier. Functions decorated with this modifier are permissioned according to an authority, represented by the `IAuthority` interface, which should at least define the `canCall` function. OpenZeppelin contracts provide an authority contract, the `AccessManager` contract, which inherits from `AccessControl` and implements the `canCall` function, and defines a set of at most 255 groups. Users can be added into one or more groups, and each group has access to none, one or more contracts, and to one or more functions of those contracts. By default, all users have access to a `public` group, set in the constructor of the `AccessManager` contract

The `AccessManager` contract defines three different contract modes:

- *Open*: Anyone can call any function of any contract that implements the `restricted` modifier
- *Closed*: No one can call any function of any contract that implements the `restricted` modifier
- *Custom*: Only users that are part of a group can call a certain function of a given target contract.

`AccessControlDefaultAdminRules` contract

The `AccessControl` contract provides basic access control management by defining roles and assigning permissions. However, managing the `DEFAULT_ADMIN_ROLE` is a critical task as it grants special permissions to control other roles, which may potentially have privileged access within the system. To mitigate this risk, OpenZeppelin's access control library provides the `AccessControlDefaultAdminRules` contract.

This contract extends the `AccessControl` contract and adds the ability to specify special rules for managing the `DEFAULT_ADMIN_ROLE`. If a specific role does not have an `admin` role assigned, the holder of the DEFAULT_ADMIN_ROLE can grant or revoke it.

This contract implements several risk mitigations:

- Only one account can hold the `DEFAULT_ADMIN_ROLE` from deployment until it is potentially renounced.
- A 2-step process is enforced to transfer the `DEFAULT_ADMIN_ROLE` to another account.
- A configurable delay is enforced between the two steps, allowing the transfer to be cancelled before it is accepted.
- The delay can be changed by scheduling a call to the `changeDefaultAdminDelay` function.
- It is not possible to use another role to manage the `DEFAULT_ADMIN_ROLE`.

The `AccessControlDefaultAdminRules` contract provides a more secure way to manage the `DEFAULT_ADMIN_ROLE` and prevents potential security risks in the system.

Security Model and Trust Assumptions

The following trust assumptions were part of this audit:

- The contracts audited use several other libraries of the repository. We assume all dependencies that are out of scope work as intended.
- The OpenZeppelin contracts library is meant to be as flexible as possible. To accomplish this, most of the functions defined in the contracts can be overridden by the user, which could potentially introduce vulnerabilities.

Client-Reported Issues

During the course of this audit, the client reported an issue that they found in one of the audited contracts.

The current `AccessControlDefaultAdminRules` implementation inherits from the `AccessControl` behavior, which allows any account to renounce any role even if it has not been granted.

However, if a user renounces the `DEFAULT_ADMIN_ROLE` without holding it, the action resets the `defaultAdmin()` and `owner()` variables.

Consider a scenario where Alice, who is the current default `admin`, initiates a transfer to the zero address by calling the `beginDefaultAdminTransfer` function, so her role can be renounced after a schedule has passed. Once the schedule has passed, Bob, who does not hold the `DEFAULT_ADMIN_ROLE`, calls the `renounceRole` function, which in turn calls the `_revokeRole` function and deletes the `_currentDefaultAdmin`.

Hence, any contract that relies on the `defaultAdmin()` function would incorrectly assume that the `admin` is the zero address, when in reality, the `DEFAULT_ADMIN_ROLE` is still held by Alice, who has not completed the renouncing.

Update: *Resolved in pull request #4177.*

Medium Severity

M-01 Calldata Is Lost Without Signatures

In the `GovernorCompatibilityBravo` contract, the `propose` function allows providing function signatures as a string array and data as a bytes array to be encoded through the `_encodeCalldata` function, which combines those two arrays into proper calldata. The function description of `_encodeCalldata` states that the function signature is optional.

However, if the `signatures` array is of size 0, then the calldata is lost in the loop. This would result in a successful proposal without the intended calldata.

Consider checking that the size of the arrays is equal to ensure the proposal is made as intended.

***Update:** Resolved in pull request #1. The OpenZeppelin contracts team stated:*

> *We will publish a version 4.8.3 with a patch for this issue as well as a security advisory.*

M-02 Potential Selector Collision When Restricting the `receive` Function

The [`AccessManager`] contract keeps track of all restricted functions of the contracts of a system, and who can call them in the [`_allowedGroups`] mapping. To differentiate between functions in a contract, function selectors are used.

However, if a contract has a function with function selector `0x00000000` that is restricted, and also has a restricted receive function, the groups that can access these two will be the same, since `msg.sig` returns `0x00000000` in the receive function.

