Rules

Rules are programmable restrictions that you can apply to any action in the TokenPolicy . They are the tool of compliance, regulation, and enforcement of certain business logic in the closed-loop system.

A rule is represented as a witness - a type with a drop ability. You can either encode it in your application logic, or include it as part of a separate module for a more modular approach.

After you add a rule to an action in the TokenPolicy , the action requires a stamp of the rule to pass confirmation.

See the Approving actions section for more details on how to approve an action.

You can publish rules as separate reusable modules. This enables you to create a library of rules that you can use in different token policies, maximizing code reuse and minimizing the risk of errors.

A rule module is a regular module with a verify -like function that typically takes a TokenPolicy , ActionRequest , and a TxContext as arguments. The function is responsible for verifying the action and stamping the ActionRequest with the rule type.

Some rules, such as denylist or allowlist require configuration. For example, a denylist rule might require a list of addresses that are not allowed to perform certain actions. A rule module can define a configuration structure and provide functions to add, modify, retrieve, and remove the configuration.

A single rule has a single configuration, even when assigned to multiple actions. If there's a need to have configuration per action, a rule module needs to define a storage structure that can hold and manage multiple configurations.

The configuration system comes with a set of guarantees to protect token owners from malicious actions (or upgrades) from rule module developers:

The only attack vector available to the rule creator is upgrading the module and creating a function to bypass the restriction. Make sure to use rules provided by a trusted developer.

The sui::token module defines the configuration API and has the following set of functions.

A rule must approve new configurations (the rule witness) and the TokenPolicy owner. The type of the configuration can be any as long as it has the store ability.

Rules can read the configuration stored in the TokenPolicy .

A rule must approve configuration modifications (the rule witness) as well as the TokenPolicy owner.

A good practice for rules is to provide a method to remove the configuration, as a rule can use a custom type for it. However, a token owner can always call the remove_rule_config function to remove the configuration.

Because the configuration has store , the token owner can wrap and transfer or store the configuration somewhere else. If the Config type has drop , the value can be ignored.

Rule structure

A rule is represented as a witness - a type with a drop ability. You can either encode it in your application logic, or include it as part of a separate module for a more modular approach.

After you add a rule to an action in the TokenPolicy , the action requires a stamp of the rule to pass confirmation.

See the Approving actions section for more details on how to approve an action.

You can publish rules as separate reusable modules. This enables you to create a library of rules that you can use in different token policies, maximizing code reuse and minimizing the risk of errors.

A rule module is a regular module with a verify -like function that typically takes a TokenPolicy , ActionRequest , and a TxContext as arguments. The function is responsible for verifying the action and stamping the ActionRequest with the rule type.

Some rules, such as denylist or allowlist require configuration. For example, a denylist rule might require a list of addresses that are not allowed to perform certain actions. A rule module can define a configuration structure and provide functions to add, modify, retrieve, and remove the configuration.

A single rule has a single configuration, even when assigned to multiple actions. If there's a need to have configuration per action, a rule module needs to define a storage structure that can hold and manage multiple configurations.

The configuration system comes with a set of guarantees to protect token owners from malicious actions (or upgrades) from rule module developers:

The only attack vector available to the rule creator is upgrading the module and creating a function to bypass the restriction. Make sure to use rules provided by a trusted developer.

The sui::token module defines the configuration API and has the following set of functions.

A rule must approve new configurations (the rule witness) and the TokenPolicy owner. The type of the configuration can be any as long as it has the store ability.

Rules can read the configuration stored in the TokenPolicy .

A rule must approve configuration modifications (the rule witness) as well as the TokenPolicy owner.

A good practice for rules is to provide a method to remove the configuration, as a rule can use a custom type for it. However, a token owner can always call the remove_rule_config function to remove the configuration.

Because the configuration has store , the token owner can wrap and transfer or store the configuration somewhere else. If the Config type has drop , the value can be ignored.

Modular rules

You can publish rules as separate reusable modules. This enables you to create a library of rules that you can use in different token policies, maximizing code reuse and minimizing the risk of errors.

A rule module is a regular module with a verify -like function that typically takes a TokenPolicy , ActionRequest , and a TxContext as arguments. The function is responsible for verifying the action and stamping the ActionRequest with the rule type.

Some rules, such as denylist or allowlist require configuration. For example, a denylist rule might require a list of addresses that are not allowed to perform certain actions. A rule module can define a configuration structure and provide functions to add, modify, retrieve, and remove the configuration.

A single rule has a single configuration, even when assigned to multiple actions. If there's a need to have configuration per action, a rule module needs to define a storage structure that can hold and manage multiple configurations.

The configuration system comes with a set of guarantees to protect token owners from malicious actions (or upgrades) from rule module developers:

The only attack vector available to the rule creator is upgrading the module and creating a function to bypass the restriction. Make sure to use rules provided by a trusted developer.

The sui::token module defines the configuration API and has the following set of functions.

A rule must approve new configurations (the rule witness) and the TokenPolicy owner. The type of the configuration can be any as long as it has the store ability.

Rules can read the configuration stored in the TokenPolicy .

A rule must approve configuration modifications (the rule witness) as well as the TokenPolicy owner.

A good practice for rules is to provide a method to remove the configuration, as a rule can use a custom type for it. However, a token owner can always call the remove_rule_config function to remove the configuration.

Because the configuration has store , the token owner can wrap and transfer or store the configuration somewhere else. If the Config type has drop , the value can be ignored.

Rule configuration

Some rules, such as denylist or allowlist require configuration. For example, a denylist rule might require a list of addresses that are not allowed to perform certain actions. A rule module can define a configuration structure and provide functions to add, modify, retrieve, and remove the configuration.

A single rule has a single configuration, even when assigned to multiple actions. If there's a need to have configuration per action, a rule module needs to define a storage structure that can hold and manage multiple configurations.

The configuration system comes with a set of guarantees to protect token owners from malicious actions (or upgrades) from rule module developers:

The only attack vector available to the rule creator is upgrading the module and creating a function to bypass the restriction. Make sure to use rules provided by a trusted developer.

The sui::token module defines the configuration API and has the following set of functions.

A rule must approve new configurations (the rule witness) and the TokenPolicy owner. The type of the configuration can be any as long as it has the store ability.

Rules can read the configuration stored in the TokenPolicy .

A rule must approve configuration modifications (the rule witness) as well as the TokenPolicy owner.

A good practice for rules is to provide a method to remove the configuration, as a rule can use a custom type for it. However, a token owner can always call the remove_rule_config function to remove the configuration.

Because the configuration has store , the token owner can wrap and transfer or store the configuration somewhere else. If the Config type has drop , the value can be ignored.

Configuration API

The sui::token module defines the configuration API and has the following set of functions.

A rule must approve new configurations (the rule witness) and the TokenPolicy owner. The type of the configuration can be any as long as it has the store ability.

Rules can read the configuration stored in the TokenPolicy .

A rule must approve configuration modifications (the rule witness) as well as the TokenPolicy owner.

A good practice for rules is to provide a method to remove the configuration, as a rule can use a custom type for it. However, a token owner can always call the remove_rule_config function to remove the configuration.

Because the configuration has store , the token owner can wrap and transfer or store the configuration somewhere else. If the Config type has drop , the value can be ignored.

Cheatsheet: Rule configuration API