



Overview

Casbin is a powerful and efficient open-source access control library that supports various [access control models](#) for enforcing authorization across the board.

Enforcing a set of rules is as simple as listing subjects, objects, and the desired allowed action (or any other format as per your needs) in a *policy* file. This is synonymous across all flows in which Casbin is used. The developer/administrator has complete control over the layout, execution, and conditions for authorization, which are set via the *model* file. Casbin provides an *Enforcer* for validating an incoming request based on the policy and model files given to the Enforcer.

Languages Supported by Casbin

Casbin provides support for various programming languages, ready to be integrated within any project and workflow:

	Go		Java
Casbin		jCasbin	
Production-ready		Production-ready	



PyCasbin
Production-ready

Casbin.NET
Production-ready

Feature Set for Different Languages

We are always working our best to make Casbin have the same set of features for all languages. However, the reality is not that beautiful.

Feature	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
Enforcement	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ABAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scaling ABAC (eval())							✗	✓	✓	✓	✓	✓
Adapter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Management API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Batch API	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗

Feature	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
Filtered Adapter	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
Watcher	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Role Manager	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Multi-Threading	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗
'in' of matcher	✓	✓	✓	✓	✓	✗	✓	✗	✗	✓	✓	✓

Note - ✓ for Watcher or Role Manager only means having the interface in the core library. It is not indicative of whether there is a watcher or role manager implementation available.

What is Casbin?

Casbin is an authorization library that can be used in flows where we want a certain `object` or entity to be accessed by a specific user or `subject`. The type of access, i.e. `action`, can be `read`, `write`, `delete`, or any other action as set by the developer. This is how Casbin is most widely used, and it's called the "standard" or classic `{ subject, object, action }` flow.

Casbin is capable of handling many complex authorization scenarios other than the standard flow. There can be the addition of [roles \(RBAC\)](#), [attributes \(ABAC\)](#), etc.

What Casbin Does

1. Enforce the policy in the classic `{ subject, object, action }` form or a customized form as you defined. Both allow and deny authorizations are supported.
2. Handle the storage of the access control model and its policy.
3. Manage the role-user mappings and role-role mappings (aka role hierarchy in RBAC).
4. Support built-in superusers like `root` or `administrator`. A superuser can do anything without explicit permissions.
5. Provide multiple built-in operators to support rule matching. For example, `keyMatch` can map a resource key `/foo/bar` to the pattern `/foo*`.

What Casbin Does NOT Do

1. Authentication (aka verifying `username` and `password` when a user logs in)
2. Manage the list of users or roles.

It's more convenient for projects to manage their lists of users, roles, or passwords. Users usually have their passwords, and Casbin is not designed as a password container. However, Casbin stores the user-role mapping for the RBAC scenario.

Get Started

Installation

Go

Java

Node.js

PHP

Python

.NET

C++

Rust

Delphi Lua

```
go get github.com/casbin/casbin/v2
```

For Maven:

```
<!-- https://mvnrepository.com/artifact/org.casbin/jcasbin -->
<dependency>
    <groupId>org.casbin</groupId>
    <artifactId>jcasbin</artifactId>
    <version>1.x.y</version>
</dependency>
```

```
# NPM
npm install casbin --save
```

```
# Yarn
yarn add casbin
```

Require this package in the `composer.json` of your project to download the package:

```
composer require casbin/casbin
```

```
pip install casbin

dotnet add package Casbin.NET

# Download source
git clone https://github.com/casbin/casbin-cpp.git

# Generate project files
cd casbin-cpp && mkdir build && cd build && cmake ..
-DCMAKE_BUILD_TYPE=Release

# Build and install casbin
cmake --build . --config Release --target casbin install -j 10

cargo install cargo-edit
cargo add casbin

// If you use async-std as async executor
cargo add async-std

// If you use tokio as async executor, make sure you activate
its `macros` feature
cargo add tokio
```

Casbin4D comes in a package (currently for Delphi 10.3 Rio) and you can install it in the IDE. However, there are no visual components which means that you can use the units independently of packages. Just import the units in your project (assuming you do not mind the number of them).

```
luarocks install casbin
```

If you receive an error message: "Your user does not have write permissions in /usr/local/lib/luarocks.rocks", you may want to run the command as a privileged

user or use your local tree with `--local`. To fix the error, you can add `--local` behind your command like this:

```
luarocks install casbin --local
```

New a Casbin enforcer

Casbin uses configuration files to define the access control model.

There are two configuration files: `model.conf` and `policy.csv`. `model.conf` stores the access model, while `policy.csv` stores the specific user permission configuration. The usage of Casbin is very straightforward. We only need to create one main structure: `enforcer`. When constructing this structure, `model.conf` and `policy.csv` will be loaded.

In other words, to create a Casbin enforcer, you need to provide a [Model](#) and an [Adapter](#).

Casbin provides a [FileAdapter](#) that you can use. See [Adapter](#) for more information.

- Example of using the Model file and the default [FileAdapter](#):

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [C++](#) [Delphi](#)

[Rust](#) [Lua](#)

```
import "github.com/casbin/casbin/v2"
```

```
import org.casbin.jcasbin.main.Enforcer;

Enforcer e = new Enforcer("path/to/model.conf", "path/to/
policy.csv");

import { newEnforcer } from 'casbin';

const e = await newEnforcer('path/to/model.conf', 'path/to/
policy.csv');

require_once './vendor/autoload.php';

use Casbin\Enforcer;

$e = new Enforcer("path/to/model.conf", "path/to/policy.csv");

import casbin

e = casbin.Enforcer("path/to/model.conf", "path/to/policy.csv")

using NetCasbin;

var e = new Enforcer("path/to/model.conf", "path/to/
policy.csv");

#include <iostream>
#include <casbin/casbin.h>

int main() {
    // Create an Enforcer
    casbin::Enforcer e("path/to/model.conf", "path/to/
policy.csv");
```

```

var
    casbin: ICasbin;
begin
    casbin := TCasbin.Create('path/to/model.conf', 'path/to/
policy.csv');
    ...
end

use casbin::prelude::*;

// If you use async_td as async executor
#[cfg(feature = "runtime-async-std")]
#[async_std::main]
async fn main() -> Result<()> {
    let mut e = Enforcer::new("path/to/model.conf", "path/to/
policy.csv").await?;
    Ok(())
}

// If you use tokio as async executor
#[cfg(feature = "runtime-tokio")]
#[tokio::main]
async fn main() -> Result<()> {
    let mut e = Enforcer::new("path/to/model.conf", "path/to/
policy.csv").await?;
    Ok(())
}

local Enforcer = require("casbin")
local e = Enforcer:new("path/to/model.conf", "path/to/
policy.csv") -- The Casbin Enforcer

```

- Use the Model text with other Adapter:

[Go](#) [Python](#)

```
import (
    "log"

    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin/v2/model"
    xormadapter "github.com/casbin/xorm-adapter/v2"
    _ "github.com/go-sql-driver/mysql"
)

// Initialize a Xorm adapter with MySQL database.
a, err := xormadapter.NewAdapter("mysql",
    "mysql_username:mysql_password@tcp(127.0.0.1:3306)/")
if err != nil {
    log.Fatalf("error: adapter: %s", err)
}

m, err := model.NewModelFromString(`  

[request_definition]  

r = sub, obj, act  
  

[policy_definition]  

p = sub, obj, act  
  

[policy_effect]  

e = some(where (p.eft == allow))  
  

[matchers]  

m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
`)
if err != nil {
    log.Fatalf("error: model: %s", err)
}
```

```
import casbin
import casbin_sqlalchemy_adapter

# Use SQLAlchemy Casbin adapter with SQLite DB
adapter = casbin_sqlalchemy_adapter.Adapter('sqlite:///test.db')

# Create a config model policy
with open("rbac_example_model.conf", "w") as f:
    f.write("""
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
""")

# Create enforcer from adapter and config policy
e = casbin.Enforcer('rbac_example_model.conf', adapter)
```

Check permissions

Add an enforcement hook into your code right before the access happens:

Go Java Node.js PHP Python .NET C++ Delphi

Rust Lua

```
sub := "alice" // the user that wants to access a resource.  
obj := "data1" // the resource that is going to be accessed.  
act := "read" // the operation that the user performs on the  
resource.  
  
ok, err := e.Enforce(sub, obj, act)  
  
if err != nil {  
    // handle err  
}  
  
if ok == true {  
    // permit alice to read data1  
} else {  
    // deny the request, show an error  
}  
  
// You could use BatchEnforce() to enforce some requests in  
// batches.  
// This method returns a bool slice, and this slice's index  
// corresponds to the row index of the two-dimensional array.  
// e.g. results[0] is the result of {"alice", "data1", "read"}  
results, err := e.BatchEnforce([][]interface{}){{"alice",  
"data1", "read"}, {"bob", "data2", "write"}, {"jack", "data3",  
"read"}})  
  
String sub = "alice"; // the user that wants to access a  
resource.  
String obj = "data1"; // the resource that is going to be  
accessed.
```

```
const sub = 'alice'; // the user that wants to access a
resource.
const obj = 'data1'; // the resource that is going to be
accessed.
const act = 'read'; // the operation that the user performs on
the resource.

if ((await e.enforce(sub, obj, act)) === true) {
    // permit alice to read data1
} else {
    // deny the request, show an error
}

$sub = "alice"; // the user that wants to access a resource.
$obj = "data1"; // the resource that is going to be accessed.
$act = "read"; // the operation that the user performs on the
resource.

if ($e->enforce($sub, $obj, $act) === true) {
    // permit alice to read data1
} else {
    // deny the request, show an error
}

sub = "alice" # the user that wants to access a resource.
obj = "data1" # the resource that is going to be accessed.
act = "read" # the operation that the user performs on the
resource.

if e.enforce(sub, obj, act):
    # permit alice to read data1
    pass
else:
    # deny the request, show an error
    pass
```

```

var sub = "alice"; # the user that wants to access a resource.
var obj = "data1"; # the resource that is going to be accessed.
var act = "read"; # the operation that the user performs on
the resource.

if (await e.EnforceAsync(sub, obj, act))
{
    // permit alice to read data1
}
else
{
    // deny the request, show an error
}

casbin::Enforcer e("../assets/model.conf", "../assets/
policy.csv");

if (e.Enforce({"alice", "/alice_data/hello", "GET"})) {
    std::cout << "Enforce OK" << std::endl;
} else {
    std::cout << "Enforce NOT Good" << std::endl;
}

if (e.Enforce({"alice", "/alice_data/hello", "POST"})) {
    std::cout << "Enforce OK" << std::endl;
} else {
    std::cout << "Enforce NOT Good" << std::endl;
}

if casbin.enforce(['alice,data1,read']) then
    // Alice is super happy as she can read data1
else
    // Alice is sad

```

```

let sub = "alice"; // the user that wants to access a
resource.
let obj = "data1"; // the resource that is going to be
accessed.
let act = "read"; // the operation that the user performs on
the resource.

if e.enforce((sub, obj, act)).await? {
    // permit alice to read data1
} else {
    // error occurs
}

if e:enforce("alice", "data1", "read") then
    -- permit alice to read data1
else
    -- deny the request, show an error
end

```

Casbin also provides API for permission management at run-time. For example, You can get all the roles assigned to a user as below:

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Delphi](#) [Rust](#)

[Lua](#)

```
roles, err := e.GetRolesForUser("alice")
```

```
List<String> roles = e.getRolesForUser("alice");
```

```
const roles = await e.getRolesForUser('alice');

$roles = $e->getRolesForUser("alice");

roles = e.get_roles_for_user("alice")

var roles = e.GetRolesForUser("alice");

roles = e.rolesForEntity("alice")

let roles = e.get_roles_for_user("alice");

local roles = e:GetRolesForUser("alice")
```

See [Management API](#) and [RBAC API](#) for more usage.

Please refer to the test cases for more usage.

How It Works

In Casbin, an access control model is abstracted into a CONF file based on the **PERM metamodel (Policy, Effect, Request, Matchers)**. Switching or upgrading the authorization mechanism for a project is as simple as modifying a configuration. You can customize your own access control model by combining the available models. For example, you can combine RBAC roles and ABAC attributes together inside one model and share one set of policy rules.

The PERM model is composed of four foundations: Policy, Effect, Request, and Matchers. These foundations describe the relationship between resources and users.

Request

Defines the request parameters. A basic request is a tuple object, requiring at least a subject (accessed entity), object (accessed resource), and action (access method).

For instance, a request definition may look like this: `r={sub, obj, act}`

This definition specifies the parameter names and ordering required by the access control matching function.

Policy

Defines the model for the access strategy. It specifies the name and order of the fields in the Policy rule document.

For instance: `p={sub, obj, act}` or `p={sub, obj, act, eft}`

Note: If eft (policy result) is not defined, the result field in the policy file will not be read, and the matching policy result will be allowed by default.

Matcher

Defines the matching rules for Request and Policy.

For example: `m = r.sub == p.sub && r.act == p.act && r.obj == p.obj`

This simple and common matching rule means that if the requested parameters (entities, resources, and methods) are equal to those found in the policy, then the policy result (`p.eft`) is returned. The result of the strategy will be saved in `p.eft`.

Effect

Performs a logical combination judgment on the matching results of Matchers.

For example: `e = some(where(p.eft == allow))`

This statement means that if the matching strategy result `p.eft` has the result of (some) allow, then the final result is true.

Let's look at another example:

```
e = some(where (p.eft == allow)) && !some(where (p.eft == deny))
```

The logical meaning of this example combination is: if there is a strategy that matches the result of allow and no strategy that matches the result of deny, the result is true. In other words, it is true when the matching strategies are all allow. If there is any deny, both are false (more simply, when allow and deny exist at the same time, deny takes precedence).

The most basic and simplest model in Casbin is ACL. The model CONF for ACL is

as follows:

```
# Request definition
[request_definition]
r = sub, obj, act

# Policy definition
[policy_definition]
p = sub, obj, act

# Policy effect
[policy_effect]
e = some(where (p.eft == allow))

# Matchers
[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

An example policy for the ACL model is:

```
p, alice, data1, read
p, bob, data2, write
```

This means:

- alice can read data1
- bob can write data2

We also support multi-line mode by appending '\' in the end:

```
# Matchers
[matchers]
```

Furthermore, if you are using ABAC, you can try the 'in' operator as shown in the following example for the Casbin golang edition (jCasbin and Node-Casbin are not supported yet):

```
# Matchers
[matchers]
m = r.obj == p.obj && r.act == p.act || r.obj in ('data2',
'data3')
```

But you **MUST** make sure that the length of the array is **MORE** than 1, otherwise it will cause a panic.

For more operators, you may take a look at [govaluate](#).

Tutorials

Before reading, please note that some tutorials are for the Casbin's model and work for all Casbin implementations in different languages. Some other tutorials are language-specific.

Our Papers

- [PML: An Interpreter-Based Access Control Policy Language for Web Services](#)

This paper digs deeply into the design details about Casbin. Please cite the following BibTex if you use Casbin/PML as a reference in your paper:

```
@article{luo2019pml,  
    title={PML: An Interpreter-Based Access Control Policy  
Language for Web Services},  
    author={Luo, Yang and Shen, Qingni and Wu, Zhonghai},  
    journal={arXiv preprint arXiv:1903.09756},  
    year={2019}  
}
```

- [Access Control Policy Specification Language Based on Metamodel \(in Chinese\)](#)

This is another longer-version paper published in Journal of Software. The citation for different formats (Refworks, EndNote, etc.) can be found at: ([another version](#)) [Access Control Policy Specification Language Based on Metamodel \(in Chinese\)](#)

Videos

- [A Secure Vault - implementing authorization middleware with Casbin -](#)

JuniorDevSG

- Sharing user permissions in a micro-service architecture based on Casbin (in Russian)
- Nest.js - Casbin RESTful RBAC authorization middleware
- Gin Tutorial Chapter 10: Learn Casbin basic models in 30 minutes (in Chinese)
- Gin Tutorial Chapter 11: Coding, API and custom function in Casbin (in Chinese)
- Gin + Casbin: Learning Permissions in Action (in Chinese)
- jCasbin Basics: A simple RBAC example (in Chinese)
- Golang's RBAC based on Casbin (in Chinese)
- Learning Gin + Casbin (1): Opening & Overview (in Chinese)
- ThinkPHP 5.1 + Casbin: Introduction (in Chinese)
- ThinkPHP 5.1 + Casbin: RBAC authorization (in Chinese)
- ThinkPHP 5.1 + Casbin: RESTful & Middleware (in Chinese)
- Quick Start for PHP-Casbin (in Chinese)
- ThinkPHP 5.1 + Casbin: How to use custom matching functions (in Chinese)
- Webman + Casbin: How to use Webman Casbin Plugin (in Chinese)

PERM Meta-Model (Policy, Effect, Request, Matchers)

- Understanding Casbin with different Access Control Model Configurations
- Modeling Authorization with PERM in Casbin
- Designing a Flexible Permissions System with Casbin
- Authorize with Access Control Lists
- Access control with PERM and Casbin (in Persian)
- RBAC? ABAC? .. PERM! New Way of Authorization for Cloud-Based Web Services and Apps (in Russian)
- Practice & Examples of Flexible Authorization Using Casbin & PERM (in

Russian)

- Permission management with Casbin (in Chinese)
- Analysis of Casbin (in Chinese)
- Design of System Permissions (in Chinese)
- Casbin: A Permission Engine (in Chinese)
- Implementing ABAC with Casbin (in Chinese)
- Source code analysis of Casbin (in Chinese)
- Permission evaluation with Casbin (in Chinese)
- Casbin: Library of the day for Go (in Chinese)

Go

Java

Node.js

PHP

.NET

Rust

Lua

HTTP & RESTful

- Basic Role-Based HTTP Authorization in Go with Casbin (or [Chinese translation](#))

Watcher

- RBAC Distributed Synchronization via Casbin Watcher (in Chinese)

Beego

- Using Casbin with Beego: 1. Get started and test (in Chinese)
- Using Casbin with Beego: 2. Policy storage (in Chinese)
- Using Casbin with Beego: 3. Policy query (in Chinese)
- Using Casbin with Beego: 4. Policy update (in Chinese)
- Using Casbin with Beego: 5. Policy update (continued) (in Chinese)

Gin

- Authorization in Golang Projects using Casbin

- Tutorial: Integrate Gin with Casbin
- Policy enforcements on K8s with Pipeline
- Authentication and authorization in Gin application with JWT and Casbin
- Backend API with Go: 1. Authentication based on JWT (in Chinese)
- Backend API with Go: 2. Authorization based on Casbin (in Chinese)
- Using Go's authorization library Casbin with Gin and GORM (in Japanese)

Echo

- Web authorization with Casbin

Iris

- Iris + Casbin: Practice for permission management (in Chinese)
- Learning iris + Casbin from scratch

Argo CD

- Organizational RBAC in Argo CD with Casbin

GShark

- GShark: Scan for sensitive information in Github easily and effectively (in Chinese)

SpringBoot

- jCasbin: a more light-weight permission management solution (in Chinese)
- Integrating jCasbin with JFinal (in Chinese)

Express

- How to Add Role-Based-Access-Control to Your Serverless HTTP API on AWS

Koa

- Authorisation with Casbin and Koa Part 1

- Authorisation with Casbin and Koa Part 2

Nest

- How to Create Role-based Authorization Middleware with Casbin and Nest.js
- nest.js: Casbin RESTful RBAC authorization middleware (Video)
- A Demo App of Attribute-based Access Control in Node.js Based on Casbin
- Multi tenant SaaS starter kit with cqrs graphql microservice architecture

Fastify

- Access Control in Node.js with Fastify and Casbin
- Casbin, Powerful and Efficient ACL for Your Projects
- Using Casbin for authorization in dotnet
- Basic Role-Based HTTP Authorization in Rust with Casbin
- How to use casbin authorization in your rust web-app [Part - 1]
- How to use casbin authorization in your rust web-app [Part - 2]

APISIX

- Authorization in APISIX using Casbin

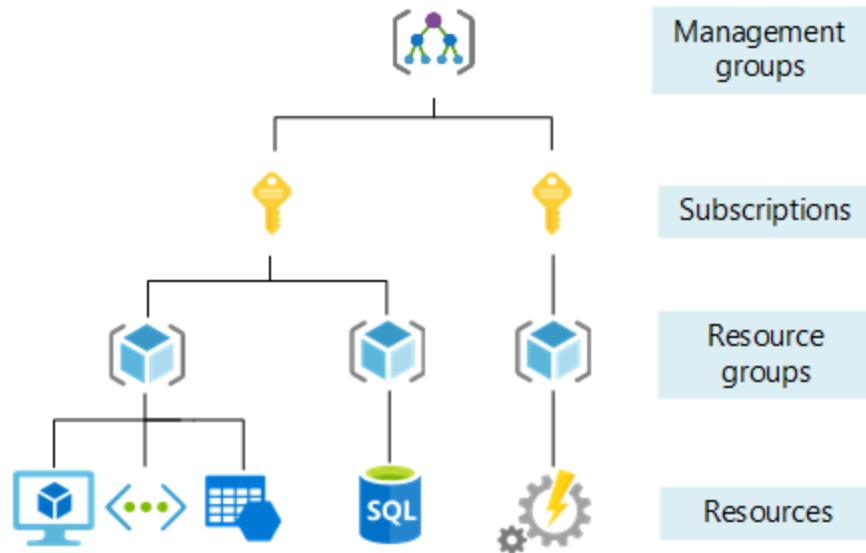
Understanding How Casbin Matching Works in Detail

In this post, I will explain the design and implementation of RBAC using the [Casbin](#) library. For a SaaS platform dealing with multiple resource hierarchies and roles that inherit permissions from higher levels, Casbin provides a performant alternative to consider.

Introduction to RBAC

RBAC is a method of restricting access to resources based on the roles that individuals hold. To better understand how hierarchical RBAC works, let's take a look at Azure's RBAC system in the next section and then attempt to implement a similar system.

Understanding Azure's Hierarchical RBAC



There is a role called **Owner** for all resources in Azure. Suppose if I have the **Owner** role assigned to me at the subscription level, that means I am the **Owner** of all the resource groups and resources under that subscription. If I have **Owner** at the resource group level, then I am the **Owner** of all the resources under that resource group.

This image shows that I have **Owner** access at the subscription level.

Microsoft Azure

Home > Subscriptions > pay-as-you-go

pay-as-you-go | Access control (IAM)

Subscription

Search

Add Download role assignments Edit columns Refresh Remove Feedback

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security Events Cost Management Cost analysis Cost alerts Budgets Advisor recommendations Billing Invoices External services Payment methods Partner information Settings Programmatic deployment Resource groups Resources Preview features

Number of role assignments for this subscription: 31 / 4000

Type: All Role: All Scope: All scopes Group by: Role

Showing a filtered set of results. Total number of role assignments: 31

Name	Type	Role	Scope	Condition
aravindkumar	User	Owner	This resource	None

When I check the IAM of a Resource Group under this Subscription, you can see that I have inherited Owner access from the subscription.

Microsoft Azure

Home > Resource groups > test-resource-group

test-resource-group | Access control (IAM)

Resource group

Search Add Download role assignments Edit columns Refresh Remove Feedback

Overview Activity log Access control (IAM) Tags Resource visualizer Events

Settings Deployments Security Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview) Budgets Advisor recommendations

Monitoring Insights (preview) Alerts Metrics Diagnostic settings

Number of role assignments for this subscription: 31 / 4000

Type: All Role: All Scope: All scopes Group by: Role

Showing a filtered set of results. Total number of role assignments: 31

Name	Type	Role	Scope	Condition
aravindkumar	User	Owner	Subscription (inherited)	None

So, this is how Azure's RBAC is hierarchical. Most enterprise software uses hierarchical RBAC because of the hierarchical nature of the resource levels. In this

tutorial, we'll try to implement a similar system using Casbin.

How Does Casbin Work?

Before diving into the implementation, it is important to understand what Casbin is and how it functions at a high level. This understanding is necessary because each Role-Based Access Control (RBAC) system may vary based on specific requirements. By grasping the workings of Casbin, we can effectively fine-tune the model.

What is ACL?

ACL stands for Access Control List. It is a method in which users are mapped to actions and actions to resources.

The model definition

Let's consider a simple example of an ACL model.

