How To Review a Governance Action in Ehereum

Alberto Cuesta Cañada

Follow

--

Listen

Share

Good governance needs a clear understanding of consequences.

When I worked for financial institutions, strict change management processes protected the technology from unintended harm during changes. Senior staff was responsible for implementing changes, but their busy schedules and the complexity of the changes often made this challenging. To avoid rejection, one had to carefully describe the intended change and follow the rules and check the appropriate boxes.

Blockchain technology is reinventing finance, including change management processes. We have been successful in making senior staff responsible for changes through the use of multisigs and DAOs, but we have not been as successful in ensuring that these signers understand what they are signing off on.

As a result, it is necessary to implore them to sign. Despite the real risk of a governance attack, it has become somewhat of a joke to simply sign off on things without understanding their implications.

Early on at Yield, I focused on the problem of multisig members not understanding what they were signing off on. This was unacceptable to me, and I wanted to prevent developers like myself from being able to rug the protocol before higher valuations might influence judgments.

If your multisig members are unsure of what they are signing, there are ways to make the process easier for them. I'll show you how.

How Governors See Change

Some blockchain applications, such as Uniswap, are permissionless and require no further action after deployment. However, most other applications have adopted change-by-governance patterns, where changes can be made as long as they are signed off by a multisig or a DAO.

Some changes are simple, such as transferring tokens or adjusting a parameter, while others are more complex and involve deploying contracts and orchestrating them.

Obtaining signatures for any change requires a significant amount of effort, so it is common to ask for a single signature for all elements included in a change. At Yield, the largest change I've seen included more than three hundred individual calls.

For simpler changes, multisig members may be willing to review the call data and may even have the ability to understand it. However, for more complex changes, this is not possible.

Governors approving a change should know which functions are being called, what parameters are being used, and what effects these will have. They should be able to find this information on their own, without help from others.

It is possible to achieve this, and it is easier than it may seem. However, first, we must understand how governance changes are executed by smart contracts.

Governance Technology

At Yield, we built our own Timelock, taking inspiration from Compound but making a crucial change. While the original Timelock can store and delay a single call, our Timelock can store a batch of calls with any number of calls included.

Other protocols may use different contracts, but the basic process remains the same: a batch of function calls is recorded, approved, and then executed.

In our Timelock, a call is a target and call data representing a call on the target:

A proposal, or change, is simply an array of calls.

The propose function is permissioned and, in our implementation, only allowed to our developers.

There is also an approve function that can only be called by the multisig. The approval time determines the earliest possible execution time.

Only developers are allowed to execute a change, and they must supply the same batch that was used in the propose function. If the batch is different or the change identifier is incorrect, the change will fail to find the approval and will not be executed.

In any situation where a change is being considered, there will be someone who decides that the change should be made, and there will be developers who turn that idea into code. This code, in the form of a batch of function calls, is then fed to the Timelock using `propose`.

The next step is for responsible senior staff, a multisig, or a DAO to review the change and decide whether to approve it, calling `approve`.

This step is the one that is often difficult because understanding the proposed change from the Timelock is difficult. We can make this process easier for them by using existing development tools to provide them with detailed information about the nature of the change.

Reviewing Change in Detail

Once a change has been proposed in the Timelock, it becomes immutable thanks to its unique identifier. If a governor approves this identifier, only that exact batch of calls can be executed.

With this in mind, we can:

1. `propose` on the real network.
2. Fork that network.
3. Impersonate the governor and `approve` the proposal in the fork.
4. `execute` in the fork

The change being executed is the same as the one that was proposed because we forked the network after the proposal. Not only that, but anyone can analyze the execution with a transaction decoder.

At Yield, we use Tenderly, but there are other options available. I made publicone of our proposal executions in a forkfor educational purposes.

As a multisig member at Yield, I review the execution of every proposal in a fork. I compare the change identifier for approval in the multisig against the change identifier in the fork, and then I know with total certainty that what is in the multisig is what would be executed.

All changes must include a text-based description of their purpose and the parameters being used. I use this to understand what the change does in a broad sense, and then use the transaction decoder to review the detail.

I start by comparing the description against the contracts involved in the execution to ensure that they match. Conducting due diligence in this process includes verifying all new contracts and ensuring that their verified code is what is in the repository.

Knowing the contracts involved, the events of the transaction provide all the necessary information for reviewing the rest of the execution.

I review the permissions granted and revoked, any changes to the oracles, and that the numerical parameters match what was discussed and agreed upon.

I document my review on Notionand make it publicly available for each change. Other governors can review my work or conduct their own due diligence in any other way they choose. In any case, I believe that the risk of being caught if I misrepresent a change is now sufficiently high.

Gnosis Safe as a Simpler Alternative

Gnosis Safe now offers a batch builder and automatically creates a simulation on Tenderly for all executions. This is the same functionality I described, with a friendlier interface.

If your governance proposals are simple enough to be created manually in a web interface, Gnosis Safe would be a simpler option. You can share the simulation report from Gnosis Safe and achieve the same result.

Conclusion

Governance of blockchain applications is difficult, and as a governor, you risk signing off on changes that could lead to the failure of the application, either by mistake or by design. Recent events have only emphasized the need to protect our users from harm.

By using developer tools and following simple processes atYield, we have been able to provide governors with clear information about what they are signing off on. We have successfully executed over one hundred separate changes, totaling thousands of contract calls, have reduced the number of governance errors, and are able to catch more mistakes earlier than before.

Nothing we do is magic; it is simply common sense. Please make sure that you understand what you sign, and ask for the necessary tools if you must. We owe it to our users.