A user may want to define a function with selector `0x00000000` for gas efficiency reasons, since:

- They are cheaper to look up when someone calls them because they are first in a contract's bytecode.
- They are cheaper to call, since users pay gas for the number of bytes of data sent, and the more zeros, the cheaper.

There are some real case scenarios where this pattern is followed.

Note that the same can happen when a restricted fallback function is defined, and no calldata is sent in the transaction.

Consider including the calldata length to differentiate between a restricted function with 0 bytes selector and restricted receive functions. Otherwise, consider thoroughly documenting this behavior, so users are aware of this edge-case and its possible impact.

Update: *Resolved in pull request #4178. The OpenZeppelin contracts team stated:*

> *Addressed by improving documentation.*

M-03 Potentially Incorre `ERC721Votes`' `_getVotingUnits` Function's Fo

`ERC721Votes` contract] is an extension of the [`ERC721` contract] that supports voting and delegation as implemented by the [`Votes` contract], where, in the base implementation, each individual NFT counts as 1 vote unit. The `Votes.sol` contract defines, among others, two functions:

- The [`_transferVotingUnits` function], which transfers the voting power from one account to another on transfers (if the sender was already delegating)
- The delegate functions, which delegate the voting power of the caller to the specified address

Additionally, the `ERC721Votes` contract defines the [`_getVotingUnits` function]. This function is defined but not implemented in the `Votes.sol` contract, and, in the `ERC721Votes` implementation, it returns the balance of tokens of the account sent as a parameter. Users can override this function to modify the method of accounting voting units.

However, if the `_getVotingUnits` function is overridden to define a different voting unit (which is likely given that it is the only function that can be overridden in the contract and it is feasible that the user would want to use other voting power systems), such as quadratic voting, this change would break the accounting of voting power when delegating it. This happens because when transferring, minting, or burning tokens, the [`_afterTokenTransfer` function] transfers the voting power by granularity of `batchSize`, instead of accounting for the real voting power defined by the potentially overridden `_getVotingUnits` function, as the [`_delegate` function] does.

Example:

Let's say that the `_getVotingUnits` function is declared as a quadratic function:

```
    function _getVotingUnits(address account) internal view virtual override returns
(uint256) {
        return balanceOf(account) * balanceOf(account);
    }
```

Alice is transferred 5 tokens and the `_transferVotingUnits` function is called by the `_afterTokenTransfer` hook, registering a total of 5 token units in the [`_totalCheckpoints`] data structure and in Alice's delegatee's voting power, Bob.

The following scenarios are then possible:

- If Bob is only Alice's delegatee, their voting power would be of 5 units. If Alice redelegates their voting power to Charlie by calling the `delegate` function, this will revert since it uses the [`_getVotingUnits` function to move delegated votes]. Since it would try to move `5 * 5 = 25` voting units, the transaction will revert.
- If Bob has more than 25 voting units because they are the delegatee of multiple accounts, then, if Alice delegates their voting power to Charlie, more voting power than expected will be subtracted from Bob (votes that corresponded to other delegators).

Even though OpenZeppelin contracts' documentation mentions that custom overrides may break some important assumptions and introduce vulnerabilities in otherwise secure code, we found this particular override to be highly likely, very prone to error, and of a very high impact given that almost any change in the formula of the `_getVotingUnits` would break the accountability system of the voting power.

Consider modifying the `_transferVotingUnits` function in the `Votes` contract to account accounts' voting power properly by using the `_getVotingUnits` function.

***Update:** Acknowledged, not resolved. The OpenZeppelin contracts team added a comment to the code informing that overriding the `_getVotingUnits` function would likely result in incorrect vote tracking. The team stated:*

> *We agree that if a developer overrides `ERC721Votes._getVotingUnits`, this might lead to errors. However, we see this as a broader issue with overrides in general. This is something we caveat and warn about in our documentation, and it is also documented in the Security Model section of the audit report: "Most of the functions defined in the contracts can be overridden by the user, which could potentially introduce vulnerabilities".*

Low Severity

L-01 Incomplete Documentation

In the `TimelockController` contract, the `scheduleBatch` function description suggests always emitting a `CallSalt` event. However, this is only the case when the salt is not `0`. Consider modifying the documentation to respect this scenario.

The `GovernorPreventLateQuorum` documentation still states that the time extension is based on a number of blocks and thereby does not follow the clock() change. Consider adapting the documentation to the code changes.

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *Improved documentation as suggested.*

L-02 Non-existent Groups Can Be Granted Access to Permissioned Functions

The `setFunctionAllowedGroup` function in the `AccessManager` contract enables the admin to add and remove a target contract and a list of selectors from a specific group.