```
[request_definition]
r = sub, act, obj

[policy_definition]
p = sub, act, obj

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

1. The `request_definition` is the query template of the system. For example, a request `alice, write, data1` can be interpreted as "Can subject Alice perform the action 'write' on object 'data1'?".
2. The `policy_definition` is the assignment template of the system. For example, by creating a policy `alice, write, data1`, you are assigning permission to subject Alice to perform the action 'write' on object 'data1'.
3. The `policy_effect` defines the effect of the policy.
4. In the `matchers` section, the request is matched with the policy using the conditions `r.sub == p.sub && r.obj == p.obj && r.act == p.act`.

Now let's test the model on the Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

```
p, alice, read, data1  
p, bob, write, data2
```

and the following in the Request editor:

```
alice, read, data1
```

The result will be:

```
true
```

Visual representation of the ACL model, policy, and request matching



What is RBAC?

RBAC stands for Role-Based Access Control. In RBAC, a user is assigned a role for a resource, and a role can contain arbitrary actions. The request then checks if the user has the permission to perform the action on the resource.

The model definition

Let's consider a simple example RBAC model:

```
[request_definition]
```

```
r = sub, act, obj
```

```
[policy_definition]
```

```
p = sub, act, obj
```

1. The role_definition is a graph relation builder that uses a Graph to compare the request object with the policy object.

Now let's test the model on Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

```
p, alice, reader, data1  
p, bob, owner, data2
```

```
g, reader, read  
g, owner, read  
g, owner, write
```

and the following in the Request editor:

```
alice, read, data1  
alice, write, data1  
bob, write, data2  
bob, read, data2  
bob, write, data1
```

The result will be:

```
true  
false  
true  
true  
false
```

Visual representation of the RBAC model, policy, and request matching



The g - Role to action mapping table has a Graph mapping the role to action. This Graph can be coded as a list of edges, as shown in the policy which is a common way of representing a Graph:

```
g, reader, read  
g, owner, read  
g, owner, write
```

INFO

p indicates a normal policy that can be compared using the $==$ operator. g is a Graph-based comparison function. You can define multiple Graph comparators by adding a numerical suffix like g , $g2$, $g3$, ... and so on.

What is Hierarchical RBAC?

In Hierarchical RBAC, there are more than one type of resources and there is an inheritance relationship between the resource types. For example, "subscription" is one type and "resourceGroup" is another type. A sub1 of type Subscription can contain multiple resourceGroups (rg1, rg2) of type ResourceGroup.

Similar to the resource hierarchy, there will be two types of roles and actions: Subscription roles and actions, and ResourceGroup roles and actions. There is an arbitrary relationship between the Subscription role and ResourceGroup role. For example, consider a Subscription Role **sub-owner**. This role is inherited by a ResourceGroup Role **rg-owner**, which means that if I am assigned the **sub-owner** role on Subscription **sub1**, then I automatically also get the **rg-owner** role on **rg1** and **rg2**.

The model definition

Let's take a simple example of the Hierarchical RBAC model:

```
[request_definition]
r = sub, act, obj

[policy_definition]
p = sub, act, obj

[role_definition]
g = _, _
g2 = _, _

[policy_effect]
e = some(where (p.eft == allow))
```

1. The role_definition is a graph relation builder which uses a Graph to compare the request object with the policy object.

Now let's test the model on the Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

```
p, alice, sub-reader, sub1
p, bob, rg-owner, rg2

// subscription role to subscription action mapping
g, sub-reader, sub-read
g, sub-owner, sub-read
g, sub-owner, sub-write

// resourceGroup role to resourceGroup action mapping
g, rg-reader, rg-read
g, rg-owner, rg-read
g, rg-owner, rg-write

// subscription role to resourceGroup role mapping
g, sub-reader, rg-reader
g, sub-owner, rg-owner

// subscription resource to resourceGroup resource mapping
g2, sub1, rg1
g2, sub2, rg2
```

And paste the following in the Request editor:

```
alice, rg-read, rg1
```

The result will be:

true

Visual representation of the RBAC model, policy, and request matching



The **g - Role to (Action, Role) Mapping** table has a graph mapping the role to the action, role mapping. This graph can be coded as a list of edges, as shown in the policy, which is a common way of representing a graph:

```

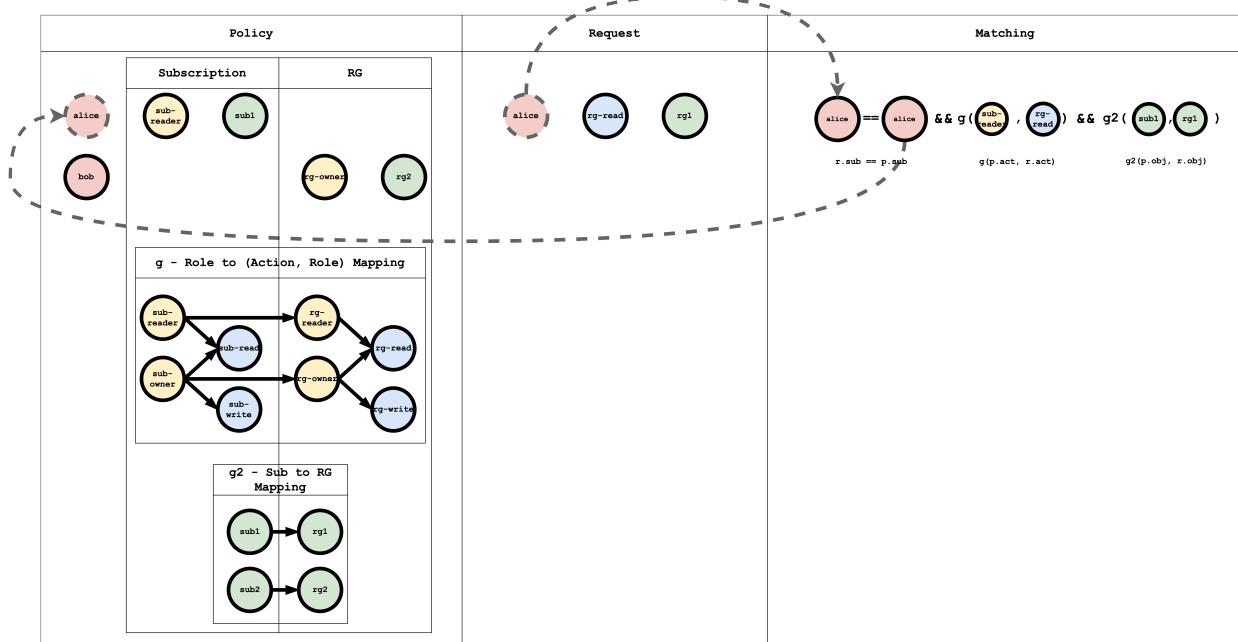
// subscription role to subscription action mapping
g, sub-reader, sub-read
g, sub-owner, sub-read
g, sub-owner, sub-write

// resourceGroup role to resourceGroup action mapping
  
```

The g2 - Sub to RG Mapping table has a graph mapping subscription to resourceGroup:

```
// subscription resource to resourceGroup resource mapping
g2, sub1, rg1
g2, sub2, rg2
```

Subject Matching Visual representation



Action Matching Visual representation



Object Matching Visual representation



 INFO

When a request is submitted to Casbin, this matching happens for all the policies. If at least one policy matches, then the result of the request is true. If no policy matches the request, then the result is false.

Conclusion

In this tutorial, we learned about how different authorization models work and how they can be modeled using Casbin. In the second part of this tutorial, we will implement this in a demo Spring Boot Application and secure the APIs using Casbin.

Model

Supported Models

Supported models of Casbin

Syntax for Models

Syntax for Models

Effector

The Effector interface in Casbin

Functions

Using built-in functions or specifying custom functions

RBAC

Casbin RBAC usage

RBAC with Pattern

RBAC with Pattern

RBAC with Domains

Usage of RBAC with domains

RBAC with Conditions

Usage of RBAC with conditions

Casbin RBAC vs. RBAC96

The Difference Between Casbin RBAC and RBAC96

ReBAC

ReBAC based on Casbin

ABAC

ABAC based on Casbin

PBAC

Policy-Based Access Control model in Casbin

BLP

Bell-LaPadula model in Casbin

Biba

Biba Integrity Model in Casbin

LBAC

Lattice-Based Access Control model in Casbin

Priority Model

Casbin's Priority Model for managing policies with different priorities

Usage Control (UCON)

Usage Control Model in Casbin

Super Admin

The Super Admin is the administrator of the entire system. It can be used in models such as RBAC, ABAC, and RBAC with domains.

Supported Models

1. **ACL (Access Control List)**
2. ACL with **superuser**
3. **ACL without users:** This is especially useful for systems that don't have authentication or user logins.
4. **ACL without resources:** In some scenarios, the target is a type of resource instead of an individual resource. Permissions like "write-article" and "read-log" can be used. This doesn't control access to a specific article or log.
5. **RBAC (Role-Based Access Control)**
6. **RBAC with resource roles:** Both users and resources can have roles (or groups) at the same time.
7. **RBAC with domains/tenants:** Users can have different sets of roles for different domains/tenants.
8. **ABAC (Attribute-Based Access Control):** Syntax sugar like "resource.Owner" can be used to get the attribute for a resource.
9. **PBAC (Policy-Based Access Control):** A flexible access control model that makes authorization decisions based on policies defined as rules, supporting dynamic and contextual authorization.
10. **BLP (Bell-LaPadula):** A formal state transition model of computer security policy that describes a set of access control rules which use security labels on objects and clearances for subjects.
11. **Biba (Biba Integrity Model):** A computer security model that restricts information flow in a system to prevent unauthorized disclosure of classified information.
12. **LBAC (Lattice-Based Access Control):** A formal access control model that combines confidentiality and integrity controls in a unified framework, implementing a lattice structure for granular access control decisions.
13. **UCON (Usage Control):** A next-generation access control model that emphasizes continuous authorization, attribute mutability, and a unified framework of authorizations, obligations, and conditions.
14. **RESTful:** Supports paths like "/res/*", "/res/:id", and HTTP methods like "GET", "POST", "PUT", "DELETE".
15. **IP Match:** Supports IP address matching for network-based access control.

16. Deny-override: Both allow and deny authorizations are supported, where deny overrides allow.
17. Priority: The policy rules can be prioritized, similar to firewall rules.

Examples

Model	Model file	Policy file
ACL	basic_model.conf	basic_policy.csv
ACL with superuser	basic_with_root_model.conf	basic_policy.csv
ACL without users	basic_without_users_model.conf	basic_without_users_policy.csv
ACL without resources	basic_without_resources_model.conf	basic_without_resources_policy.csv
RBAC	rbac_model.conf	rbac_policy.csv
RBAC with resource roles	rbac_with_resource_roles_model.conf	rbac_with_resource_roles_policy.csv
RBAC with domains/tenants	rbac_with_domains_model.conf	rbac_with_domains_policy.csv
ReBAC	rebac_model.conf	rebac_policy.csv

Model	Model file	Policy file
ABAC	abac_model.conf	N/A
BLP	blp_model.conf	N/A
Biba	biba_model.conf	N/A
LBAC	lbac_model.conf	N/A
IP Match	ipmatch_model.conf	ipmatch_policy.csv
RESTful	keymatch_model.conf	keymatch_policy.csv
Deny-override	rbac_with_not_deny_model.conf	rbac_with_deny_policy.csv
Allow-and-deny	rbac_with_deny_model.conf	rbac_with_deny_policy.csv
Priority	priority_model.conf	priority_policy.csv
Explicit Priority	priority_model_explicit	priority_policy_explicit.csv
Subject-Priority	subject_priority_model.conf	subject_priority_policy.csv

Syntax for Models

- A model configuration (CONF) should have at least four sections:
[request_definition], [policy_definition], [policy_effect], and
[matchers].
- If a model uses Role-Based Access Control (RBAC), it should also include the
[role_definition] section.
- A model configuration (CONF) can contain comments. Comments start with
the # symbol, and everything after the # symbol will be commented out.

Request definition

The [request_definition] section defines the arguments in the
e.Enforce(...) function.

```
[request_definition]
r = sub, obj, act
```

In this example, sub, obj, and act represent the classic access triple: the subject (accessing entity), the object (accessed resource), and the action (access method). However, you can customize your own request format. For example, you can use sub, act if you don't need to specify a particular resource, or sub,
sub2, obj, act if you have two accessing entities.

Policy Definition

The `[policy_definition]` is the definition for a policy. It defines the meaning of the policy. For example, we have the following model:

```
[policy_definition]
p = sub, obj, act
p2 = sub, act
```

And we have the following policy (if in a policy file):

```
p, alice, data1, read
p2, bob, write-all-objects
```

Each line in a policy is called a policy rule. Each policy rule starts with a `policy type`, such as `p` or `p2`. It is used to match the policy definition if there are multiple definitions. The above policy shows the following binding. The binding can be used in the matcher.

```
(alice, data1, read) -> (p.sub, p.obj, p.act)
(bob, write-all-objects) -> (p2.sub, p2.act)
```



The elements in a policy rule are always regarded as `strings`. If you have any questions about this, please refer to the discussion at:
<https://github.com/casbin/casbin/issues/113>

Policy Effect

[policy_effect] is the definition for the policy effect. It determines whether the access request should be approved if multiple policy rules match the request. For example, one rule permits and the other denies.

```
[policy_effect]
e = some(where (p.eft == allow))
```

The above policy effect means that if there's any matched policy rule of `allow`, the final effect is `allow` (also known as allow-override). `p.eft` is the effect for a policy, and it can be either `allow` or `deny`. It is optional, and the default value is `allow`. Since we didn't specify it above, it uses the default value.

Another example for the policy effect is:

```
[policy_effect]
e = !some(where (p.eft == deny))
```

This means that if there are no matched policy rules of `deny`, the final effect is `allow` (also known as deny-override). `some` means that there exists one matched policy rule. `any` means that all matched policy rules (not used here). The policy effect can even be connected with logical expressions:

```
[policy_effect]
e = some(where (p.eft == allow)) && !some(where (p.eft == deny))
```

This means that there must be at least one matched policy rule of `allow`, and

there cannot be any matched policy rule of `deny`. Therefore, in this way, both allow and deny authorizations are supported, and the deny overrides.

 NOTE

Although we designed the syntax of the policy effect as above, the current implementations only use hard-coded policy effects. This is because we found that there isn't much need for that level of flexibility. So for now, you must use one of the built-in policy effects instead of customizing your own.

The supported built-in policy effects are:

Policy Effect	Meaning	Example
<code>some(where (p.eft == allow))</code>	allow-override	ACL, RBAC, etc.
<code>!some(where (p.eft == deny))</code>	deny-override	Deny- override
<code>some(where (p.eft == allow)) && !some(where (p.eft == deny))</code>	allow-and- deny	Allow-and- deny
<code>priority(p.eft) deny</code>	priority	Priority
<code>subjectPriority(p.eft)</code>	priority based on role	Subject- Priority

Matchers

[matchers] is the definition for policy matchers. The matchers are expressions that define how the policy rules are evaluated against the request.

```
[matchers]  
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

The above matcher is the simplest and means that the subject, object, and action in a request should match the ones in a policy rule.

Arithmetic operators like +, -, *, / and logical operators like &&, ||, ! can be used in matchers.

Order of expressions in matchers

The order of expressions can greatly affect performance. Take a look at the following example for more details:

```
const rbac_models = `  
[request_definition]  
r = sub, obj, act  
  
[policy_definition]  
p = sub, obj, act  
  
[role_definition]  
g = _, _  
  
[policy_effect]
```

The enforcement time may be very long, up to 6 seconds.

