How To Use AccessControl.sol

Three example configurations for the new OpenZeppelin access control contract.

Alberto Cuesta Cañada

Follow

Coinmonks

--

4

Listen

Share

Restricting access to authorized users is really important in smart contracts. With $1 Billion locked in DeFi applications, you would really hope everyone knows what they are doing.

Recently, I collaborated with OpenZeppelin to refactor the access control in their widely used openzeppelin-contracts repository. If you have ever coded a smart contract, most likely you have inherited from their code.

In this article, I'm going to give you a walkthrough of how to use the revamped AccessControl.sol to restrict access to your smart contracts. I'll do this through three examples.

But first, let me tell you the story of how we got here. Please feel free to skip the next section if you are not interested in how this contract came to be and only want to learn how to use it.

How we got here

Getting here for me has been a process that took about a year. I first had the idea for a hierarchical role-based control contract in May 2019. I published the code in Medium, along with some references to what OpenZeppelin was doing at the time.

I reused that code for AllianceBlock and Insig, becoming convinced that the implementation offered something both unique and powerful. From the lessons learned I wrote a second article showing when to use Ownable.sol, Whitelist.sol and RBAC.sol.

To write that article I needed to research deeply into what OpenZeppelin had been doing in terms of access control. You don't challenge the most-forked solidity repository without doing your homework.

The guys at OpenZeppelin team are really welcoming, and shortly after publishing that second article they invited me to participate in the refactor of their access control contracts. Out of that collaboration we created an access control contract that is much more flexible than their previous approach, and much more robust than my previous approach.

AccessControl.sol is available from the openzeppelin-contracts repository and I don't keep a different codebase anymore.

As an interesting aside, and as further proof of how important is access control, a version of Ownable.sol existed back in 2016. It was Manuel Araoz's 9th commit towards what would become openzeppelin-contracts.

In solidity years that is like digging dinosaur bones, and proves that access control is one of the very first things you should think about when coding smart contracts. Especially if you intend to code robust ones.

Story time over, let's dive into the code.

How does AccessControl.sol work

A role in AccessControl.sol is a struct that contains a set of addresses, representing the accounts bearing that role. All roles are stored in a mapping indexed by a bytes32 identifier, unique for each role.

Each role also contains the bytes32 identifier of another role, which we call its admin role.

There are internal functions to grant and revoke roles, and to set a role as the admin of another. These internal functions have no restrictions and you can use them if you are extending AccessControl.sol, as we will do later in this article.

There are also external functions that restrict granting and revoking roles, as well as setting roles as admins of others. These functions can only be called externally by accounts that have been granted the admin role of the role being modified.

As an example, imagine that we have two roles, DEFAULT_ADMIN_ROLE and USER_ROLE. DEFAULT_ADMIN_ROLE is the admin role for USER_ROLE.

Only accounts that have been granted the DEFAULT_ADMIN_ROLE can grant or revoke USER_ROLE for an account.

Likewise, only accounts that have been granted the DEFAULT_ADMIN_ROLE can redefine the admin role for USER_ROLE.

A bit of magic worked here by nventuro was to define DEFAULT_ADMIN_ROLE as the uninitialized bytes32 variable. That means that all roles, by default, have DEFAULT_ADMIN_ROLE as the admin role.

That's it, let me show you in more detail with examples.

Community

This contract replicates the functionality that existed in OpenZeppelinRoles.sol before.

- There is a single role (DEFAULT_ADMIN_ROLE)
- Anyone that has that role can grant it to others.
- In Community.sol roles can't be revoked from a different account, but accounts can renounce to a role they hold.
- The address passed on to the constructor is the first account being granted the role.

Administered

This smart contract implements a traditional setup with administrators and users.

- DEFAULT_ADMIN_ROLE is the admin role of USER (by default).
- The address passed on to the constructor is the initial administrator.
- Admins can add other admins.
- Admins can grant and revoke user permissions to any accounts.
- The only way for an admin to lose its admin role is to renounce from it.

Hierarchy

This is the structure implemented in AccessControl.sol, with a minimal modification to build a hierarchy of roles safely.

- The address passed on to the constructor has the DEFAULT_ADMIN_ROLE, and is by default admin of all roles. I'll call that address root.
- Root can grant the DEFAULT_ADMIN_ROLE to any account.
- Any root account can revoke the DEFAULT_ADMIN_ROLE role from any other account, so better not to grant it in the first place unless you know what you are doing.
- All roles exist from the beginning, as keys in a mapping. Any role can be granted or revoked by an account in the DEFAULT_ADMIN_ROLE.
- In AccessControl.sol each role has an admin role (DEFAULT_ADMIN_ROLE by default). In Hierarchy.sol we allow for any account with that admin role can change the relationship and choose a new admin role. This can be used to build a hierarchy of roles, for example: DEFAULT_ADMIN_ROLE -> USER_ADMIN -> USER.

It's useful to have AccessControl.sol relevant code handy to understand what's going on.

Conclusion

I'm really happy and really proud of having got to this point.

Really proud because contributing code and ideas toopenzeppelin-contracts is the strongest validation possible as a smart contract developer.

Really happy because finally the access control contracts that I've been using have now gone through a thorough audit and have been refined by masters in their field. I can now use them with complete confidence that they are right.

And really proud and happy to contribute to the community. Giving back is the best present.

Now it is up to you to continue. As usual, if you have an interesting project and need some advice on using these features, please get in contact. BUIDL!

Get Best Software Deals Directly In Your Inbox