However, this function does not check whether the specified group already exists in the `_createdGroups` variable. Consequently, it is possible to add a target and a list of selectors to a non-existent group. This could be confusing and error-prone since future added groups may already have access to a function that they should not have access to.

Consider adding a check that ensures the specified group exists before granting access to a function. This check will help ensure that only the intended groups have access to the target contract and selectors. Otherwise, consider properly documenting this behavior and the rationale behind it.

Update: *Resolved in pull request #4178.*

L-03 Missing Error Messages in `require` Statements

Throughout the codebase, there are `require` statements that lack error messages. For instance:

- The `require` statement on line 93 of `Governor.sol`
- The `require` statement on line 62 of `Votes.sol`
- The `require` statement on line 53 of `ERC20Votes.sol`

Consider including specific, informative error messages in `require` statements to improve overall code clarity and facilitate troubleshooting whenever a requirement is not satisfied.

***Update:** Resolved in pull request #4176.*

Notes & Additional Information

N-01 Misleading Comments

There are several places where docstrings could be improved:

- The `_upperBinaryLookup` function is called by the `upperLookup` and `upperLookupRecent` functions. However, docstrings of these functions state the opposite things. `upperLookup` and `upperLookupRecent` state that the return key is lower or equal to the search key but `_upperBinaryLookup` states that the return key is greater than the search key.
- The `upperLookup` and `upperLookupRecent` functions do not specify how `Trace224` structs of length 0 are handled.
- There are no docstring on how `ShortStrings` is stored in memory.

Consider modifying the docstrings to reflect these items.

***Update:** Resolved in pull requests #4218 and #4224.*

N-05 State Variable Visibility Not Explicitly Declared

Within `AccessManager.sol`, the state variable `_createdGroups` lacks an explicitly declared visibility.

For clarity, consider always explicitly declaring the visibility of variables, even when the default visibility matches the intended visibility.

Update: *Resolved in pull request #4178.*

N-06 Function Visibility Can Be Public

The `Governor` contract provides certain functions that can be called from the contract itself, its children, and the outside world to query some of the proposal's variables, such as `proposalDeadline` and `proposalSnapshot` (to get the `voteStart` and `voteEnd` variables). It also defines the `_proposalProposer` function, which is internal, and therefore not accessible by other contracts or the outside world. For consistency, and to let users get the proposer's address of a particular proposal, consider changing the visibility to `public`, and renaming the function to `proposalProposer`, or consider creating a new function named `proposalProposer` that returns what the `_proposalProposer` function returns.

Update: *Resolved in pull request #4176.*

N-07 Inconsistent `SafeCast` Usage

The usage of the `SafeCast` library varies between contracts. Some contracts declare `using SafeCast for uint256` at the top (e.g., `Governor.sol`) while others do `SafeCast.toUint32(number)` (e.g., `Votes.sol`). Consider applying a consistent style throughout the codebase to improve its readability.

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *Changed to make it consistent (removed "using for" syntax).*

N-08 Interface and Implementation Mismatch

The diff under review introduced a change to the `IGovernanceCompatibilityBravo` interface. Since the `cancel` function is now natively supported by the underlying `Governor` contract, the `cancel` function also moved to the `IGovernor` interface.

However, the function signature varies from the `IGovernor` interface to the `IGovernanceCompatibilityBravo` interface, as seen below:

`IGovernor` interface

```
function cancel(
    address[] memory targets,
    uint256[] memory values,
    bytes[] memory calldatas,
    bytes32 descriptionHash
) public virtual returns (uint256 proposalId);
```

`IGovernanceCompatibilityBravo` interface

```
function cancel(uint256 proposalId) public virtual;
```

This currently leaves one of the implemented `cancel` functions without its interface counterpart. Consider adding it back to the interface for completeness.

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *The function had been removed from the interface previously but it shouldn't have. Added it back.*

N-09 "Last updated" and "Available since" Comments Unchanged or Missing

The files of the contracts library all have a comment on top of the file indicating in which release it was last changed. For the diffs in scope, this is yet unchanged. Consider updating these comments before the actual release, and consider adding the version from which new contracts are available.

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

> *The "last updated" heading is automatically changed by our release scripts. Unclear which of the "available since" are missing.*

N-10 Latest Governance Interface Is Unsupported

The [`Governor` contract] implements the ERC-165 standard and thereby the `supportsInterface` function. Within that function, different versions of the `Governance` interface are supported for backwards compatibility. However, the latest version is not respected in the set of supported interfaces.