```
go test -run ^TestManyRoles$ github.com/casbin/casbin/v2 -v

==== RUN TestManyRoles
    rbac_api_test.go:598: RESPONSE abu
/projects/1      GET : true IN: 438.379µs
    rbac_api_test.go:598: RESPONSE abu      /projects/
2499      GET : true IN: 39.005173ms
    rbac_api_test.go:598: RESPONSE jasmine
/projects/1      GET : true IN: 1.774319ms
    rbac_api_test.go:598: RESPONSE jasmine      /projects/
2499      GET : true IN: 6.164071648s
    rbac_api_test.go:600: More than 100 milliseconds for
jasmine /projects/2499 GET : 6.164071648s
    rbac_api_test.go:598: RESPONSE jasmine      /projects/
2499      GET : true IN: 12.164122ms
--- FAIL: TestManyRoles (6.24s)
FAIL
FAIL      github.com/casbin/casbin/v2      6.244s
FAIL
```

However, if we adjust the order of the expressions in matchers and put more time-consuming expressions like functions behind, the execution time will be very short.

Changing the order of expressions in matchers in the above example to:

```
[matchers]
m = r.obj == p.obj && g(r.sub, p.sub) && r.act == p.act
```

```
go test -run ^TestManyRoles$ github.com/casbin/casbin/v2 -v
==== RUN TestManyRoles
```

Multiple Section Types

If you need multiple policy definitions or multiple matchers, you can use `p2` or `m2` as examples. In fact, all four sections mentioned above can use multiple types, and the syntax is `r` followed by a number, such as `r2` or `e2`. By default, these four sections should correspond one-to-one. For example, your `r2` section will only use the `m2` matcher to match `p2` policies.

You can pass an `EnforceContext` as the first parameter of the `enforce` method to specify the types. The `EnforceContext` is defined as follows:

[Go](#) [Node.js](#) [Java](#)

```
EnforceContext{"r2", "p2", "e2", "m2"}  
type EnforceContext struct {  
    RType string  
    PType string  
    EType string  
    MType string  
}  
  
const enforceContext = new EnforceContext('r2', 'p2', 'e2',  
    'm2');  
class EnforceContext {  
    constructor(rType, pType, eType, mType) {  
        this.pType = pType;  
        this.eType = eType;  
        this.mType = mType;  
        this.rType = rType;  
    }  
}
```

```
EnforceContext enforceContext = new EnforceContext("2");
public class EnforceContext {
    private String pType;
    private String eType;
    private String mType;
    private String rType;
    public EnforceContext(String suffix) {
        this.pType = "p" + suffix;
        this.eType = "e" + suffix;
        this.mType = "m" + suffix;
        this.rType = "r" + suffix;
    }
}
```

Here is an example usage. Please refer to the [model](#) and [policy](#). The request is as follows:

[Go](#) [Node.js](#) [Java](#)

```
// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
enforceContext := NewEnforceContext("2")
// You can also specify a certain type individually
enforceContext.EType = "e"
// Don't pass in EnforceContext; the default is r, p, e, m
e.Enforce("alice", "data2", "read")           // true
// Pass in EnforceContext
e.Enforce(enforceContext, struct{ Age int }{Age: 70}, "/data1",
"read")           //false
e.Enforce(enforceContext, struct{ Age int }{Age: 30}, "/data1",
"read")           //true
```

```
// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
const enforceContext = new NewEnforceContext('2');

// You can also specify a certain type individually
enforceContext.eType = "e"

// Don't pass in EnforceContext; the default is r, p, e, m
e.Enforce("alice", "data2", "read")           // true

// Pass in EnforceContext
e.Enforce(enforceContext, {Age: 70}, "/data1", "read")
//false
e.Enforce(enforceContext, {Age: 30}, "/data1", "read")
//true

// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
EnforceContext enforceContext = new EnforceContext("2");
// You can also specify a certain type individually
enforceContext.setType("e");
// Don't pass in EnforceContext; the default is r, p, e, m
e.enforce("alice", "data2", "read"); // true
// Pass in EnforceContext
// TestEvalRule is located in https://github.com/casbin/jcasbin/
// blob/master/src/test/java/org/casbin/jcasbin/main/
AbacAPIUnitTest.java#L56
e.enforce(enforceContext, new
AbacAPIUnitTest.TestEvalRule("alice", 70), "/data1", "read");
// false
e.enforce(enforceContext, new
AbacAPIUnitTest.TestEvalRule("alice", 30), "/data1", "read");
// true
```

Special Grammar

You could also use the "in" operator, which is the only operator with a text name. This operator checks the array on the right-hand side to see if it contains a value that is equal to the value on the left side. Equality is determined by using the == operator, and this library does not check the types between the values. As long as two values can be cast to interface{} and can still be checked for equality with ==, they will act as expected. Note that you can use a parameter for the array, but it must be an []interface{}.

Also refer to [rbac_model_matcher_using_in_op](#), [keyget2_model](#), and [keyget_model](#).

Example:

```
[request_definition]
r = sub, obj
...
[matchers]
m = r.sub.Name in (r.obj.Admins)
```

```
e.Enforce(Sub{Name: "alice"}, Obj{Name: "a book", Admins:
[]interface{}{"alice", "bob"}})
```

Expression Evaluator

The matcher evaluation in Casbin is implemented by expression evaluators in each language. Casbin integrates their powers to provide the unified PERM language. In addition to the model syntax provided here, these expression evaluators may offer

extra functionality that might not be supported by another language or implementation. Please be cautious when using this functionality.

The expression evaluators used by each Casbin implementation are as follows:

Implementation	Language	Expression Evaluator
Casbin	Golang	https://github.com/casbin/govaluate
jCasbin	Java	https://github.com/killme2008/aviator
Node-Casbin	Node.js	https://github.com/donmccurdy/expressioneval
PHP-Casbin	PHP	https://github.com/symfony/expression-language
PyCasbin	Python	https://github.com/danthedeckie/simpleeval
Casbin.NET	C#	https://github.com/davideicardi/DynamicExpresso
Casbin4D	Delphi	https://github.com/casbin4d/Casbin4D/tree/master/SourceCode/Common/Third%20Party/TExpressionParser
casbin-rs	Rust	https://github.com/jonathandturner/rhai
casbin-cpp	C++	https://github.com/ArashPartow/exprtk

 NOTE

If you encounter a performance issue with Casbin, it is likely caused by the low efficiency of the expression evaluator. You can address the issue to Casbin or the expression evaluator directly for advice on speeding up the performance. For more details, please refer to the [Benchmarks](#) section.

Effector

The `Effect` represents the result of a policy rule, and the `Effector` is the interface for handling effects in Casbin.

MergeEffects()

The `MergeEffects()` function is used to merge all matching results collected by the enforcer into a single decision.

For example:

Go

```
Effect, explainIndex, err = e.MergeEffects(expr, effects,  
matches, policyIndex, policyLength)
```

In this example:

- `Effect` is the final decision that is merged by this function (initialized as `Indeterminate`).
- `explainIndex` is the index of `eft` (`Allow` or `Deny`), and it is initialized as `-1`.
- `err` is used to check if the effect is supported.
- `expr` is the string representation of the policy effects.
- `effects` is an array of effects, which can be `Allow`, `Indeterminate`, or `Deny`.

- `matches` is an array that indicates whether the result matches the policy.
- `policyIndex` is the index of the policy in the model.
- `policyLength` is the length of the policy.

The code above illustrates how to pass the parameters to the `MergeEffects()` function, and the function will process the effects and matches based on the `expr`.

To use the `Effector`, follow these steps:

Go

```
var e Effector
Effect, explainIndex, err = e.MergeEffects(expr, effects,
matches, policyIndex, policyLength)
```

The basic idea of `MergeEffects()` is that if the `expr` can match the results, indicating that the `p_eft` is `allow`, then all effects can be merged. If no deny rules are matched, then the decision is `allow`.

NOTE

If the `expr` does not match the condition `"priority(p_eft) || deny"`, and the `policyIndex` is shorter than `policyLength-1`, it will short-circuit some effects in the middle.

Functions

Functions in matchers

You can even specify functions in a matcher to make it more powerful. You can use built-in functions or specify your own function. The built-in key-matching functions take the following format:

```
bool function_name(string url, string pattern)
```

They return a boolean indicating whether the `url` matches the `pattern`.

The supported built-in functions are:

Function	url	pattern	Example
keyMatch	a URL path like <code>/alice_data/resource1</code>	a URL path or a <code>*</code> pattern like <code>/alice_data/*</code>	<code>keymatch_model.conf/keymatch_policy.csv</code>
keyMatch2	a URL path like <code>/alice_data/resource1</code>	a URL path or a <code>:</code> pattern like <code>/alice_data/:resource</code>	<code>keymatch2_model.conf/keymatch2_policy.csv</code>
keyMatch3	a URL path like <code>/alice_data/resource1</code>	a URL path or a <code>{}</code> pattern like <code>/alice_data/{resource}</code>	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L171-L196
keyMatch4	a URL path like <code>/alice_data/123/book/123</code>	a URL path or a <code>{}</code> pattern like <code>/alice_data/{id}/book/{id}</code>	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L208-L222
keyMatch5	a URL path like <code>/alice_data/123/?status=1</code>	a URL path, a <code>{}</code> or <code>*</code> pattern like <code>/alice_data/{id}/*</code>	https://github.com/casbin/casbin/blob/1cde2646d10ad1190c0d784c3a1c0e1ace1b5bc9/util/builtin_operators_test.go#L485-L526
regexMatch	any string	a regular expression pattern	<code>keymatch_model.conf/keymatch_policy.csv</code>
ipMatch	an IP address like <code>192.168.2.123</code>	an IP address or a CIDR like <code>192.168.2.0/24</code>	<code>ipmatch_model.conf/ipmatch_policy.csv</code>
globMatch	a path-like path like <code>/alice_data/resource1</code>	a glob pattern like <code>/alice_data/*</code>	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L426-L466

For key-getting functions, they usually take three parameters (except `keyGet`):

```
bool function_name(string url, string pattern, string key_name)
```

They will return the value of the key `key_name` if it matches the pattern, and return `""` if nothing is matched.

For example, `KeyGet2("/resource1/action",("/:res/action", "res")` will return `"resource1"`, and `KeyGet3("/resource1_admin/action", "{res}_admin/*", "res")` will return `"resource1"`. As for `KeyGet`, which takes two parameters, `KeyGet("/resource1/action", "/")` will return `"resource1/action"`.

Function	url	pattern	key_name	example
keyGet	a URL path like <code>/proj/resource1</code>	a URL path or a <code>*</code> pattern like <code>/proj/*</code>	\	<code>keyget_model.conf/keymatch_policy.csv</code>
keyGet2	a URL path like <code>/proj/resource1</code>	a URL path or <code>:</code> pattern like <code>/proj/:resource</code>	key name specified in the pattern	<code>keyget2_model.conf/keymatch2_policy.csv</code>
keyGet3	a URL path like <code>/proj/res3_admin/</code>	a URL path or <code>{}</code> pattern like <code>/proj/{resource}_admin/*</code>	key name specified in the pattern	<code>https://github.com/casbin/casbin/blob/7bd496f94f5a2739a392d333a9aaa10ae397673/util/builtin_operators_test.go#L209-L247</code>

See details for the above functions at: https://github.com/casbin/casbin/blob/master/util/builtin_operators_test.go

How to add a customized function

First, prepare your function. It takes several parameters and returns a bool:

```
func KeyMatch(key1 string, key2 string) bool {
    i := strings.Index(key2, "*")
    if i == -1 {
        return key1 == key2
    }

    if len(key1) > i {
        return key1[:i] == key2[:i]
    }
    return key1 == key2[:i]
}
```

Then, wrap it with `interface{}` types:

```
func KeyMatchFunc(args ...interface{}) (interface{}, error) {
    name1 := args[0].(string)
    name2 := args[1].(string)
```

Finally, register the function to the Casbin enforcer:

```
e.AddFunction("my_func", KeyMatchFunc)
```

Now, you can use the function in your model CONF like this:

```
[matchers]
m = r.sub == p.sub && my_func(r.obj, p.obj) && r.act == p.act
```

RBAC

Role Definition

The `[role_definition]` is used to define the RBAC role inheritance relations. Casbin supports multiple instances of RBAC systems, where users can have roles and their inheritance relations, and resources can have roles and their inheritance relations too. These two RBAC systems won't interfere with each other.

This section is optional. If you don't use RBAC roles in the model, then omit this section.

```
[role_definition]
g = _, _
g2 = _, _
```

The above role definition shows that `g` is an RBAC system, and `g2` is another RBAC system. `_, _` means there are two parties involved in an inheritance relation. In the most common case, you usually use `g` alone if you only need roles for users. You can also use both `g` and `g2` when you need roles (or groups) for both users and resources. Please see the [rbac_model.conf](#) and [rbac_model_with_resource_roles.conf](#) for examples.

Casbin stores the actual user-role mapping (or resource-role mapping if you are using roles on resources) in the policy. For example:

```
p, data2_admin, data2, read
g, alice, data2_admin
```

It means that `alice` inherits/is a member of the role `data2_admin`. Here, `alice`

can be a user, a resource, or a role. Casbin only recognizes it as a string.

Then, in a matcher, you should check the role as shown below:

```
[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

It means that the `sub` in the request should have the role `sub` in the policy.

ⓘ NOTE

1. Casbin only stores the user-role mapping.
2. Casbin doesn't verify whether a user is a valid user or a role is a valid role. That should be taken care of by authentication.
3. Do not use the same name for a user and a role inside an RBAC system, because Casbin recognizes users and roles as strings, and there's no way for Casbin to know whether you are specifying user `alice` or role `alice`. You can simply solve it by using `role_alice`.
4. If `A` has role `B`, and `B` has role `C`, then `A` has role `C`. This transitivity is infinite for now.

! TOKEN NAME CONVENTION

Conventionally, the subject token name in the policy definition is `sub` and placed at the beginning. Now, Golang Casbin supports customized token names and places. If the subject token name is `sub`, the subject token can be placed at an arbitrary place without any extra action needed. If the subject token name is not `sub`, `e.SetFieldIndex()` for `constant.SubjectIndex` should be called after the enforcer is initialized, regardless of its position.

```
# `subject` here is for sub
[policy_definition]
p = obj, act, subject

e.SetFieldIndex("p", constant.SubjectIndex, 2) // index
starts from 0
ok, err := e.DeleteUser("alice") // without SetFieldIndex,
it will raise an error
```

Role Hierarchy

Casbin's RBAC supports RBAC1's role hierarchy feature, which means that if `alice` has `role1`, and `role1` has `role2`, then `alice` will also have `role2` and inherit its permissions.

Here, we have a concept called a hierarchy level. So, in this example, the hierarchy level is 2. For the built-in role manager in Casbin, you can specify the maximum hierarchy level. The default value is 10. This means that an end user like `alice` can only inherit 10 levels of roles.

```
// NewRoleManager is the constructor for creating an instance
of the
// default RoleManager implementation.
func NewRoleManager(maxHierarchyLevel int) rbac.RoleManager {
    rm := RoleManager{}
    rm.allRoles = &sync.Map{}
    rm.maxHierarchyLevel = maxHierarchyLevel
    rm.hasPattern = false
```

How to Distinguish Role from User?

Casbin doesn't distinguish between roles and users in its RBAC. They are both treated as strings. If you only use a single-level RBAC (where a role will never be a member of another role), you can use `e.GetAllSubjects()` to get all users and `e.GetAllRoles()` to get all roles. They will list all `u` and all `r`, respectively, in all `g, u, r` rules.

But if you are using a multi-level RBAC (with role hierarchy) and your application doesn't record whether a name (string) is a user or a role, or you have a user and a role with the same name, you can add a prefix to the role like `role::admin` before passing it to Casbin. This way, you will know if it's a role by checking this prefix.

How to Query Implicit Roles or Permissions?

When a user inherits a role or permission via RBAC hierarchy instead of being directly assigned them in a policy rule, we call this type of assignment "implicit". To query such implicit relations, you need to use these two APIs:

`GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()` instead of `GetRolesForUser()` and `GetPermissionsForUser()`. For more details, please see [this GitHub issue](#).

Using Pattern Matching in RBAC

See [RBAC with Pattern](#)

Role Manager

See the [Role Managers](#) section for details.

RBAC with Pattern

Quick Start

- Use pattern in `g(_)`.

```
e, _ := NewEnforcer("./example.conf", "./example.csv")
e.AddNamedMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

- Use pattern with domain.

```
e.AddNamedDomainMatchingFunc("g", "KeyMatch2",
    util.KeyMatch2)
```

- Use all patterns.

Just combine the use of both APIs.

As shown above, after you create the `enforcer` instance, you need to activate pattern matching via the `AddNamedMatchingFunc` and `AddNamedDomainMatchingFunc` APIs, which determine how the pattern matches.

 NOTE

If you use the online editor, You can add a pattern matching function by clicking the "Add Role Matching" button in the lower left corner.

matchingForGFunction

X

```
1  (user, role) => {
2      return user.department ===
3          role.department;
4 }
```

Add
Function

Add Role
Matching

Add Domain
Matching



Use pattern matching in RBAC

Sometimes, you want certain subjects, objects, or domains/tenants with a specific pattern to be automatically granted a role. Pattern matching functions in RBAC can help you do that. A pattern matching function shares the same parameters and return value as the previous [matcher function](#).

The pattern matching function supports each parameter of `g`.

We know that normally RBAC is expressed as `g(r.sub, p.sub)` in a matcher.

Then we can use a policy like:

```
p, alice, book_group, read
```

So `alice` can read all books including `book 1` and `book 2`. But there can be thousands of books, and it's very tedious to add each book to the book role (or group) with one `g` policy rule.

But with pattern matching functions, you can write the policy with only one line:

```
g, /book/:id, book_group
```

Casbin will automatically match `/book/1` and `/book/2` into the pattern `/book/:id` for you. You only need to register the function with the enforcer like:

[Go](#) [Node.js](#)

```
e.AddNamedMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

```
await e.addNamedMatchingFunc('g', Util.keyMatch2Func);
```

When using a pattern matching function in domains/tenants, you need to register the function with the enforcer and model.

[Go](#) [Node.js](#)

```
e.AddNamedDomainMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

```
await e.addNamedDomainMatchingFunc('g', Util.keyMatch2Func);
```

If you don't understand what `g(r.sub, p.sub, r.dom)` means, please read [rbac-with-domains](#). In short, `g(r.sub, p.sub, r.dom)` will check whether the

user `r.sub` has a role `p.sub` in the domain `r.dom`. So this is how the matcher works. You can see the full example [here](#).

Apart from the pattern matching syntax above, we can also use pure domain pattern.

For example, if we want `sub` to have access in different domains, `domain1` and `domain2`, we can use the pure domain pattern:

```
p, admin, domain1, data1, read  
p, admin, domain1, data1, write  
p, admin, domain2, data2, read  
p, admin, domain2, data2, write  
  
g, alice, admin, *  
g, bob, admin, domain2
```

In this example, we want `alice` to read and write `data` in domain1 and domain2. Pattern matching `*` in `g` makes `alice` have access to two domains.

By using pattern matching, especially in scenarios that are more complicated and have a lot of domains or objects to consider, we can implement the `policy_definition` in a more elegant and effective way.

RBAC with Domains

Role Definition with Domain Tenants

The RBAC roles in Casbin can be global or domain-specific. Domain-specific roles mean that the roles for a user can be different when the user is in different domains/tenants. This is very useful for large systems like a cloud, as users are usually in different tenants.

The role definition with domains/tenants should look like this:

```
[role_definition]  
g = __, __, __
```

The third `__` represents the name of the domain/tenant, and this part should not be changed. Then the policy can be:

```
p, admin, tenant1, data1, read  
p, admin, tenant2, data2, read  
  
g, alice, admin, tenant1  
g, alice, user, tenant2
```

This means that the `admin` role in `tenant1` can read `data1`. And `alice` has the `admin` role in `tenant1` and the `user` role in `tenant2`. Therefore, she can read `data1`. However, since `alice` is not an `admin` in `tenant2`, she cannot read `data2`.

Then, in a matcher, you should check the role as follows:

```
[matchers]
m = g(r.sub, p.sub, r.dom) && r.dom == p.dom && r.obj == p.obj
&& r.act == p.act
```

Please refer to the [rbac_with_domains_model.conf](#) for examples.

(!) TOKEN NAME CONVENTION

Note: Conventionally, the domain token name in policy definition is `dom` and is placed as the second token (`sub, dom, obj, act`). Now, Golang Casbin supports customized token names and placement. If the domain token name is `dom`, the domain token can be placed at an arbitrary position without any additional action. If the domain token name is not `dom`, `e.SetFieldIndex()` for `constant.DomainIndex` should be called after the enforcer is initialized, regardless of its position.

```
# `domain` here for `dom`
[policy_definition]
p = sub, obj, act, domain
```

```
e.SetFieldIndex("p", constant.DomainIndex, 3) // index
starts from 0
users := e.GetAllUsersByDomain("domain1") // without
SetFieldIndex, it will raise an error
```

RBAC with Conditions

Conditional RoleManager

`ConditionalRoleManager` supports custom condition functions at the policy level.

For example, when we need a temporary role policy, we can follow the following approach:

`model.conf`

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _, (_, _)

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

`g = _, _, (_, _)` uses `(_, _)` to contain a list of arguments to pass to the condition function and `_` as a parameter placeholder

`policy.csv`

```
p, alice, data1, read
p, data2_admin, data2, write
p, data3_admin, data3, read
p, data4_admin, data4, write
p, data5_admin, data5, read
p, data6_admin, data6, write
p, data7_admin, data7, read
p, data8_admin, data8, write

g, alice, data2_admin, 0000-01-01 00:00:00, 0000-01-02 00:00:00
g, alice, data3_admin, 0000-01-01 00:00:00, 9999-12-30 00:00:00
g, alice, data4_admin, _, _
g, alice, data5_admin, _, 9999-12-30 00:00:00
g, alice, data6_admin, _, 0000-01-02 00:00:00
g, alice, data7_admin, 0000-01-01 00:00:00, _
g, alice, data8_admin, 9999-12-30 00:00:00, _
```

Basic Usage

Add a conditional function for the role policy(g type policy) through `AddNamedLinkConditionFunc`, and when enforcing is executed, the corresponding parameters will be automatically obtained and passed in the conditional function for checking. If the check passes, then the corresponding role policy(g type policy) is valid, otherwise it is invalid

```
e.AddNamedLinkConditionFunc("g", "alice", "data2_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data3_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data4_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data5_admin",
util.TimeMatchFunc)
```

Custom condition functions

Custom conditional functions need to conform to the following function types

```
type LinkConditionFunc = func(args ...string) (bool, error)
```

for example:

```
// TimeMatchFunc is the wrapper for TimeMatch.
func TimeMatchFunc(args ...string) (bool, error) {
    if err := validateVariadicStringArgs(2, args...); err != nil {
        return false, fmt.Errorf("%s: %s", "TimeMatch", err)
    }
    return TimeMatch(args[0], args[1])
}

// TimeMatch determines whether the current time is between
// startTime and endTime.
// You can use "_" to indicate that the parameter is ignored
func TimeMatch(startTime, endTime string) (bool, error) {
    now := time.Now()
    if startTime != "_" {
        if start, err := time.Parse("2006-01-02 15:04:05",
startTime); err != nil {
            return false, err
        } else if !now.After(start) {
            return false, nil
        }
    }

    if endTime != "_" {
        if end, err := time.Parse("2006-01-02 15:04:05",
endTime); err != nil {
            return false, err
        } else if now.After(end) {
            return false, nil
        }
    }
}
```

Conditional RoleManager with domains

model.conf

```
[request_definition]
r = sub, dom, obj, act

[policy_definition]
p = sub, dom, obj, act

[role_definition]
g = _, _, _, (_, _)

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub, r.dom) && r.dom == p.dom && r.obj == p.obj
&& r.act == p.act
```

policy.csv

```
p, alice, data1, read
p, data2_admin, data2, write
p, data3_admin, data3, read
p, data4_admin, data4, write
p, data5_admin, data5, read
p, data6_admin, data6, write
p, data7_admin, data7, read
p, data8_admin, data8, write

g, alice, data2_admin, domain2, 0000-01-01 00:00:00, 0000-01-02
```

Basic Usage

Add a conditional function for the role policy(g type policy) through `AddNamedDomainLinkConditionFunc`, and when enforcing is executed, the corresponding parameters will be automatically obtained and passed in the conditional function for checking. If the check passes, then the corresponding role policy(g type policy) is valid, otherwise it is invalid

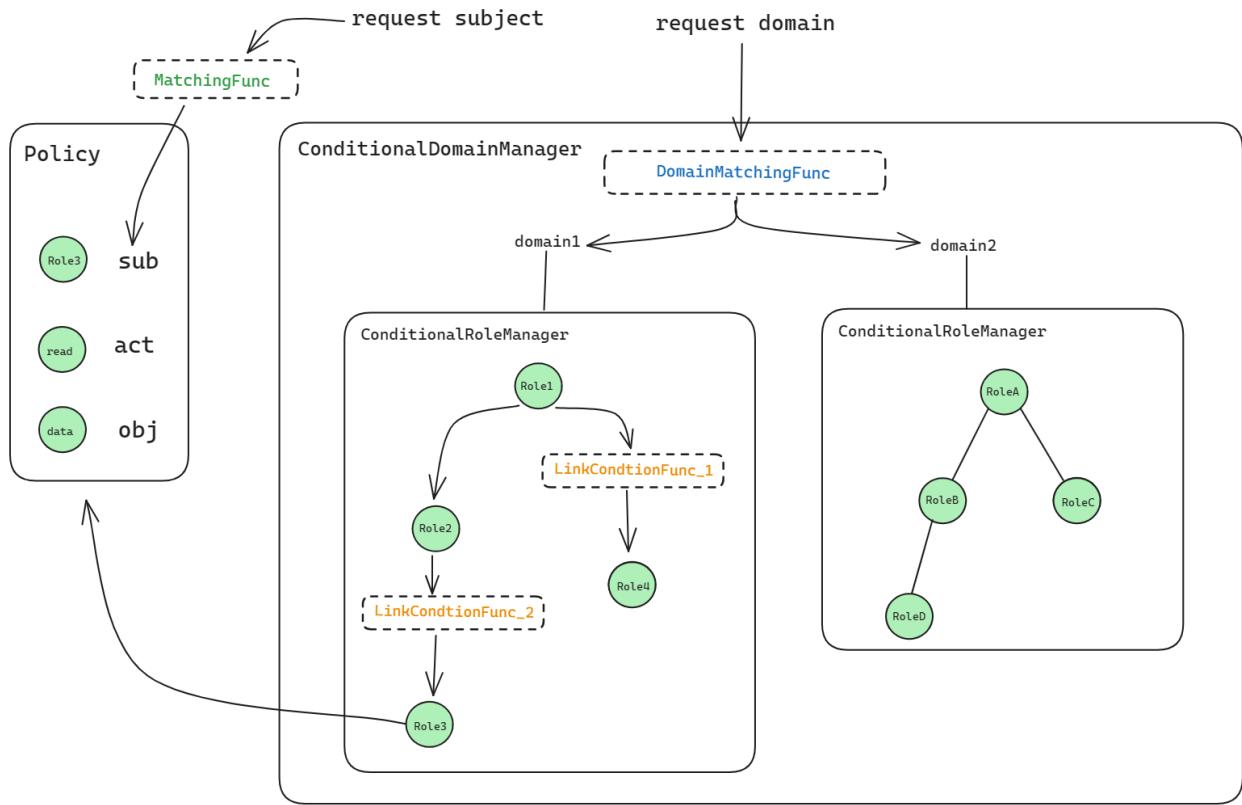
```
e.AddNamedDomainLinkConditionFunc("g", "alice", "data2_admin",
"domain2", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data3_admin",
"domain3", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data4_admin",
"domain4", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data5_admin",
"domain5", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data6_admin",
"domain6", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data7_admin",
"domain7", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data8_admin",
"domain8", util.TimeMatchFunc)

e.enforce("alice", "domain1", "data1", "read")          //
except: true
e.enforce("alice", "domain2", "data2", "write")         //
except: false
e.enforce("alice", "domain3", "data3", "read")          //
except: true
e.enforce("alice", "domain4", "data4", "write")         //
except: true
e.enforce("alice", "domain5", "data5", "read")          //
except: true
```

Custom condition functions

Like the basic `Conditional RoleManager`, custom functions are supported, and there is no difference in use.

Note that `DomainMatchingFunc`, `MatchingFunc`, and `LinkConditionFunc` are at different levels and are used in different situations.



Casbin RBAC vs. RBAC96

Casbin RBAC and RBAC96

In this document, we will compare Casbin RBAC with [RBAC96](#).

Casbin RBAC supports nearly all the features of RBAC96 and adds new features on top of that.

RBAC Version	Support Level	Description
RBAC0	Fully Supported	RBAC0 is the basic version of RBAC96. It clarifies the relationship between Users, Roles, and Permissions.
RBAC1	Fully Supported	RBAC1 adds role hierarchies on top of RBAC0. This means that if <code>alice</code> has <code>role1</code> , <code>role1</code> has <code>role2</code> , then <code>alice</code> will also have <code>role2</code> and inherit its permissions.
RBAC2	Mutually Exclusive Handling Supported (like this)	RBAC2 adds constraints on RBAC0. This allows RBAC2 to handle mutually exclusive policies. However, quantitative limits are not supported.
RBAC3	Mutually Exclusive	RBAC3 is a combination of RBAC1 and RBAC2. It supports role hierarchies and constraints found in

RBAC Version	Support Level	Description
	Handling Supported (like this)	RBAC1 and RBAC2. However, quantitative limits are not supported.

The Difference Between Casbin RBAC and RBAC96

1. In Casbin, the distinction between User and Role is not as clear as in RBAC96.

In Casbin, both the User and the Role are treated as strings. For example, consider the following policy file:

```
p, admin, book, read
p, alice, book, read
g, amber, admin
```

If you call the method `GetAllSubjects()` using an instance of the Casbin Enforcer:

```
e.GetAllSubjects()
```

the return value will be:

```
[admin alice]
```

This is because in Casbin, subjects include both Users and Roles.

However, if you call the method `GetAllRoles()`:

```
e.GetAllRoles()
```

the return value will be:

```
[admin]
```

From this, you can see that there is a distinction between Users and Roles in Casbin, but it is not as sharp as in RBAC96. Of course, you can add a prefix to your policies such as `user::alice` and `role::admin` to clarify their relationships.

2. Casbin RBAC provides more permissions than RBAC96.

RBAC96 defines only 7 permissions: read, write, append, execute, credit, debit, and inquiry.

However, in Casbin, we treat permissions as strings. This allows you to create permissions that better suit your needs.

3. Casbin RBAC supports domains.

In Casbin, you can perform authorizations based on domains. This feature makes your Access Control Model more flexible.

ReBAC

What is the ReBAC Model?

ReBAC (Relationship-Based Access Control) is a modern access control model that focuses on relationships between entities for permission management. Compared to traditional RBAC (Role-Based Access Control) or ABAC (Attribute-Based Access Control), ReBAC is better suited for systems with complex relationship networks, such as social networks, collaboration platforms, and multi-tenant systems.

In ReBAC, authorization decisions are based on relationships between entities, such as:

- Is the user the "owner" of the resource?
- Is the user a "friend" of the resource's "creator"?
- Does the user belong to an organization associated with the resource?
- Is the user an "admin" of a certain "project"?

These relationships are typically modeled as graph structures or paths.

ReBAC Support in Casbin

Casbin provides the following mechanisms to implement ReBAC:

- User-Resource-Role Relationships
- Resource-Type Relationships

A single policy rule can cover multiple users and multiple resources of the same type, enabling flexible and scalable permission control through relationship

combinations.

Casbin uses `.conf` files to define access control models. Below is an official ReBAC model example:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = role, obj_type, act

[role_definition]
g = _, _, _
g2 = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, r.obj, p.role) && g2(r.obj, p.obj_type) && r.act
== p.act
```

```
# Permission definition: The "collaborator" role can read files
# of type "doc"
p, collaborator, doc, read

# User-Resource-Role Relationship: alice is a collaborator of
# doc1
g, alice, doc1, collaborator

# Resource-Type Relationship: doc1 is of type "doc"
g2, doc1, doc
```

By checking whether a user has a specific role for a given resource and whether the resource belongs to a specified type, permissions are automatically derived

through role relationships + type relationships + permission definitions.

ABAC

What is the ABAC model?

ABAC stands for Attribute-Based Access Control. It allows you to control access by using the attributes (properties) of the subject, object, or action instead of using the string values themselves. You may have heard of a complicated ABAC access control language called XACML. Casbin's ABAC, on the other hand, is much simpler. In Casbin's ABAC, you can use structs or class instances instead of strings for model elements.

Let's take a look at the official ABAC example:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == r.obj.Owner
```

In the matcher, we use `r.obj.Owner` instead of `r.obj`. The `r.obj` passed in the `Enforce()` function will be a struct or class instance rather than a string. Casbin will use reflection to retrieve the `obj` member variable in that struct or class for you.

Here is a definition for the `r.obj` struct or class:

```
type testResource struct {
    Name string
    Owner string
}
```

If you want to pass parameters to the enforcer through JSON, you need to enable the function with `e.EnableAcceptJsonRequest(true)`.

For example:

```
e, _ := NewEnforcer("examples/abac_model.conf")
e.EnableAcceptJsonRequest(true)

data1Json := `{"Name": "data1", "Owner": "bob"}

ok, _ := e.Enforce("alice", data1Json, "read")
```

 NOTE

Enabling the function of accepting JSON parameters may result in a performance drop of 1.1 to 1.5 times.

How to use ABAC?

To use ABAC, you need to do two things:

1. Specify the attributes in the model matcher.
2. Pass in the struct or class instance for the element as an argument to Casbin's `Enforce()` function.

🔥 DANGER

Currently, only request elements like `r.sub`, `r.obj`, `r.act`, and so on support ABAC. You cannot use it on policy elements like `p.sub` because there is no way to define a struct or class in Casbin's policy.

💡 TIP

You can use multiple ABAC attributes in a matcher. For example: `m = r.sub.Domain == r.obj.Domain`.

💡 TIP

If you need to use a comma in a policy that conflicts with CSV's separator, you can escape it by surrounding the statement with quotation marks. For example, `"keyMatch("bob", r.sub.Role)"` will not be split.

Scaling the model for complex and large numbers of ABAC rules

The above implementation of the ABAC model is simple at its core. However, in many cases, the authorization system requires a complex and large number of ABAC rules. To accommodate this requirement, it is recommended to add the rules in the policy instead of the model. This can be done by introducing an `eval()` functional construct. Here is an example:

This is the definition of the `CONF` file used to define the ABAC model.

```
[request_definition]
```

In this example, `p.sub_rule` is a struct or class (user-defined type) that contains the necessary attributes to be used in the policy.

This is the policy that is used against the model for `Enforcement`. Now, you can use the object instance passed to `eval()` as a parameter to define certain ABAC constraints.

```
p, r.sub.Age > 18, /data1, read  
p, r.sub.Age < 60, /data2, write
```

PBAC

What is the PBAC model?

PBAC stands for Policy-Based Access Control. It is a flexible access control model that makes authorization decisions based on policies defined as rules. Unlike traditional access control models that rely on static roles or attributes, PBAC allows for dynamic, rule-based authorization that can evaluate complex conditions using the `eval()` function.

In PBAC, access decisions are made by evaluating policies that can include:

- **Dynamic Rules:** Policies that evaluate expressions at runtime
- **Complex Logic:** Support for boolean operations, comparisons, and attribute-based conditions
- **Contextual Information:** User attributes, resource properties, and environmental factors
- **Business Rules:** Domain-specific authorization logic that reflects organizational policies

PBAC Model Definition

Here's the PBAC model configuration:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub_rule, obj_rule, act
```

In this model:

- `p.sub_rule` contains the subject rule to be evaluated (e.g., `r.sub.Age >= 18`)
- `p.obj_rule` contains the object rule to be evaluated (e.g., `r.obj.Level >= 1`)
- `eval()` function evaluates the policy rules dynamically against the request

Policy and Request Examples

Basic Policy

Policy:

```
p, r.sub.Age >= 18, r.obj.Level >= 1, play
```

Request Examples:

```
{"Age":25}, {"Level":2}, play      # ALLOWED (Age >= 18 and Level  
>= 1)  
{"Age":16}, {"Level":2}, play      # DENIED (Age < 18)  
{"Age":20}, {"Level":0}, play      # DENIED (Level < 1)  
{"Age":25}, {"Level":2}, read      # DENIED (action doesn't match  
policy)
```

Complex Policy

Policy:

```
p, r.sub.Department == "IT" && r.sub.Level >= 3,  
r.obj.Confidential == false, read
```

Request Examples:

```
{"Department":"IT","Level":3}, {"Confidential":false}, read  
# ALLOWED  
{"Department":"IT","Level":2}, {"Confidential":false}, read  
# DENIED (Level < 3)  
{"Department":"HR","Level":3}, {"Confidential":false}, read  
# DENIED (Department != "IT")  
{"Department":"IT","Level":3}, {"Confidential":true}, read  
# DENIED (Confidential == true)
```

Code Example:

```
e, _ := NewEnforcer("examples/pbac_model.conf", "examples/  
pbac_policy.csv")  
  
// Enable JSON request support  
e.EnableAcceptJsonRequest(true)  
  
// Define subject and object with attributes  
subject := `{"Department": "IT", "Level": 3}`  
object := `{"Confidential": false}`  
action := "read"  
  
// Check permission  
ok, _ := e.Enforce(subject, object, action)  
if ok {  
    fmt.Println("Permission granted")  
} else {  
    fmt.Println("Permission denied")  
}
```


BLP

Overview

The Bell-LaPadula (BLP) model is a formal state transition model of computer security policy that describes a set of access control rules which use security labels on objects and clearances for subjects. It was developed by David Elliott Bell and Leonard J. LaPadula in 1973.

Model

```
[request_definition]
r = sub, sub_level, obj, obj_level, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = (r.act == "read" && r.sub_level >= r.obj_level) || (r.act
== "write" && r.sub_level <= r.obj_level)
```

Policy

BLP model typically doesn't require explicit policy rules as the access control is

determined by the security levels of subjects and objects. The matcher function implements the BLP rules:

- **No Read Up:** A subject cannot read an object with a higher security level
- **No Write Down:** A subject cannot write to an object with a lower security level

Examples

Request Examples

```
alice, 3, data1, 1, read      # alice (level 3) reads data1
(level 1) - ALLOWED
bob, 2, data2, 2, read       # bob (level 2) reads data2 (level
2) - ALLOWED
charlie, 1, data1, 1, read   # charlie (level 1) reads data1
(level 1) - ALLOWED
bob, 2, data3, 3, read       # bob (level 2) reads data3 (level
3) - DENIED (No Read Up)
charlie, 1, data2, 2, read   # charlie (level 1) reads data2
(level 2) - DENIED (No Read Up)

alice, 3, data3, 3, write    # alice (level 3) writes data3
(level 3) - ALLOWED
bob, 2, data3, 3, write     # bob (level 2) writes data3 (level
3) - ALLOWED
charlie, 1, data2, 2, write # charlie (level 1) writes data2
(level 2) - ALLOWED
alice, 3, data1, 1, write   # alice (level 3) writes data1
(level 1) - DENIED (No Write Down)
bob, 2, data1, 1, write     # bob (level 2) writes data1 (level
1) - DENIED (No Write Down)
```

Security Levels

In the BLP model, security levels are typically represented as integers where higher numbers indicate higher security levels:

- Level 1: Public/Unclassified
- Level 2: Confidential
- Level 3: Secret
- Level 4: Top Secret

Use Cases

BLP model is commonly used in:

- Military and government systems
- Financial institutions
- Healthcare systems
- Any environment requiring strict information flow control

Implementation Notes

- The model enforces mandatory access control (MAC)
- Security levels are assigned by system administrators
- Access decisions are based purely on security levels, not user identity
- The model prevents information leakage through read/write operations

Biba

Overview

The Biba Model (also known as Biba Integrity Model) is a formal state transition model of computer security policy developed by Kenneth J. Biba in 1975. It describes a set of access control rules designed to ensure data integrity. Unlike the Bell-LaPadula model which focuses on confidentiality, the Biba model is specifically designed to protect data integrity and prevent unauthorized modification of data.