Consider adding support for the current interface for the sake of completeness.

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *An updated interface id had been deliberately omitted because we are concerned about the sustainability of that approach as the contract evolves. We realize now that ERC-165 is not a good fit, but we also realize that as-is the contract is missing a method for the detection of the new features.*
>
> *We've added an interface id for the 2 new functions in this release.*

N-11 Missing Event Parameters

Some events should be modified to emit information that could be helpful for users:

- The `ProposalExecuted` and `ProposalCanceled` events in the `Governor` contract could emit the block/timestamp (i.e., clock()) in which the proposal was executed/ canceled.
- The `AccessModeUpdated` event could emit the old contract mode.
- The `AuthorityUpdated` event in the `AccessManaged` contract could emit the old authority address.

Update: *Resolved in pull request #4178. The OpenZeppelin contracts team stated:*

> *`ProposalExecuted` and `ProposalCanceled` are locked by backwards compatibility.*
>
> *`AuthorityUpdated` in fact already includes the old authority address as used in `AccessManaged`. Note that we used the same event as supported by Solmate.*
>
> *In `AccessModeUpdated` we have now added a parameter for the old mode.*

N-12 Naming Issues Hinder Code Understanding and Readability

To favor explicitness and readability, several parts of the contracts may benefit from better naming. Our suggestions are:

- Renaming the `Checkpoint.fromBlock` parameter to `Checkpoint.timepoint` in the `ERC20Votes` contract, to adhere to the `clock()` changes throughout the diff.
- The `status` variable in the `execute` function should be named `proposalState`. The concept of `status` is never introduced until that function, and the word "state" and "status" have different meanings.
- The `_push` function in the `Votes.sol` contract should be renamed to `_upsert` or `_pushOrUpdate`. If the last time an item was pushed to the `Checkpoints` array is equal to the current time (`clock()`), the function will not actually push a new item but instead, update the latest one.
- The `AccessManaged` contract could be renamed to `AccessManageable`, to follow the same naming convention used throughout the library (`Initializable`, `Enumerable`, `Ownable`, `Pausable`, `Mintable`, `Burnable`, etc).

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *Most of the suggestions can't be applied for v4.9 due to backwards compatibility constraints, but will be considered for v5.0.*
>
> *The name `AccessManageable` does not sound good to us, so we're leaving this for consideration later.*
>
> *`status` was renamed to `currentState`.*

N-13 Impossible to Get an Account's Latest Votes Through the `GovernorVotes` Contract

The `_getVotes` function in the `GovernorVotes` contract overrides the same function from the `Governor` contract so that the `getVotes` function from the Governor contract returns the votes of a given account at a given point in time.

However, the `_getVotes` function in the `GovernorVotes` contract calls the `getPastVotes` function from the token contract (which, in the context of this library, could be either of the `ERC721Votes` or `ERC20Votes` contracts), making it impossible to get the latest vote of a given account through the Governor contract.

Since this behavior is intentional to avoid double-voting issues, consider adding an external `getLatestVotes` function in the Governor contract that users can call to get the latest state of an account.

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

> *We will not add this getter to avoid adding more surface to the Governor contracts. The information can be queried on the token if needed.*

N-14 Overlapping `TimelockController` Statuses

In the `TimelockController` contract, the status of operations can be queried (e.g., with `isOperationPending` or `isOperationReady`). The checks that lead to either status partly overlap, such that a ready operation is also always pending as well. While this makes sense, this behavior could confuse developers, who may expect the statuses to be exclusive.

Consider explicitly documenting this behavior in the functions' NatSpec.

Update: *Resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *We have added documentation to clarify.*

N-15 Require Statements With Multiple Conditions

Within the codebase, there are multiple `require` statements that require multiple conditions to be satisfied. For instance:

- The `require` statement on line 110 of `AccessControlDefaultAdminRules.sol`
- The `require` statement on line 248 of `AccessControlDefaultAdminRules.sol`
- The `require` statement on line 420 of `Governor.sol`

To simplify the codebase and raise the most helpful error messages for failing `require` statements, consider having a single require statement per condition.