Model

```
[request_definition]
r = sub, sub_level, obj, obj_level, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = (r.act == "read" && r.sub_level <= r.obj_level) || (r.act
== "write" && r.sub_level >= r.obj_level)
```

Policy

The Biba model typically doesn't require explicit policy rules as the access control is determined by the integrity levels of subjects and objects. The matcher function implements the Biba integrity rules:

- **No Read Down (Simple Integrity Property):** A subject cannot read an object with a lower integrity level
- **No Write Up (Star Integrity Property):** A subject cannot write to an object with a higher integrity level

Core Principles

The Biba model is characterized by the phrase "read up, write down", which is the inverse of the Bell-LaPadula model's "read down, write up". This approach ensures:

1. **Data Integrity Protection:** Prevents corruption of high-integrity data by lower-integrity sources
2. **Controlled Information Flow:** Ensures that information flows only from higher to lower integrity levels for writes
3. **Trust Preservation:** Maintains the trustworthiness of data at each integrity level

Examples

Request Examples

```
alice, 3, data1, 1, read    # alice (level 3) reads data1  
(level 1) - DENIED (No Read Down)  
bob, 2, data2, 2, read      # bob (level 2) reads data2 (level  
2) - ALLOWED  
charlie, 1, data1, 1, read   # charlie (level 1) reads data1  
(level 1) - ALLOWED  
bob, 2, data3, 3, read      # bob (level 2) reads data3 (level  
3) - ALLOWED  
charlie, 1, data2, 2, read   # charlie (level 1) reads data2  
(level 2) - ALLOWED  
  
alice, 3, data3, 3, write    # alice (level 3) writes data3  
(level 3) - ALLOWED  
bob, 2, data3, 3, write      # bob (level 2) writes data3 (level  
3) - DENIED (No Write Up)  
charlie, 1, data2, 2, write  # charlie (level 1) writes data2  
(level 2) - DENIED (No Write Up)  
alice, 3, data1, 1, write    # alice (level 3) writes data1  
(level 1) - ALLOWED  
bob, 2, data1, 1, write      # bob (level 2) writes data1 (level  
1) - ALLOWED
```

Integrity Levels

In the Biba model, integrity levels are typically represented as integers where higher numbers indicate higher integrity levels:

- Level 1: Low integrity (e.g., public data, user-generated content)
- Level 2: Medium integrity (e.g., verified data, trusted sources)
- Level 3: High integrity (e.g., system data, administrative content)
- Level 4: Critical integrity (e.g., security policies, system configuration)

Use Cases

The Biba model is commonly used in:

- Financial systems where data accuracy is paramount
- Healthcare records management
- Database systems requiring data integrity
- Any environment where preventing data corruption is more important than preventing data disclosure
- Systems where the accuracy and reliability of information is critical

Implementation Notes

- The model enforces mandatory access control (MAC) focused on integrity
- Integrity levels are assigned by system administrators based on data trustworthiness
- Access decisions are based on integrity levels rather than user identity
- The model prevents data corruption through controlled read/write operations
- Unlike Bell-LaPadula, most Biba applications use only a small number of integrity levels

Comparison with Bell-LaPadula

Aspect	Bell-LaPadula	Biba
Primary Focus	Confidentiality	Integrity
Read Rule	No Read Up	No Read Down
Write Rule	No Write Down	No Write Up
Phrase	"Read down, write up"	"Read up, write down"

LBAC

What is the LBAC model?

LBAC stands for Lattice-Based Access Control. It is a formal access control model that can be implemented in various ways. The example shown below demonstrates one possible implementation in Casbin that combines confidentiality and integrity controls.

Model Definition

Here is an example LBAC model implementation in Casbin:

```
[request_definition]
r = sub, subject_confidentiality, subject_integrity, obj,
object_confidentiality, object_integrity, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = (r.act == "read" && r.subject_confidentiality >=
r.object_confidentiality && r.subject_integrity >=
r.object_integrity) || (r.act == "write" &&
r.subject_confidentiality <= r.object_confidentiality &&
```

How it works

In this example implementation, the matcher function implements access control rules based on two security properties:

- **Confidentiality Control:** Prevents unauthorized disclosure of information
- **Integrity Control:** Prevents unauthorized modification of information

The access control decisions are made by comparing the security levels of subjects and objects in both dimensions.

Examples

Request Examples

The following are example requests to demonstrate how this LBAC implementation works:

```
# Normal read operations (ALLOWED)
admin, 5, 5, file_topsecret, 3, 3, read      # admin (conf:5,
int:5) reads file_topsecret (conf:3, int:3) - ALLOWED
manager, 4, 4, file_secret, 4, 2, read       # manager (conf:4,
int:4) reads file_secret (conf:4, int:2) - ALLOWED
staff, 3, 3, file_internal, 2, 3, read      # staff (conf:3,
int:3) reads file_internal (conf:2, int:3) - ALLOWED
guest, 2, 2, file_public, 2, 2, read        # guest (conf:2,
int:2) reads file_public (conf:2, int:2) - ALLOWED

# Read operation violations (DENIED)
staff, 3, 3, file_secret, 4, 2, read      # staff (conf:3,
```

Security Levels

In this example implementation, both confidentiality and integrity levels are represented as integers where higher numbers indicate higher security levels:

Confidentiality Levels

- Level 1: Public/Unclassified
- Level 2: Confidential
- Level 3: Secret
- Level 4: Top Secret

Integrity Levels

- Level 1: Low integrity (e.g., public data, user-generated content)
- Level 2: Medium integrity (e.g., verified data, trusted sources)
- Level 3: High integrity (e.g., system data, administrative content)
- Level 4: Critical integrity (e.g., security policies, system configuration)

Use Cases

This LBAC implementation example is suitable for:

- Multi-level security environments
- Applications where data protection and data accuracy are both important

Implementation Notes

- The model enforces mandatory access control (MAC) with dual security properties
- Security levels are assigned by system administrators
- Access decisions are based on both confidentiality and integrity levels
- The model prevents both information leakage and data corruption

Priority Model

Casbin supports loading policies with priority.

Load Policy with Implicit Priority

It's quite simple: the order determines the priority; policies that appear earlier have higher priority.

model.conf:

```
[policy_effect]
e = priority(p.eft) || deny
```

Load Policy with Explicit Priority

Also see: [casbin#550](#)

A smaller priority value indicates a higher priority. If there's a non-numerical character in the priority, it will be placed last instead of throwing an error.

⚠ TOKEN NAME CONVENTION

The conventionally used priority token name in the policy definition is "priority". To use a custom one, you need to invoke `e.SetFieldIndex()` and reload the policies (see the full example on [TestCustomizedFieldIndex](#)).

model.conf:

```
[policy_definition]
p = customized_priority, sub, obj, act, eft
```

Golang code example:

```
e, _ := NewEnforcer("./example/
priority_model_explicit_customized.conf",
                    "./example/
priority_policy_explicit_customized.csv")
// Due to the customized priority token, the enforcer
fails to handle the priority.
ok, err := e.Enforce("bob", "data2", "read") // the result
will be `true, nil`
// Set PriorityIndex and reload
e.SetFieldIndex("p", constant.PriorityIndex, 0)
err := e.LoadPolicy()
if err != nil {
    log.Fatalf("LoadPolicy: %v", err)
}
ok, err := e.Enforce("bob", "data2", "read") // the result
will be `false, nil`
```

Currently, explicit priority only supports `AddPolicy` & `AddPolicies`. If `UpdatePolicy` has been called, you shouldn't change the priority attribute.

model.conf:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = priority, sub, obj, act, eft
```

policy.csv

```
p, 10, data1_deny_group, data1, read, deny
p, 10, data1_deny_group, data1, write, deny
p, 10, data2_allow_group, data2, read, allow
p, 10, data2_allow_group, data2, write, allow

p, 1, alice, data1, write, allow
p, 1, alice, data1, read, allow
p, 1, bob, data2, read, deny

g, bob, data2_allow_group
g, alice, data1_deny_group
```

request:

```
alice, data1, write --> true // because `p, 1, alice, data1,
write, allow` has the highest priority
bob, data2, read --> false
bob, data2, write --> true // because bob has the role of
`data2_allow_group` which has the right to write data2, and
there's no deny policy with higher priority
```

Load Policy with Priority Based on Role and User Hierarchy

The inherited structure of roles and users can only be multiple trees, not graphs. If a user has multiple roles, you have to make sure the user has the same level in different trees. If two roles have the same level, the policy (associated with the role) that appeared earlier has higher priority. For more details, also see

[casbin#833](#) and [casbin#831](#).

model.conf:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act, eft

[role_definition]
g = _, _

[policy_effect]
e = subjectPriority(p.eft) || deny

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

policy.csv

```
p, root, data1, read, deny
p, admin, data1, read, deny

p, editor, data1, read, deny
p, subscriber, data1, read, deny

p, jane, data1, read, allow
p, alice, data1, read, allow

g, admin, root

g, editor, admin
g, subscriber, admin
```

Request:

```
jane, data1, read --> true // because jane is at the bottom,  
her priority is higher than that of editor, admin, and root  
alice, data1, read --> true
```

The role hierarchy looks like this:

```
role: root  
  └ role: admin  
    └ role editor  
      └ user: jane  
    └ role: subscriber  
      └ user: alice
```

The priority automatically looks like this:

```
role: root          # auto priority: 30  
  └ role: admin    # auto priority: 20  
    └ role: editor  # auto priority: 10  
    └ role: subscriber # auto priority: 10
```

Usage Control (UCON)

Overview

The Usage Control (UCON) model is an extension of traditional access control models that integrates authorizations, obligations, and conditions. Developed by Sandhu and Park, UCON provides a comprehensive framework for controlling access to and usage of digital resources in distributed systems. Unlike traditional access control models that focus only on authorization decisions at access time, UCON introduces continuous enforcement and attribute mutability.

Model & Policy

Note: Casbin does not directly support UCON in its core library. Instead, UCON functionality is provided through the extension library [casbin-ucon](#), which adds session-based access control with conditions, obligations, and continuous monitoring capabilities. It can use the same models and policies supported by the Casbin core.

model

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))
```

policy

```
p, alice, document1, read
```

Note: In UCON, Conditions and Obligations are not written in policy.csv; they are added and managed in code through the casbin-ucon API.

Core Components

The UCON model consists of three core components:

1. **Authorizations (A)**: Traditional access control rules based on subject and object attributes
2. **oBligations (B)**: Requirements that must be fulfilled by subjects before or during resource usage
3. **Conditions (C)**: Environmental or system factors independent of subjects and objects

Key Features

1. **Continuity**: Unlike traditional models that check permissions only at access time, UCON enables ongoing enforcement during the entire usage session
2. **Mutability**: Subject and object attributes can be updated as a consequence of usage
3. **Pre-decisions**: Authorizations, obligations, and conditions checked before access
4. **Ongoing-decisions**: Continuous checking during resource usage
5. **Post-decisions**: Updates to attributes after usage completion

Use Cases

The UCON model is particularly useful in:

- Digital Rights Management (DRM) systems
- Healthcare information systems
- Cloud computing environments
- IoT device access control
- Data sharing in collaborative environments
- Financial systems with complex compliance requirements

Implementation with casbin-ucon

To implement UCON in your Casbin-based application, you need to use the [casbin-ucon](#) extension library:

```
go get github.com/casbin/casbin-ucon
```

```
// Import the required packages
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin-ucon"
    "fmt"
    "time"
)

func main() {
    // Create standard Casbin enforcer
```

Implementation Notes

- UCON encompasses traditional access control models like MAC, DAC, and RBAC
- Implementation requires a reference monitor capable of continuous monitoring
- Attribute updates (mutability) should be handled atomically to maintain consistency
- Obligations may require integration with external monitoring systems
- Conditions evaluation may depend on environmental factors outside the access control system
- When using casbin-ucon, you need to properly manage session lifecycle and monitoring

Comparison with Traditional Models

Aspect	Traditional Access Control	UCON
Decision Time	Pre-access only	Pre, ongoing, and post access
Attribute Mutability	Static	Dynamic (can change during usage)
Decision Factors	Authorizations only	Authorizations, obligations, and conditions

Aspect	Traditional Access Control	UCON
Enforcement	One-time check	Continuous monitoring
Revocation	Explicit	Can be automatic based on attribute changes

References

For complete API documentation, detailed usage, and latest updates about casbin-ucon, please refer to [casbin-ucon](#) .

Super Admin

The Super Admin is the administrator of the entire system. It can be used in models such as RBAC, ABAC, and RBAC with domains. The detailed example is as follows:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act || r.sub
== "root"
```

This example illustrates that, with the defined `request_definition`, `policy_definition`, `policy_effect`, and `matchers`, Casbin determines whether the request can match the policy. One important aspect is checking if the `sub` is root. If the judgment is correct, authorization is granted, and the user has permission to perform all actions.

Similar to the root user in Linux systems, being authorized as root grants access to all files and settings. If we want a `sub` to have full access to the entire system, we can assign it the role of Super Admin, granting the `sub` permission to perform all actions.

Storage

Model Storage

Model storage

Policy Storage

Policy Storage

Policy Subset Loading

Loading filtered policies

Model Storage

Unlike the policy, the model can only be loaded, it cannot be saved. We believe that the model is not a dynamic component and should not be modified at runtime, so we have not implemented an API to save the model into storage.

However, there is good news. We provide three equivalent ways to load a model, either statically or dynamically:

Load model from .CONF file

This is the most common way to use Casbin. It is easy to understand for beginners and convenient for sharing when you need help from the Casbin team.

The content of the `.CONF` file [examples/rbac_model.conf](#) is as follows:

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

Then you can load the model file as follows:

```
e := casbin.NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

Load model from code

The model can be initialized dynamically from code instead of using a `.CONF` file.

Here's an example for the RBAC model:

```
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin/v2/model"
    "github.com/casbin/casbin/v2/persist/file-adapter"
)

// Initialize the model from Go code.
m := model.NewModel()
m.AddDef("r", "r", "sub, obj, act")
m.AddDef("p", "p", "sub, obj, act")
m.AddDef("g", "g", "_, _")
m.AddDef("e", "e", "some(where (p.eft == allow))")
m.AddDef("m", "m", "g(r.sub, p.sub) && r.obj == p.obj && r.act
== p.act")

// Load the policy rules from the .CSV file adapter.
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")

// Create the enforcer.
e := casbin.NewEnforcer(m, a)
```

Load model from string

Alternatively, you can load the entire model text from a multi-line string. The advantage of this approach is that you do not need to maintain a model file.

```
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin/v2/model"
)

// Initialize the model from a string.
text :=
`

[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
`

m, _ := model.NewModelFromString(text)

// Load the policy rules from the .CSV file adapter.
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")
```


Policy Storage

In Casbin, the policy storage is implemented as an [adapter](#).

Loading policy from a .CSV file

This is the most common way to use Casbin. It is easy to understand for beginners and convenient for sharing when you ask the Casbin team for help.

The content of the `.csv` file [examples/rbac_policy.csv](#) is as follows:

```
p, alice, data1, read
p, bob, data2, write
p, data2_admin, data2, read
p, data2_admin, data2, write
g, alice, data2_admin
```

NOTE

If your file contains commas, you should wrap them in double quotes. For example:

```
p, alice, "data1,data2", read      --correct
p, alice, data1,data2, read        --incorrect (the whole
phrase "data1,data2" should be wrapped in double quotes)
```

If your file contains commas and double quotes, you should enclose the field in double quotes and double any embedded double quotes.

```
p, alice, data, "r.act in (\"get\", \"post\")"      --
correct
p, alice, data, "r.act in (\"get\", \"post\")"
incorrect (you should use "" to escape "")
```

Related issue: [casbin#886](#)

Adapter API

Method	Type	Description
LoadPolicy()	basic	Load all policy rules from the storage
SavePolicy()	basic	Save all policy rules to the storage
AddPolicy()	optional	Add a policy rule to the storage
RemovePolicy()	optional	Remove a policy rule from the storage
RemoveFilteredPolicy()	optional	Remove policy rules that match the filter from the storage

Database Storage Format

Your policy file

```
p, data2_admin, data2, read  
p, data2_admin, data2, write  
g, alice, admin
```

Corresponding database structure (such as MySQL)

id	ptype	v0	v1	v2	v3	v4	v5
1	p	data2_admin	data2	read			
2	p	data2_admin	data2	write			
3	g	alice	admin				

Meaning of each column

- `id`: The primary key in the database. It does not exist as part of the `casbin policy`. The way it is generated depends on the specific adapter.
- `ptype`: It corresponds to `p`, `g`, `g2`, etc.
- `v0 - v5`: The column names have no specific meaning and correspond to the values in the `policy csv` from left to right. The number of columns depends on how many you define yourself. In theory, there can be an infinite number of columns, but generally only 6 columns are implemented in the adapter. If this is not enough for you, please submit an issue to the corresponding adapter repository.

Adapter Details

For more details about the use of the adapter API and database table structure

design, please visit: </docs/adapters>

Policy Subset Loading

Some adapters support filtered policy management. This means that the policy loaded by Casbin is a subset of the policy stored in the database based on a given filter. This allows for efficient policy enforcement in large, multi-tenant environments where parsing the entire policy becomes a performance bottleneck.

To use filtered policies with a supported adapter, simply call the `LoadFilteredPolicy` method. The valid format for the filter parameter depends on the adapter used. To prevent accidental data loss, the `SavePolicy` method is disabled when a filtered policy is loaded.

For example, the following code snippet uses the built-in filtered file adapter and the RBAC model with domains. In this case, the filter limits the policy to a single domain. Any policy lines for domains other than `"domain1"` are omitted from the loaded policy:

```
import (
    "github.com/casbin/casbin/v2"
    fileadapter "github.com/casbin/casbin/v2/persist/file-
adapter"
)

enforcer, _ := casbin.NewEnforcer()

adapter := fileadapter.NewFilteredAdapter("examples/
rbac_with_domains_policy.csv")
enforcer.InitWithAdapter("examples/
rbac_with_domains_model.conf", adapter)

filter := &fileadapter.Filter{
    P: []string{"", "domain1"},
```

There is another method that supports the subset loading feature:

`LoadIncrementalFilteredPolicy`. `LoadIncrementalFilteredPolicy` is similar to `LoadFilteredPolicy`, but it does not clear the previously loaded policy. It only appends the filtered policy to the existing policy.

Scenarios

Data Permissions

Solutions for Data Permissions

Menu Permissions

Example for Menu Permissions

Data Permissions

We have two solutions for data permissions (filtering): using implicit assignment APIs or using the `BatchEnforce()` API.

1. Query Implicit Roles or Permissions

When a user inherits a role or permission via an RBAC hierarchy instead of being directly assigned them in a policy rule, we refer to this type of assignment as "implicit". To query such implicit relations, you need to use the following two APIs: `GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()`, instead of `GetRolesForUser()` and `GetPermissionsForUser()`. For more details, please refer to [this GitHub issue](#).

2. Use `BatchEnforce()`

`BatchEnforce()` enforces each request and returns the results in a boolean array.

For example:

Go Node.js Java

```
boolArray, err := e.BatchEnforce(requests)

const boolArray = await e.batchEnforce(requests);

List<Boolean> boolArray = e.batchEnforce(requests);
```

Menu Permissions

We begin by introducing a Spring Boot example featuring a menu system. This example leverages jCasbin to manage menu permissions. Ultimately, it aims to abstract a middleware, specifically for menu permissions, which could be extended to other languages supported by Casbin, such as Go and Python.

1. Configuration Files

You need to set up role and permission management in the `policy.csv` file, along with the parent-child relationships between menu items. For more details, please refer to [this GitHub repo](#).

1.1 Overview

Using `policy.csv`, you can flexibly configure role permissions and menu structures for fine-grained access control. This configuration file defines access permissions for different roles on various menu items, associations between users and roles, and the hierarchical relationships between menu items.

1.2 Permission Definitions (Policies)

- **Policy Rules:** Policies are defined with a `p` prefix, specifying roles (`sub`) and their permissions (`act`) on menu items (`obj`), along with the rule's effect (`efc`), where `allow` indicates permission is granted, and `deny` indicates it is denied.

Examples:

- `p, ROLE_ROOT, SystemMenu, read, allow` means the `ROLE_ROOT` role has read access to the `SystemMenu` menu item.

- `p, ROLE_ROOT, UserMenu, read, deny` means the `ROLE_ROOT` role is denied read access to the `UserMenu` menu item.

1.3 Roles and User Associations

- **Role Inheritance:** User-role relationships and role hierarchies are defined with a `g` prefix. This allows users to inherit permissions from one or multiple roles.

Examples:

- `g, user, ROLE_USER` means the user `user` is assigned the `ROLE_USER` role.
- `g, ROLE_ADMIN, ROLE_USER` means `ROLE_ADMIN` inherits permissions from `ROLE_USER`.

1.4 Menu Item Hierarchy

- **Menu Relationships:** Parent-child relationships between menu items are defined with a `g2` prefix, aiding in the construction of a menu's structure.

Examples:

- `g2, UserSubMenu_allow, UserMenu` indicates `UserSubMenu_allow` is a submenu of `UserMenu`.
- `g2, (NULL), SystemMenu` indicates `SystemMenu` has no submenu item, meaning it is a top-level menu item.

1.5 Menu Permission Inheritance and Default Rules

When managing menu permissions with jCasbin, the permission relationship between parent and child menus follows specific inheritance rules, with two important default rules:

Inheritance of Parent Menu Permissions:

If a parent menu is explicitly granted `allow` permission, all its submenus also default to `allow` permission unless specifically marked as `deny`. This means once a parent menu is accessible, its submenus are also accessible by default.

Handling Parent Menus Without Direct Permission Settings:

If a parent menu has no direct permission settings (neither explicitly allowed nor denied) but has at least one submenu explicitly granted `allow` permission, then the parent menu is implicitly considered to have `allow` permission. This ensures users can navigate to these submenus.

1.6 Special Permission Inheritance Rules

Regarding the inheritance of permissions between roles, especially in scenarios involving `deny` permissions, the following rules must be followed to ensure system security and precise control of permissions:

Distinction Between Explicit and Default Denials:

If a role, such as `ROLE_ADMIN`, is explicitly denied access to a menu item, such as `AdminSubMenu_deny` (marked as `deny`), then even if this role is inherited by another role (e.g., `ROLE_ROOT`), the inheriting role is not permitted access to the denied menu item. This ensures explicit security policies are not bypassed due to role inheritance.

Inheritance of Default Denial Permissions:

Conversely, if a role's denial of access to a menu item (e.g., `UserSubMenu_deny`) is default (not explicitly marked as `deny`, but because it was not explicitly granted `allow`), then when this role is inherited by another role (e.g., `ROLE_ADMIN`), the inheriting role may override the default `deny` status, allowing access to these menu items.

1.7 Example Description

policy:

```
p, ROLE_ROOT, SystemMenu, read, allow
p, ROLE_ROOT, AdminMenu, read, allow
p, ROLE_ROOT, UserMenu, read, deny
p, ROLE_ADMIN, UserMenu, read, allow
p, ROLE_ADMIN, AdminMenu, read, allow
p, ROLE_ADMIN, AdminSubMenu_deny, read, deny
p, ROLE_USER, UserSubMenu_allow, read, allow

g, user, ROLE_USER
g, admin, ROLE_ADMIN
g, root, ROLE_ROOT
g, ROLE_ADMIN, ROLE_USER

g2, UserSubMenu_allow, UserMenu
g2, UserSubMenu_deny, UserMenu
g2, UserSubSubMenu, UserSubMenu_allow
g2, AdminSubMenu_allow, AdminMenu
g2, AdminSubMenu_deny, AdminMenu
g2, (NULL), SystemMenu
```

MenuName	ROLE_ROOT	ROLE_ADMIN	ROLE_USER
SystemMenu	✓	✗	✗
UserMenu	✗	✓	✗
UserSubMenu_allow	✗	✓	✓

MenuName	ROLE_ROOT	ROLE_ADMIN	ROLE_USER
UserSubSubMenu	✗	✓	✓
UserSubMenu_deny	✗	✓	✗
AdminMenu	✓	✓	✗
AdminSubMenu_allow	✓	✓	✗
AdminSubMenu_deny	✓	✗	✗

2. Menu Permission Control

The list of all menu items accessible by a given username can be identified through the `findAccessibleMenus()` function available in the [MenuService](#). To check whether a specific user has the rights to access a designated menu item, the `checkMenuAccess()` method can be utilized. This approach ensures that menu permissions are effectively controlled, leveraging jCasbin's capabilities to manage access rights efficiently.

Extensions

Enforcers

The Enforcer is the main structure in Casbin that acts as an interface for users to perform operations on policy rules and models.

Adapters

Supported adapters and usage

Watchers

Maintaining consistency between multiple Casbin enforcer instances

Dispatchers

Dispatchers provide a way to synchronize incremental changes of policy.

Role Managers

The role manager is used to manage the RBAC role hierarchy in Casbin.

Middlewares

Casbin middlewares

 **GraphQL Middlewares**

Authorization for GraphQL endpoints

 **Cloud Native Middlewares**

Cloud Native Middlewares

Enforcers

The `Enforcer` is the main structure in Casbin. It acts as an interface for users to perform operations on policy rules and models.

Supported Enforcers

A complete list of Casbin enforcers is provided below. Any 3rd-party contribution on a new enforcer is welcomed. Please inform us, and we will add it to this list :)

[Go](#)

[Python](#)

Enforcer	Author	Description
Enforcer	Casbin	The <code>Enforcer</code> is the basic structure for users to interact with Casbin policies and models. You can find more details about the <code>Enforcer</code> API here .
CachedEnforcer	Casbin	The <code>CachedEnforcer</code> is based on the <code>Enforcer</code> and supports caching the evaluation result of a request in memory using a map. It provides the ability to clear caches within a specified expiration time. Moreover, it guarantees thread safety with a Read-Write lock. You can use <code>EnableCache</code> to enable caching of evaluation results (default is enabled).

Enforcer	Author	Description
		The other API methods of <code>CachedEnforcer</code> are the same as <code>Enforcer</code> .

<code>DistributedEnforcer</code>	Casbin	The <code>DistributedEnforcer</code> supports multiple instances in distributed clusters. It wraps the <code>SyncedEnforcer</code> for the dispatcher. You can find more details about the dispatcher here .
<code>SyncedEnforcer</code>	Casbin	The <code>SyncedEnforcer</code> is based on the <code>Enforcer</code> and provides synchronized access. It is thread-safe.
<code>SyncedCachedEnforcer</code>	Casbin	The <code>SyncedCachedEnforcer</code> wraps the <code>Enforcer</code> and provides decision sync cache.

Enforcer	Author	Description
<code>Enforcer</code>	Casbin	The <code>Enforcer</code> is the basic structure for users to interact with Casbin policies and models. You can find more details about the <code>Enforcer</code> API here .
<code>DistributedEnforcer</code>	Casbin	The <code>DistributedEnforcer</code> supports multiple instances in distributed clusters. It wraps the <code>SyncedEnforcer</code> for the

Enforcer	Author	Description
		dispatcher. You can find more details about the dispatcher here .
SyncedEnforcer	Casbin	The <code>SyncedEnforcer</code> is based on the <code>Enforcer</code> and provides synchronized access. It is thread-safe.
AsyncEnforcer	Casbin	The <code>AsyncEnforcer</code> provides async API.
FastEnforcer	Casbin	The <code>FastEnforcer</code> uses a new model which is 50x faster than the normal model. You can find more here

Adapters

In Casbin, the policy storage is implemented as an adapter (aka middleware for Casbin). A Casbin user can use an adapter to load policy rules from a storage (aka `LoadPolicy()`), or save policy rules to it (aka `SavePolicy()`). To keep light-weight, we don't put adapter code in the main library.

Supported adapters

A complete list of Casbin adapters is provided as below. Any 3rd-party contribution on a new adapter is welcomed, please inform us and we will put it in this list:

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Ruby](#) [Swift](#) [Lua](#)

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For <code>.CSV (Comma-Separated Values)</code> files
Filtered File Adapter (built-in)	File	@faceless-saint	✗	For <code>.CSV (Comma-Separated Values)</code> files with policy subset loading support
SQL Adapter	SQL	@Blank-Xu	✓	MySQL, PostgreSQL, SQL Server, SQLite3 are supported in <code>master</code> branch and Oracle is supported in <code>oracle</code> branch by <code>database/sql</code>
Xorm Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, TiDB, SQLite, SQL Server, Oracle are supported by Xorm
GORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3, SQL Server are supported by GORM

Adapter	Type	Author	AutoSave	Description
GORM Adapter Ex	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3, SQL Server are supported by GORM
Ent Adapter	ORM	Casbin	✓	MySQL, MariaDB, PostgreSQL, SQLite, Gremlin-based graph databases are supported by ent ORM
Beego ORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3 are supported by Beego ORM
SQLX Adapter	ORM	@memwey	✓	MySQL, PostgreSQL, SQLite, Oracle are supported by SQLX
Sqlx Adapter	ORM	@Blank-Xu	✓	MySQL, PostgreSQL, SQL Server, SQLite3 are supported in <code>master</code> branch and Oracle is supported in <code>oracle</code> branch by sqlx
GF ORM Adapter	ORM	@vance-liu	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
GoFrame ORM Adapter	ORM	@kotlin2018	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
gf-adapter	ORM	@zcyc	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Gdb Adapter	ORM	@jxo-me	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM

Adapter	Type	Author	AutoSave	Description
GoFrame V2 Adapter	ORM	@hailaz	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Bun Adapter	ORM	@JunNishimura	✓	MySQL, SQLite, PostgreSQL, SQL Server are supported by Bun ORM
Filtered PostgreSQL Adapter	SQL	Casbin	✓	For PostgreSQL
Filtered pgx Adapter	SQL	@pckhoi	✓	PostgreSQL is supported by pgx
Pgx Adapter	SQL	@gtoxlili	✓	PostgreSQL is supported by pgx, supports customizable column count
PostgreSQL Adapter	SQL	@cychiuae	✓	For PostgreSQL
RQLite Adapter	SQL	EDOMO Systems	✓	For RQLite
MongoDB Adapter	NoSQL	Casbin	✓	For MongoDB based on MongoDB Go Driver
RethinkDB Adapter	NoSQL	@adityapandey9	✓	For RethinkDB
Cassandra Adapter	NoSQL	Casbin	✗	For Apache Cassandra DB
DynamoDB Adapter	NoSQL	HOOQ	✗	For Amazon DynamoDB
Dynacasbin	NoSQL	NewbMiao	✓	For Amazon DynamoDB

Adapter	Type	Author	AutoSave	Description
ArangoDB Adapter	NoSQL	@adamwasila	✓	For ArangoDB
Amazon S3 Adapter	Cloud	Soluto	✗	For Minio and Amazon S3
Go CDK Adapter	Cloud	@bartventer	✓	Adapter based on Go Cloud Dev Kit that supports: Amazon DynamoDB, Azure CosmosDB, GCP Firestore, MongoDB, In-Memory
Azure Cosmos DB Adapter	Cloud	@spacycoder	✓	For Microsoft Azure Cosmos DB
GCP Firestore Adapter	Cloud	@reedom	✗	For Google Cloud Platform Firestore
GCP Cloud Storage Adapter	Cloud	qurami	✗	For Google Cloud Platform Cloud Storage
GCP Cloud Spanner Adapter	Cloud	@flowerinthenight	✓	For Google Cloud Platform Cloud Spanner
Consul Adapter	KV store	@ankitm123	✗	For HashiCorp Consul
Redis Adapter (Redigo)	KV store	Casbin	✓	For Redis
Redis Adapter	KV store	@mlsen	✓	For Redis

Adapter	Type	Author	AutoSave	Description
(go-redis)				
Etcd Adapter	KV store	@sebastianliu	✗	For etcd
BoltDB Adapter	KV store	@speza	✓	For Bolt
Bolt Adapter	KV store	@wirepair	✗	For Bolt
BadgerDB Adapter	KV store	@inits	✓	For BadgerDB
Protobuf Adapter	Stream	Casbin	✗	For Google Protocol Buffers
JSON Adapter	String	Casbin	✗	For JSON
String Adapter	String	@qiangmzsx	✗	For String
HTTP File Adapter	HTTP	@h4ckedneko	✗	For http.FileSystem
FileSystem Adapter	File	@naucon	✗	For fs.FS and embed.FS
Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
JDBC	JDBC	Casbin	✓	MySQL, Oracle, PostgreSQL, DB2, Sybase,

Adapter	Type	Author	AutoSave	Description	
Adapter				SQL Server are supported by JDBC	
Hibernate Adapter	ORM	Casbin	<input checked="" type="checkbox"/>	Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL are supported by Hibernate	
MyBatis Adapter	ORM	Casbin	<input checked="" type="checkbox"/>	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server (the same as JDBC) are supported by MyBatis 3	
Hutool Adapter	ORM	@mapleafgo	<input checked="" type="checkbox"/>	MySQL, Oracle, PostgreSQL, SQLite are supported by Hutool	
MongoDB Adapter	NoSQL	Casbin	<input checked="" type="checkbox"/>	MongoDB is supported by mongodb-driver-sync	
DynamoDB Adapter	NoSQL	Casbin	<input type="checkbox"/>	For Amazon DynamoDB	
Redis Adapter	KV store	Casbin	<input checked="" type="checkbox"/>	For Redis	
Adapter	Type	Author		AutoSave	Description
File Adapter (built-in)	File	Casbin		<input type="checkbox"/>	For .CSV (Comma-Separated Values) files
Filtered File Adapter (built-in)	File	Casbin		<input type="checkbox"/>	For .CSV (Comma-Separated Values) files with policy subset loading support
String Adapter (built-in)	String	@calebfaruki		<input type="checkbox"/>	For String
Basic Adapter	Native ORM	Casbin		<input checked="" type="checkbox"/>	pg, mysql, mysql2, sqlite3, oracledb, mssql are

Adapter	Type	Author	AutoSave	Description
				supported by the adapter itself
Sequelize Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by Sequelize
TypeORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL, MongoDB are supported by TypeORM
Prisma Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, AWS Aurora, Azure SQL are supported by Prisma
Knex Adapter	ORM	knex	✓	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle are supported by Knex.js
Objection.js Adapter	ORM	@willsoto	✓	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle are supported by Objection.js
MikroORM Adapter	ORM	@baisheng	✓	MongoDB, MySQL, MariaDB, PostgreSQL, SQLite are supported by MikroORM
Node PostgreSQL Native	SQL	@touchifyapp	✓	PostgreSQL adapter with advanced policy subset loading support and

Adapter	Type	Author	AutoSave	Description
Adapter				improved performances built with node-postgres .
Mongoose Adapter	NoSQL	elastic.io and Casbin	<input checked="" type="checkbox"/>	MongoDB is supported by Mongoose
Mongoose Adapter (No-Transaction)	NoSQL	minhducck	<input checked="" type="checkbox"/>	MongoDB is supported by Mongoose
Node MongoDB Native Adapter	NoSQL	NathanBhanji	<input checked="" type="checkbox"/>	For Node MongoDB Native
Node MongoDB Native Adapter	NoSQL	@juicyleff	<input checked="" type="checkbox"/>	For Node MongoDB Native
DynamoDB Adapter	NoSQL	@fospitia	<input checked="" type="checkbox"/>	For Amazon DynamoDB
Couchbase Adapter	NoSQL	@MarkMYoung	<input checked="" type="checkbox"/>	For Couchbase
Redis Adapter	KV store	Casbin	<input type="checkbox"/>	For Redis
Redis Adapter	KV store	@NandaKishorJeripothula	<input type="checkbox"/>	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Database Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by techone/database
Zend Db Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Oracle, IBM DB2, Microsoft SQL Server, Other PDO Driver are supported by zend-db
Doctrine DBAL Adapter (Recommend)	ORM	Casbin	✓	Powerful PHP database abstraction layer (DBAL) with many features for database schema introspection and management.
Medoo Adapter	ORM	Casbin	✓	Medoo is a lightweight PHP Database Framework to Accelerate Development, supports all SQL databases, including MySQL , MSSQL , SQLite , MariaDB , PostgreSQL , Sybase , Oracle and more.
Laminas-db Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PDO, etc. are supported by laminas-db
Zend-db Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PDO, etc. are supported by zend-db
ThinkORM Adapter (ThinkPHP)	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Oracle, Microsoft SQL Server, MongoDB are supported by ThinkORM
Redis Adapter	KV store	@nsnake	✗	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Django ORM Adapter	ORM	Casbin	✓	PostgreSQL, MariaDB, MySQL, Oracle, SQLite, IBM DB2, Microsoft SQL Server, Firebird, ODBC are supported by Django ORM
SQLObject Adapter	ORM	Casbin	✓	PostgreSQL, MySQL, SQLite, Microsoft SQL Server, Firebird, Sybase, MAX DB, pyfirebirdsql are supported by SQLObject
SQLAlchemy Adapter	ORM	Casbin	✓	PostgreSQL, MySQL, SQLite, Oracle,

Adapter	Type	Author	AutoSave	Description
				Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
Async SQLAlchemy Adapter	ORM	Casbin	✓	PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
Async Databases Adapter	ORM	Casbin	✓	PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server, Firebird, Sybase are supported by Databases
Peewee Adapter	ORM	@shblhy	✓	PostgreSQL, MySQL, SQLite are supported by Peewee

Adapter	Type	Author	AutoSave	Description
MongoEngine Adapter	ORM	@zhangbailong945	✗	MongoDB is supported by MongoEngine
Pony ORM Adapter	ORM	@drorvinkler	✓	MySQL, PostgreSQL, SQLite, Oracle, CockroachDB are supported by Pony ORM
Tortoise ORM Adapter	ORM	@thearchitector	✓	PostgreSQL (>=9.4), MySQL, MariaDB, and SQLite are supported by Tortoise ORM
Async Ormar Adapter	ORM	@shepilov-vladislav	✓	PostgreSQL, MySQL, SQLite are supported by Ormar
SQLModel Adapter	ORM	@shepilov-vladislav	✓	PostgreSQL, MySQL, SQLite are supported by SQLModel
Couchbase Adapter	NoSQL	ScienceLogic	✓ (without <code>remove_filtered_policy()</code>)	For Couchbase

Adapter	Type	Author	AutoSave	Description
DynamoDB Adapter	NoSQL	@abqadeer	✓	For DynamoDB
Pymongo Adapter	NoSQL	Casbin	✗	MongoDB is supported by Pymongo
Redis Adapter	KV store	Casbin	✓	For Redis
GCP Firebase Adapter	Cloud	@devrushi41	✓	For Google Cloud Platform Firebase

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
EF Adapter	ORM	Casbin	✗	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, DB2, etc. are supported by Entity Framework 6
EFCore Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, DB2, etc. are supported by Entity Framework Core
Linq2DB Adapter	ORM	@Tirael	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, Access, Firebird, Sybase, etc. are supported by linq2db
Azure	Cloud	@sagarkhandelwal	✓	For Microsoft Azure Cosmos DB

Adapter	Type	Author	AutoSave	Description	
Cosmos DB Adapter					
Adapter		Type	Author	AutoSave	Description
File Adapter (built-in)		File	Casbin	✗	For .CSV (Comma-Separated Values) files
Diesel Adapter	ORM	Casbin	✓	SQLite, PostgreSQL, MySQL are supported by Diesel	
Sqlx Adapter	ORM	Casbin	✓	PostgreSQL, MySQL are supported by Sqlx with fully asynchronous operation	
SeaORM Adapter	ORM	@lingdu1234	✓	PostgreSQL, MySQL, SQLite are supported by SeaORM with fully asynchronous operation	
SeaORM Adapter	ORM	@ZihanType	✓	PostgreSQL, MySQL, SQLite are supported by SeaORM with fully asynchronous operation	
Rbatis Adapter	ORM	rbatis	✓	MySQL, PostgreSQL, SQLite, SQL Server, MariaDB, TiDB, CockroachDB, Oracle are supported by Rbatis	
Dynamodb Adapter	NoSQL	@fospitia	✓	For Amazon DynamoDB	
MongoDB Adapter	MongoDB	@wangjun861205	✓	For MongoDB	

Adapter		Type	Author		AutoSave	Description
JSON Adapter		String	Casbin		✓	For JSON
YAML Adapter		String	Casbin		✓	For YAML
String Adapter		String	Casbin		✗	For String
Adapter	Type	Author	AutoSave		Description	
File Adapter (built-in)	File	Casbin	✗		For .CSV (Comma-Separated Values) files	
Sequel Adapter	ORM	CasbinRuby	✓		ADO, Amalgalite, IBM_DB, JDBC, MySQL, Mysql2, ODBC, Oracle, PostgreSQL, SQLAnywhere, SQLite3, and TinyTDS are supported by Sequel	
Adapter		Type	Author	AutoSave	Description	
File Adapter (built-in)		File	Casbin	✗	For .CSV (Comma-Separated Values) files	
Memory Adapter (built-in)		Memory	Casbin	✗	For memory	
Fluent Adapter		ORM	Casbin	✓	PostgreSQL, SQLite, MySQL, MongoDB are supported by Fluent	
Adapter	Type	Author		AutoSave	Description	
File Adapter	File	Casbin		✗	For .CSV (Comma-Separated Values)	

Adapter	Type	Author	AutoSave	Description
(built-in)				files
Filtered File Adapter (built-in)	File	Casbin	✗	For .CSV (Comma-Separated Values) files with policy subset loading support
LuaSQL Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite3 are supported by LuaSQL
4DaysORM Adapter	ORM	Casbin	✓	MySQL, SQLite3 are supported by 4DaysORM
OpenResty Adapter	ORM	@tom2nonames	✓	MySQL, PostgreSQL are supported by it

ⓘ NOTE

1. If `casbin.NewEnforcer()` is called with an explicit or implicit adapter, the policy will be loaded automatically.
2. You can call `e.LoadPolicy()` to reload the policy rules from the storage.
3. If the adapter does not support the `Auto-Save` feature, The policy rules cannot be automatically saved back to the storage when you add or remove policies. You have to call `SavePolicy()` manually to save all policy rules.