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

> *We will not apply this suggestion because we are more concerned about code size bloat from additional revert reasons than about the specificity of the revert reasons.*

N-16 Typographical Errors

To improve readability, consider correcting the following typographical errors:

- on line 11 of `AccessManaged.sol`, "allows certain callers access to certain functions" should be "allows certain callers to access certain functions"
- on line 77 of `AccessManager.sol`, "succintly" should be "succinctly"
- on line 20 of `Governor.sol`, "several function" should be "several functions"
- on line 76 of `GovernorCompatibilityBravo.sol`, "their" should be "there"
- on line 167 of `IGovernor.sol`, "vote ends" should be "vote end"
- on line 28 and line 34 of `IVotes.sol` as well as line 75 and line 88 of `Votes.sol`, "the value the end" should be "the value at the end"

Update: *Resolved in pull request #4178 and pull request #4176.*

N-17 Unclear Proposal State

When a proposal is not canceled, executed or active, it may succeed or be defeated. For a proposal to succeed, two conditions must be met: the voting must succeed and the quorum must be reached. These two conditions are defined by the developer because the default functions `_quorumReached` and `_voteSucceeded` are not implemented.

Since the criteria for a proposal to succeed depends on two user-defined functions, it may be unclear why a proposal was defeated. To provide more specific information about why a proposal was defeated, consider modularizing the proposal state `Defeated` into two categories: `DefeatedQuorum` and `DefeatedVoting`.

Update: *Acknowledged, will resolve. The OpenZeppelin contracts team stated:*

> *We cannot change the state enum due to backwards compatibility, but will consider a getter such as `defeatReason` for future versions.*

N-18 Lack of Indexed Parameter in Event

The `ProposalCreated` event of the `IGovernor` interface is fired whenever a new proposal is made. Among its parameters are the proposer's address and the proposal ID. Consider making these parameters `indexed` to leverage the filtering of events.

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

> *Cannot change due to backwards compatibility, in particular with `GovernorBravo`.*

N-19 Unused Imports

Consider removing the following unused imports to improve the overall clarity and readability of the codebase.

- In `TimelockController.sol`, the import `Address` is unused and could be removed.
- In `AccessManager.sol`, the import `AccessControl` is unused and could be removed.

Update: *Resolved in pull request #4176 and pull request #4178.*

N-20 Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function body for the purpose of being returned as the function's output. They are an alternative to explicit in-line `return` statements.

Within the codebase, there are multiple instances of unused named return variables:

- The `newAdmin` return variable in the `pendingDefaultAdmin` function in `AccessControlDefaultAdminRules.sol`.
- The `schedule` return variable in the `pendingDefaultAdmin` function in `AccessControlDefaultAdminRules.sol`.
- The `newDelay` return variable in the `pendingDefaultAdminDelay` function in `AccessControlDefaultAdminRules.sol`.
- The `registered` return variable in the `isOperation` function in `TimelockController.sol`.
- The `pending` return variable in the `isOperationPending` function in `TimelockController.sol`.
- The `ready` return variable in the `isOperationReady` function in `TimelockController.sol`.
- The `done` return variable in the `isOperationDone` function in `TimelockController.sol`.

- The `timestamp` return variable in the `getTimestamp` function in `TimelockController.sol`.
- The `duration` return variable in the `getMinDelay` function in `TimelockController.sol`.
- The `hash` return variable in the `hashOperation` function in `TimelockController.sol`.
- The `hash` return variable in the `hashOperationBatch` function in `TimelockController.sol`.
- The `targets` return variable in the `_getProposalParameters` function in `GovernorCompatibilityBravo.sol`.
- The `values` return variable in the `_getProposalParameters` function in `GovernorCompatibilityBravo.sol`.
- The `calldatas` return variable in the `_getProposalParameters` function in `GovernorCompatibilityBravo.sol`.
- The `descriptionHash` return variable in the `_getProposalParameters` function in `GovernorCompatibilityBravo.sol`.
- The `targets` return variable in the `getActions` function in `GovernorCompatibilityBravo.sol`.
- The `values` return variable in the `getActions` function in `GovernorCompatibilityBravo.sol`.
- The `signatures` return variable in the `getActions` function in `GovernorCompatibilityBravo.sol`.
- The `calldatas` return variable in the `getActions` function in `GovernorCompatibilityBravo.sol`.
- The `timepoint` return variable in the `clock` function in `GovernorVotes.sol`.
- The `clockmode` return variable in the `CLOCK_MODE` function in `GovernorVotes.sol`.
- The `timepoint` return variable in the `clock` function in `GovernorVotesComp.sol`.
- The `clockmode` return variable in the `CLOCK_MODE` function in `GovernorVotesComp.sol`.

Consider either using or removing any unused named return variables.

Update: *Partially resolved in pull request #4176. The OpenZeppelin contracts team stated:*

> *Removed some but not all. In particular, we want to have named return variables when there are multiple return values, as a form of documentation.*

Conclusions

Three medium-severity issues were found. Several changes were proposed to improve the code's overall quality and reduce the attack surface.