Examples

Here we provide several examples:

File adapter (built-in)

Below shows how to initialize an enforcer from the built-in file adapter:

[Go](#) [PHP](#) [Rust](#)

```
import "github.com/casbin/casbin"

e := casbin.NewEnforcer("examples/basic_model.conf", "examples/
basic_policy.csv")

use Casbin\Enforcer;

$e = new Enforcer('examples/basic_model.conf', 'examples/basic_policy.csv');

use casbin::prelude::*;

let mut e = Enforcer::new("examples/basic_model.conf", "examples/
basic_policy.csv").await?;
```

This is the same with:

[Go](#) [PHP](#) [Rust](#)

```
import (
    "github.com/casbin/casbin"
    "github.com/casbin/casbin/file-adapter"
)

a := fileadapter.NewAdapter("examples/basic_policy.csv")
e := casbin.NewEnforcer("examples/basic_model.conf", a)

use Casbin\Enforcer;
use Casbin\Persist\Adapters\FileAdapter;

$a = new FileAdapter('examples/basic_policy.csv');
$e = new Enforcer('examples/basic_model.conf', $a);

use casbin::prelude::*;

let a = FileAdapter::new("examples/basic_policy.csv");
let e = Enforcer::new("examples/basic_model.conf", a).await?;
```

MySQL adapter

Below shows how to initialize an enforcer from MySQL database. it connects to a MySQL DB on 127.0.0.1:3306 with root and blank password.

[Go](#) [Rust](#) [PHP](#)

```
import (
    "github.com/casbin/casbin"
    "github.com/casbin/mysql-adapter"
)

a := mysqladapter.NewAdapter("mysql", "root:@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basic_model.conf", a)

// https://github.com/casbin-rs/diesel-adapter
// make sure you activate feature `mysql`

use casbin::prelude::*;
use diesel_adapter::{ConnOptions, DieselAdapter};

let mut conn_opts = ConnOptions::default();
conn_opts
    .set_hostname("127.0.0.1")
    .set_port(3306)
    .set_host("127.0.0.1:3306") // overwrite hostname, port config
    .set_database("casbin")
    .set_auth("casbin_rs", "casbin_rs");

let a = DieselAdapter::new(conn_opts)?;
let mut e = Enforcer::new("examples/basic_model.conf", a).await?;

// https://github.com/php-casbin/dbal-adapter

use Casbin\Enforcer;
use CasbinAdapter\DBAL\Adapter as DatabaseAdapter;

$config = [
    // Either 'driver' with one of the following values:
    // pdo_mysql,pdo_sqlite,pdo_pgsql,pdo_oci (unstable),pdo_sqlsrv,pdo_sqlsrv,
    // mysqli,sqlanywhere,sqlsrv,ibm_db2 (unstable),drizzle_pdo_mysql
```

Use your own storage adapter

You can use your own adapter like below:

```
import (
    "github.com/casbin/casbin"
    "github.com/your-username/your-repo"
)

a := yourpackage.NewAdapter(params)
e := casbin.NewEnforcer("examples/basic_model.conf", a)
```

Migrate/Convert between different adapter

If you want to convert adapter from A to B, you can do like this:

1.Load policy from A to memory

```
e, _ := NewEnforcer(m, A)
```

or

```
e.SetAdapter(A)
e.LoadPolicy()
```

2.convert your adapter from A to B

```
e.SetAdapter(B)
```

3.Save policy from memory to B

```
e.SavePolicy()
```

Load/Save at run-time

You may also want to reload the model, reload the policy or save the policy after initialization:

```
// Reload the model from the model CONF file.  
e.LoadModel()  
  
// Reload the policy from file/database.  
e.LoadPolicy()  
  
// Save the current policy (usually after changed with Casbin API) back to file/  
database.  
e.SavePolicy()
```

AutoSave

There is a feature called `Auto-Save` for adapters. When an adapter supports `Auto-Save`, it means it can support adding a single policy rule to the storage, or removing a single policy rule from the storage. This is unlike `SavePolicy()`, because the latter will delete all policy rules in the storage and save all policy rules from Casbin enforcer to the storage. So it may suffer performance issue when the number of policy rules is large.

When the adapter supports `Auto-Save`, you can switch this option via `Enforcer.EnableAutoSave()` function. The option is enabled by default (if the adapter supports it).

ⓘ NOTE

1. The `Auto-Save` feature is optional. An adapter can choose to implement it or not.
2. `Auto-Save` only works for a Casbin enforcer when the adapter the enforcer uses supports it.
3. See the `AutoSave` column in the above adapter list to see if `Auto-Save` is supported by an adapter.

Here's an example about how to use `Auto-Save`:

```
import (
```

For more examples, please see: https://github.com/casbin/xorm-adapter/blob/master/adapter_test.go

How to write an adapter

All adapters should implement the [Adapter](#) interface by providing at least two mandatory methods: `LoadPolicy(model model.Model) error` and `SavePolicy(model model.Model) error`.

The other three functions are optional. They should be implemented if the adapter supports the [Auto-Save](#) feature.

Method	Type	Description
LoadPolicy()	mandatory	Load all policy rules from the storage
SavePolicy()	mandatory	Save all policy rules to the storage
AddPolicy()	optional	Add a policy rule to the storage
RemovePolicy()	optional	Remove a policy rule from the storage
RemoveFilteredPolicy()	optional	Remove policy rules that match the filter from the storage

 NOTE

If an adapter doesn't support [Auto-Save](#), it should provide an empty implementation for the three optional functions. Here's an example for Golang:

```
// AddPolicy adds a policy rule to the storage.
func (a *Adapter) AddPolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}

// RemovePolicy removes a policy rule from the storage.
func (a *Adapter) RemovePolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}
```

Casbin enforcer will ignore the `not implemented` error when calling these three optional functions.

There're details about how to write an adapter.

- Data Structure. Adapter should support reading at *least* six columns.
- Database Name. The default database name should be `casbin`.
- Table Name. The default table name should be `casbin_rule`.
- Ptype Column. Name of this column should be `ptype` instead of `p_type` or `Ptype`.
- Table definition should be `(id int primary key, ptype varchar, v0 varchar, v1 varchar, v2 varchar, v3 varchar, v4 varchar, v5 varchar)`.
- The unique key index should be built on columns `ptype, v0, v1, v2, v3, v4, v5`.
- `LoadFilteredPolicy` requires a `filter` as parameter. The filter should be something like this.

```
{  
    "p": [ [ "alice" ], [ "bob" ] ],  
    "g": [ [ "", "book_group" ], [ "", "pen_group" ] ],  
    "g2": [ [ "alice" ] ]  
}
```

Who is responsible to create the DB?

As a convention, the adapter should be able to automatically create a database named `casbin` if it doesn't exist and use it for policy storage. Please use the Xorm adapter as a reference implementation: <https://github.com/casbin/xorm-adapter>

Context Adapter

`ContextAdapter` provides a context-aware interface for Casbin adapters.

Through context, you can implement features such as timeout control for the Adapter API

Example

`gormadapter` supports adapter with context, the following is a timeout control implemented using context

```

ca, _ := NewContextAdapter("mysql", "root:@tcp(127.0.0.1:3306)/", "casbin")
// Limited time 300s
ctx, cancel := context.WithTimeout(context.Background(), 300*time.Microsecond)
defer cancel()

err := ca.AddPolicyCtx(ctx, "p", "p", []string{"alice", "data1", "read"})
if err != nil {
    panic(err)
}

```

How to write an context adapter

`ContextAdapter` API only has an extra layer of context processing than ordinary `Adapter` API, and on the basis of implementing ordinary Adapter API, you can encapsulate your own processing logic for context

A simple reference to the `gormadapter`: [adapter.go](#)

Transaction

Casbin now supports Transactions. Here's an example about how to use `Transaction` in `gormadapter`:

```

db, _ := gorm.Open(...)
adapter, _ := gormadapter.NewTransactionalAdapterByDB(db)
e, _ := casbin.NewTransactionalEnforcer("examples/rbac_model.conf", adapter)

ctx := context.Background()

// WithTransaction executes a function within a transaction.
// If the function returns an error, the transaction is rolled back.
// Otherwise, it's committed automatically.
err := e.WithTransaction(ctx, func(tx *casbin.Transaction) error {
    tx.AddPolicy("alice", "data1", "read")
    tx.AddPolicy("alice", "data1", "write")
    return nil
})

// If you wish to manually handle the transaction
tx, _ := e.BeginTransaction(ctx)
tx.AddPolicy("alice", "data1", "write")
if err := tx.Commit(); err != nil {

```

To add this capability to your adapter, you need to implement `TransactionalAdapter` and `TransactionContext` in [persist/transaction.go](#).

Code reference to the `gormadapter`: [adapter.go](#)

Watchers

We support the use of distributed messaging systems like [etcd](#) to maintain consistency between multiple Casbin enforcer instances. This allows our users to concurrently use multiple Casbin enforcers to handle a large number of permission checking requests.

Similar to policy storage adapters, we do not include watcher code in the main library. Any support for a new messaging system should be implemented as a watcher. A complete list of Casbin watchers is provided below. We welcome any third-party contributions for a new watcher, please inform us and we will add it to this list:

[Go](#) [Java](#) [Node.js](#) [Python](#) [.NET](#) [Ruby](#) [PHP](#)

Watcher	Type	Author	Description
PostgreSQL WatcherEx	Database	@IguteChung	WatcherEx for PostgreSQL
Redis WatcherEx	KV store	Casbin	WatcherEx for Redis
Redis Watcher	KV store	@billcobbler	Watcher for Redis
Etcd Watcher	KV store	Casbin	Watcher for etcd
TiKV Watcher	KV store	Casbin	Watcher for TiKV
Kafka Watcher	Messaging system	@wgarunap	Watcher for Apache Kafka
NATS Watcher	Messaging system	Soluto	Watcher for NATS
ZooKeeper Watcher	Messaging	Gepsr	Watcher for Apache ZooKeeper

Watcher	Type	Author	Description
	system		
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@rusenask	Watcher based on Go Cloud Dev Kit that works with leading cloud providers and self-hosted infrastructure
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@bartventer	WatcherEx based on Go Cloud Dev Kit that works with leading cloud providers and self-hosted infrastructure
RocketMQ Watcher	Messaging system	@fmyxyz	Watcher for Apache RocketMQ
Watcher	Type	Author	Description
Etcd Adapter	KV store	@mapleafgo	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Redis WatcherEx	KV store	Casbin	WatcherEx for Redis
Lettuce-Based Redis Watcher	KV store	Casbin	Watcher for Redis based on Lettuce)
PostgreSQL Watcher	Database	Casbin	Watcher for PostgreSQL
Kafka Watcher	Messaging system	Casbin	Watcher for Apache Kafka

Watcher	Type	Author	Description
Etcd Watcher	KV store	Casbin	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Pub/Sub Watcher	Messaging system	Casbin	Watcher for Google Cloud Pub/Sub
MongoDB Change Streams Watcher	Database	Casbin	Watcher for MongoDB Change Streams
Postgres Watcher	Database	@mcollina	Watcher for PostgreSQL

Watcher	Type	Author	Description
Etcd Watcher	KV store	Casbin	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Redis Watcher	KV store	ScienceLogic	Watcher for Redis
Redis Async Watcher	KV store	@kevinkelin	Watcher for Redis
PostgreSQL Watcher	Database	Casbin	Watcher for PostgreSQL
RabbitMQ Watcher	Messaging system	Casbin	Watcher for RabbitMQ

Watcher	Type	Author	Description
Redis Watcher	KV store	@Sbou	Watcher for Redis

Watcher	Type	Author	Description
Redis Watcher	KV store	CasbinRuby	Watcher for Redis
RabbitMQ Watcher	Messaging system	CasbinRuby	Watcher for RabbitMQ
Watcher	Type	Author	Description
Redis Watcher	KV store	@Tinywan	Watcher for Redis

WatcherEx

In order to support incremental synchronization between multiple instances, we provide the `WatcherEx` interface. We hope it can notify other instances when the policy changes, but there is currently no implementation of `WatcherEx`. We recommend that you use dispatcher to achieve this.

Compared with `Watcher` interface, `WatcherEx` can distinguish what type of update action is received, e.g., `AddPolicy` and `RemovePolicy`.

WatcherEx Apis:

API	Description
<code>SetUpdateCallback(func(string))</code> error	<code>SetUpdateCallback</code> sets the callback function that the watcher will call, when the policy in DB has been changed by other instances. A classic callback is <code>Enforcer.LoadPolicy()</code> .
<code>Update()</code> error	<code>Update</code> calls the update callback of other instances to synchronize their policy. It is usually called after changing the policy in DB, like <code>Enforcer.SavePolicy()</code> , <code>Enforcer.AddPolicy()</code> ,

API	Description
	Enforcer.RemovePolicy(), etc.
Close()	Close stops and releases the watcher, the callback function will not be called any more.
UpdateForAddPolicy(sec, ptype string, params ...string) error	UpdateForAddPolicy calls the update callback of other instances to synchronize their policy. It is called after a policy is added via Enforcer.AddPolicy(), Enforcer.AddNamedPolicy(), Enforcer.AddGroupingPolicy() and Enforcer.AddNamedGroupingPolicy().
UpdateForRemovePolicy(sec, ptype string, params ...string) error	UPdateForRemovePolicy calls the update callback of other instances to synchronize their policy. It is called after a policy is removed by Enforcer.RemovePolicy(), Enforcer.RemoveNamedPolicy(), Enforcer.RemoveGroupingPolicy() and Enforcer.RemoveNamedGroupingPolicy().
UpdateForRemoveFilteredPolicy(sec, ptype string, fieldIndex int, fieldValues ...string) error	UpdateForRemoveFilteredPolicy calls the update callback of other instances to synchronize their policy. It is called after Enforcer.RemoveFilteredPolicy(), Enforcer.RemoveFilteredNamedPolicy(), Enforcer.RemoveFilteredGroupingPolicy() and Enforcer.RemoveFilteredNamedGroupingPolicy().
UpdateForSavePolicy(model model.Model) error	UpdateForSavePolicy calls the update callback of other instances to synchronize their policy. It is called after Enforcer.SavePolicy()

API	Description
UpdateForAddPolicies(sec string, ptype string, rules ...[]string) error	UpdateForAddPolicies calls the update callback of other instances to synchronize their policy. It is called after Enforcer.AddPolicies(), Enforcer.AddNamedPolicies(), Enforcer.AddGroupingPolicies() and Enforcer.AddNamedGroupingPolicies().
UpdateForRemovePolicies(sec string, ptype string, rules ...[]string) error	UpdateForRemovePolicies calls the update callback of other instances to synchronize their policy. It is called after Enforcer.RemovePolicies(), Enforcer.RemoveNamedPolicies(), Enforcer.RemoveGroupingPolicies() and Enforcer.RemoveNamedGroupingPolicies().

Dispatchers

Dispatchers provide a way to synchronize incremental changes of policy. They should be based on consistency algorithms such as Raft to ensure the consistency of all enforcer instances. Through dispatchers, users can easily establish distributed clusters.

The dispatcher's method is divided into two parts. The first part is the method combined with Casbin. These methods should be called inside Casbin. Users can use the more complete API provided by Casbin itself.

The other part is the method defined by the dispatcher itself, including the dispatcher initialization method, and different functions provided by different algorithms, such as dynamic membership and config changes.

 NOTE

We hope dispatchers only ensure the consistency of the Casbin enforcer at runtime. So if the policy is inconsistent during initialization, the dispatchers will not work properly. Users need to ensure that the state of all instances is consistent before using dispatchers.

A complete list of Casbin dispatchers is provided below. Any 3rd-party contributions on a new dispatcher are welcomed. Please inform us, and we will add it to this list.

[Go](#)

Adapter	Type	Author	Description
Hashicorp Raft Dispatcher	Raft	Casbin	A dispatcher based on Hashicorp Raft
KDKYG/casbin-dispatcher	Raft	@KDKYG	A dispatcher based on Hashicorp Raft

DistributedEnforcer

DistributedEnforcer wraps SyncedEnforcer for the dispatcher.

Go

```
e, _ := casbin.NewDistributedEnforcer("examples/basic_model.conf", "examples/basic_policy.csv")
```

Role Managers

The role manager is used to manage the RBAC role hierarchy (user-role mapping) in Casbin. A role manager can retrieve role data from Casbin policy rules or external sources such as LDAP, Okta, Auth0, Azure AD, etc. We support different implementations of a role manager. To keep the lightweight, we don't include role manager code in the main library (except the default role manager). A complete list of Casbin role managers is provided below. Any third-party contributions for a new role manager are welcome. Please inform us, and we will add it to this list:)

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#)

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy
Session Role Manager	EDOMO Systems	Supports role hierarchy stored in the Casbin policy, with time-range-based sessions
Okta Role Manager	Casbin	Supports role hierarchy stored in Okta
Auth0 Role Manager	Casbin	Supports role hierarchy stored in Auth0's Authorization Extension

For developers: all role managers must implement the [RoleManager](#) interface. The [Session Role Manager](#) can be used as a reference implementation.

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

For developers: all role managers must implement the [RoleManager](#) interface. The [Default Role Manager](#) can be used as a reference implementation.

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy
Session Role Manager	Casbin	Supports role hierarchy stored in the Casbin policy, with time-range-based sessions

For developers: all role managers must implement the [RoleManager](#) interface. The [Default Role Manager](#) can be used as a reference implementation.

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

For developers: all role managers must implement the [RoleManager](#) interface. The [Default Role Manager](#) can be used as a reference implementation.

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

For developers: all role managers must implement the [RoleManager](#) interface. The [Default Role Manager](#) can be used as a reference implementation.

API

See the [API](#) section for details.

Middlewares

Web frameworks

Go Java Node.js PHP Python C++ .NET Rust Lua

Swift

Name	Description
Gin	A HTTP web framework featuring a Martini-like API with much better performance, via plugin: authz or gin-casbin
Beego	An open-source, high-performance web framework for Go, via built-in plugin: plugins/authz
Caddy	Fast, cross-platform HTTP/2 web server with automatic HTTPS, via plugin: caddy-authz
Traefik	The cloud native application proxy, via plugin: traefik-auth-plugin
Kratos	Your ultimate Go microservices framework for the cloud-native era, via plugin: tx7do/kratos-casbin or overstarry/kratos-casbin
Go kit	A toolkit for microservices, via built-in plugin: plugins/authz
Fiber	An Express inspired web framework written in Go, via middleware: casbin in gofiber/contrib or fiber-casbinrest or fiber-

Name	Description
	boilerplate or gofiber-casbin
FastHTTP	Fast HTTP package for Go. Tuned for high performance. Zero memory allocations in hot paths. Up to 10x faster than <code>net/http</code> , via plugin: fasthttp-auth
Revel	A high productivity, full-stack web framework for the Go language, via plugin: auth/casbin
Echo	High performance, minimalist Go web framework, via plugin: echo-authz or echo-casbin or casbinrest or echo-boilerplate
Iris	The fastest web framework for Go in (THIS) Earth. HTTP/2 Ready-To-GO, via plugin: casbin or iris-middleware-casbin
GoFrame	A modular, powerful, high-performance and enterprise-class application development framework of Golang, via plugin: gf-casbin
Negroni	Idiomatic HTTP Middleware for Golang, via plugin: negroni-authz
Chi	A lightweight, idiomatic and composable router for building HTTP services, via plugin: chi-authz
Buffalo	A Go web development eco-system, designed to make your life easier, via plugin: buffalo-mw-rbac
Macaron	A high productive and modular web framework in Go, via plugin:

Name	Description
	authz
DotWeb	Simple and easy go web micro framework, via plugin: authz
Tango	Micro & pluggable web framework for Go, via plugin: authz
Baa	An express Go web framework with routing, middleware, dependency injection and http context, via plugin: authz
Tyk	An open source Enterprise API Gateway, supporting REST, GraphQL, TCP and gRPC protocols, via plugin: tyk-authz
Hertz	Go HTTP framework with high-performance and strong-extensibility for building micro-services, via plugin: casbin
Name	Description
Spring Boot	Makes it easy to create Spring-powered applications and services, via plugin: casbin-spring-boot-starter or Simple SpringBoot security demo with jCasbin
Apache Shiro	A powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management, via plugin: shiro-casbin or shiro-jcasbin-spring-boot-starter
JFinal	A simple, light, rapid, independent and extensible Java WEB + ORM framework, via plugin: jfinal-authz

Name	Description
Nutz	Web framework (MVC/IOC/AOP/DAO/JSON) for all Java developers, via plugin: nutz-authz
mangoo I/O	An intuitive, lightweight, high performance full stack Java web framework, via built-in plugin: AuthorizationService.java
Name	Description
Shield	An authZ server and authZ aware reverse-proxy built on top of casbin.
Express	Fast, unopinionated, minimalist web framework for node, via plugin: express-authz
Koa	Expressive middleware for node.js using ES2017 async functions, via plugin: koa-authz or koajs-starter or koa-casbin
LoopBack 4	A highly extensible Node.js and TypeScript framework for building APIs and microservices, via plugin: loopback4-authorization
Nest	Progressive Node.js framework for building efficient and scalable server-side applications on top of TypeScript & JavaScript. via plugin: nest-authz or nest-casbin or NestJS Casbin Module or nestjs-casbin or acl-nest or nestjs-casbin-typeorm
Fastify	Fast and low overhead web framework, for Node.js. via plugin: fastify-casbin or fastify-casbin-rest

Name	Description
Egg	Born to build better enterprise frameworks and apps with Node.js & Koa, via plugin: egg-authz or egg-zrole
hapi	The Simple, Secure Framework Developers Trust. via plugin: hapi-authz
Casbin JWT Express	Authorization middleware that uses stateless JWT token to validate ACL rules using Casbin
Hono	Fast, lightweight, built on Web Standards. via plugin: @hono/casbin
Name	Description
Laravel	The PHP framework for web artisans, via plugin: laravel-authz
Yii PHP Framework	A fast, secure, and efficient PHP framework, via plugin: yii-permission or yii-casbin
CakePHP	Build fast, grow solid PHP Framework, via plugin: cake-permission
CodeIgniter	Associate users with roles and permissions in CodeIgniter4 Web Framework, via plugin: CodeIgniter Permission
ThinkPHP 5.1	The ThinkPHP 5.1 framework, via plugin: think-casbin

Name	Description
ThinkPHP 6.0	The ThinkPHP 6.0 framework, via plugin: think-authz
Symfony	The Symfony PHP framework, via plugin: symfony-permission or symfony-casbin
Hyperf	A coroutine framework that focuses on hyperspeed and flexibility, via plugin: hyperf-permission or donjan-deng/hyperf-casbin or cblink/hyperf-casbin
EasySwoole	A distributed, persistent memory PHP framework based on the Swoole extension, via plugin: easyswoole-permission or easyswoole-hyperfOrm-permission
Slim	A PHP micro framework that helps you quickly write simple yet powerful web applications and APIs, via plugin: casbin-with-slim
Phalcon	A full-stack PHP framework delivered as a C-extension, via plugin: phalcon-permission
Webman	High performance HTTP Service Framework for PHP based on Workerman, via plugin: webman-permission or webman-casbin
Name	Description
Django	A high-level Python Web framework, via plugin: django-casbin or django-casbin-auth

Name	Description
Flask	A microframework for Python based on Werkzeug, Jinja 2 and good intentions, via plugin: flask-authz or Flask-Casbin (3rd-party, but maybe more friendly) or rbac-flask
FastAPI	A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints, via plugin: fastapi-casbin-auth or Fastapi-app
OpenStack	The most widely deployed open source cloud software in the world, via plugin: openstack-patron
Tornado	Tornado is a Python web framework and asynchronous networking library, via plugin: tornado-authz
Name	Description
Nginx	A HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, via plugin: nginx-casbin-module
Name	Description
ASP.NET Core	An open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends, via plugin: Casbin.AspNetCore
ASP.NET Core	A simple demo of using Casbin at ASP.NET Core framework, via plugin: CasbinACL-aspNetCore

Name	Description
Actix	A Rust actors framework, via plugin: actix-casbin
Actix web	A small, pragmatic, and extremely fast rust web framework, via plugin: actix-casbin-auth
Rocket	a web framework for Rust that makes it simple to write fast, secure web applications without sacrificing flexibility, usability, or type safety, via plugin: rocket-authz or rocket-casbin-auth
Axum web	A ergonomic and modular rust web framework, via plugin: axum-casbin-auth
Poem web	A full-featured and easy to use web framework with the Rust programming language, via plugin: poem-casbin
Name	Description
OpenResty	A dynamic web platform based on NGINX and LuaJIT, via plugin: lua-resty-casbin and casbin-openresty-example
Kong	A cloud-native, platform-agnostic, scalable API Gateway distinguished for its high performance and extensibility via plugins, via plugin: kong-authz
APISIX	A dynamic, real-time, high-performance API gateway, via plugin: authz-casbin

Name	Description
Vapor	A server-side Swift web framework, via plugin: vapor-authz

Cloud providers

[Node.js](#)

Name	Description
Okta	One trusted platform to secure every identity, via plugin: casbin-spring-boot-demo
Auth0	An easy to implement, adaptable authentication and authorization platform, via plugin: casbin-auth0-rbac

GraphQL Middlewares

Casbin follows the officially suggested way to provide authorization for GraphQL endpoints by having a single source of truth for authorization: <https://graphql.org/learn/authorization/>. In other words, Casbin should be placed between the GraphQL layer and your business logic.

```
// Casbin authorization logic lives inside postRepository
var postRepository = require('postRepository');

var postType = new GraphQLObjectType({
  name: 'Post',
  fields: {
    body: {
      type: GraphQLString,
      resolve: (post, args, context, { rootValue }) => {
        return postRepository.getBody(context.user, post);
      }
    }
  }
});
```

Supported GraphQL Middlewares

A complete list of Casbin GraphQL middlewares is provided below. Any third-party contributions on a new GraphQL middleware are welcomed. Please inform us, and we will add it to this list:)

[Go](#) [Node.js](#) [Python](#)

Middleware	GraphQL Implementation	Author	Description
graphql-authz	graphql	Casbin	An authorization middleware for graphql-go
graphql-casbin	graphql	@esmaeilpour	An implementation of using Graphql and Casbin together
gqlgen_casbin_RBAC_example	gqlgen	@WenyXu	(empty)
Middleware	GraphQL Implementation	Author	Description
graphql-authz	GraphQL.js	Casbin	A Casbin authorization middleware for GraphQL.js
Middleware	GraphQL Implementation	Author	Description
graphql-authz	GraphQL-core 3	@Checho3388	A Casbin authorization middleware for GraphQL-core 3

Cloud Native Middlewares

Cloud Native Projects

Go Node.js

Project	Author	Description
k8s-authz	Casbin	Authorization middleware for Kubernetes
envoy-authz	Casbin	Authorization middleware for Istio and Envoy
kubesphere-authz	Casbin	Authorization middleware for kubeSphere

Project	Author	Description
ODPF Shield	Open Data Platform	ODPF Shield is a cloud native role-based authorization-aware reverse-proxy service.

API

API Overview

Casbin API Usage

Management API

The primitive API that provides full support for Casbin policy management

RBAC API

A more friendly API for RBAC. This API is a subset of Management API. The RBAC users could use this API to simplify the code

RBAC with Domains API

A more user-friendly API for RBAC with domains. This API is a subset of the Management API. RBAC users can use this API to simplify their code.

RBAC with Conditions API

A more user-friendly API for RBAC with conditions.

RoleManager API

The RoleManager API provides an interface for defining operations to manage roles. The addition of a matching function to the RoleManager allows the use of wildcards...

API Overview

This overview only shows you how to use Casbin APIs and doesn't explain how Casbin is installed or how it works. You can find those tutorials here: [Installation of Casbin](#) and [How Casbin Works](#). So, when you start reading this tutorial, we assume that you have fully installed and imported Casbin into your code.

Enforce API

Let's start with the Enforce APIs of Casbin. We will load a RBAC model from `model.conf` and load policies from `policy.csv`. You can learn about the Model syntax [here](#), and we won't discuss it in this tutorial. We assume that you can understand the config files given below:

`model.conf`

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

policy.csv

```
p, admin, data1, read
p, admin, data1, write
p, admin, data2, read
p, admin, data2, write
p, alice, data1, read
p, bob, data2, write
g, amber, admin
g, abc, admin
```

After reading the config files, please read the following code.

```
// Load information from files.
enforcer, err := casbin.NewEnforcer("./example/model.conf",
"./example/policy.csv")
if err != nil {
    log.Fatalf("Error, detail: %s", err)
}
ok, err := enforcer.Enforce("alice", "data1", "read")
```

This code loads the access control model and policies from local files. The function `casbin.NewEnforcer()` will return an enforcer. It will recognize its two parameters as file paths and load the files from there. Errors occurred in the process are stored in the variable `err`. This code uses the default adapter to load the model and policies, and of course, you can achieve the same result by using a third-party adapter.

The code `ok, err := enforcer.Enforce("alice", "data1", "read")` is used to confirm access permissions. If Alice can access data1 with the read operation, the returned value of `ok` will be `true`; otherwise, it will be `false`. In this example, the value of `ok` is `true`.

EnforceEx API

Sometimes you may wonder which policy allowed the request, so we have prepared the function `EnforceEx()`. You can use it like this:

```
ok, reason, err := enforcer.EnforceEx("amber", "data1", "read")
fmt.Println(ok, reason) // true [admin data1 read]
```

The `EnforceEx()` function will return the exact policy string in the return value `reason`. In this example, `amber` is an `admin` role, so the policy `p, admin, data1, read` allowed this request to be `true`. The output of this code is in the comment.

Casbin has provided many APIs similar to this one. These APIs add some extra functions to the basic ones. They include:

- `ok, err := enforcer.EnforceWithMatcher(matcher, request)`

This function uses a matcher.

- `ok, reason, err := enforcer.EnforceExWithMatcher(matcher, request)`

This is a combination of `EnforceWithMatcher()` and `EnforceEx()`.

- `boolArray, err := enforcer.BatchEnforce(requests)`

This function allows for a list of jobs and returns an array.

This is a simple use case of Casbin. You can use Casbin to start an authorization server using these APIs. We will show you some other types of APIs in the

following paragraphs.

Management API

Get API

These APIs are used to retrieve specific objects in policies. In this example, we are loading an enforcer and retrieving something from it.

Please take a look at the following code:

```
enforcer, err := casbin.NewEnforcer("./example/model.conf",
"./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}
allSubjects := enforcer.GetAllSubjects()
fmt.Println(allSubjects)
```

Similar to the previous example, the first four lines are used to load necessary information from local files. We won't discuss that here any further.

The code `allSubjects := enforcer.GetAllSubjects()` retrieves all the subjects in the policy file and returns them as an array. We then print that array.

Typically, the output of the code should be:

```
[admin alice bob]
```

You can also change the function `GetAllSubjects()` to `GetAllNamedSubjects()` to get the list of subjects that appear in the current

named policy.

Similarly, we have prepared `GetAll` functions for `Objects`, `Actions`, `Roles`. To access these functions, you simply need to replace the word `Subject` in the function name with the desired category.

Additionally, there are more getters available for policies. The method of calling and the return values are similar to the ones mentioned above.

- `policy = e.GetPolicy()` retrieves all the authorization rules in the policy.
- `filteredPolicy := e.GetFilteredPolicy(0, "alice")` retrieves all the authorization rules in the policy with specified field filters.
- `namedPolicy := e.GetNamedPolicy("p")` retrieves all the authorization rules in the named policy.
- `filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")` retrieves all the authorization rules in the named policy with specified field filters.
- `groupingPolicy := e.GetGroupingPolicy()` retrieves all the role inheritance rules in the policy.
- `filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")` retrieves all the role inheritance rules in the policy with specified field filters.
- `namedGroupingPolicy := e.GetNamedGroupingPolicy("g")` retrieves all the role inheritance rules in the policy.
- `namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0, "alice")` retrieves all the role inheritance rules in the policy with specified field filters.

Add, Delete, Update API

Casbin provides a variety of APIs for dynamically adding, deleting, or modifying

policies at runtime.

The following code demonstrates how to add, remove, and update policies, as well as how to check if a policy exists:

```
// load information from files
enforcer, err := casbin.NewEnforcer("./example/model.conf",
"./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}

// add a policy and use HasPolicy() to confirm
enforcer.AddPolicy("added_user", "data1", "read")
hasPolicy := enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // true, the policy was added
successfully

// remove a policy and use HasPolicy() to confirm
enforcer.RemovePolicy("alice", "data1", "read")
hasPolicy = enforcer.HasPolicy("alice", "data1", "read")
fmt.Println(hasPolicy) // false, the policy was removed
successfully

// update a policy and use HasPolicy() to confirm
enforcer.UpdatePolicy([]string{"added_user", "data1", "read"}, 
[]string{"added_user", "data1", "write"})
hasPolicy = enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // false, the original policy has expired
hasPolicy = enforcer.HasPolicy("added_user", "data1", "write")
fmt.Println(hasPolicy) // true, the new policy is in effect
```

By using these APIs, you can edit your policies dynamically. Similarly, we have provided similar APIs for `FilteredPolicy`, `NamedPolicy`, `FilteredNamedPolicy`, `GroupingPolicy`, `NamedGroupingPolicy`,

`FilteredGroupingPolicy`, `FilteredNamedGroupingPolicy`. To use them, simply replace the word `Policy` in the function name with the appropriate category.

Furthermore, by changing the parameters to arrays, you can perform batch editing of your policies.

For example, consider functions like this:

```
enforcer.UpdatePolicy([]string{"eve", "data3", "read"},  
[]string{"eve", "data3", "write"})
```

If we change `Policy` to `Policies` and modify the parameters as follows:

```
enforcer.UpdatePolicies([][]string{{"eve", "data3", "read"},  
{"jack", "data3", "read"}}, [][]string{{"eve", "data3",  
"write"}, {"jack", "data3", "write"}})
```

then we can perform batch editing of these policies.

The same operations can also be applied to `GroupingPolicy`, `NamedGroupingPolicy`.

AddEx API

Casbin provides the AddEx series of APIs to help users add rules in batches.

```
AddPoliciesEx(rules [][]string) (bool, error)  
AddNamedPoliciesEx(ptype string, rules [][]string) (bool, error)  
AddGroupingPoliciesEx(rules [][]string) (bool, error)  
AddNamedGroupingPoliciesEx(ptype string, rules [][]string)
```

The difference between these methods and the methods without the Ex suffix is that if one of the rules already exists, they will continue checking the next rule instead of returning false immediately.

For example, let's compare `AddPolicies` and `AddPoliciesEx`.

You can run and observe the following code by copying it into the test under casbin.

```
func TestDemo(t *testing.T) {
    e, err := NewEnforcer("examples/basic_model.conf",
"examples/basic_policy.csv")
    if err != nil {
        fmt.Printf("Error, details: %s\n", err)
    }
    e.ClearPolicy()
    e.AddPolicy("user1", "data1", "read")
    fmt.Println(e.GetPolicy())
    testGetPolicy(t, e, [][]string{{"user1", "data1", "read"}})

    // policy {"user1", "data1", "read"} now exists

    // Use AddPolicies to add rules in batches
    ok, _ := e.AddPolicies([][]string{{"user1", "data1",
"read"}, {"user2", "data2", "read"}})
    fmt.Println(e.GetPolicy())
    // {"user2", "data2", "read"} failed to add because
    // {"user1", "data1", "read"} already exists
    // AddPolicies returns false and no other policies are
    // checked, even though they may not exist in the existing ruleset
    // ok == false
    fmt.Println(ok)
    testGetPolicy(t, e, [][]string{{"user1", "data1", "read"}})

    // Use AddPoliciesEx to add rules in batches
```

RBAC API

Casbin provides some APIs for you to modify the RBAC model and policies. If you are familiar with RBAC, you can easily use these APIs.

Here, we only show you how to use the RBAC APIs of Casbin and won't talk about RBAC itself. You can get more details [here](#).

We use the following code to load the model and policies, just like before.

```
enforcer, err := casbin.NewEnforcer("./example/model.conf",
"./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}
```

Then, we can use an instance of the Enforcer `enforcer` to access these APIs.

```
roles, err := enforcer.GetRolesForUser("amber")
fmt.Println(roles) // [admin]
users, err := enforcer.GetUsersForRole("admin")
fmt.Println(users) // [amber abc]
```

`GetRolesForUser()` returns an array that contains all the roles that amber has. In this example, amber has only one role, which is admin, so the array `roles` is `[admin]`. Similarly, you can use `GetUsersForRole()` to get the users who belong to a role. The return value of this function is also an array.

```
enforcer.HasRoleForUser("amber", "admin") // true
```

You can use `HasRoleForUser()` to confirm whether the user belongs to the role. In this example, amber is a member of admin, so the return value of the function is `true`.

```
fmt.Println(enforcer.Enforce("bob", "data2", "write")) // true  
enforcer.DeletePermission("data2", "write")  
fmt.Println(enforcer.Enforce("bob", "data2", "write")) // false
```

You can use `DeletePermission()` to delete a permission.

```
fmt.Println(enforcer.Enforce("alice", "data1", "read")) // true  
enforcer.DeletePermissionForUser("alice", "data1", "read")  
fmt.Println(enforcer.Enforce("alice", "data1", "read")) // false
```

And use `DeletePermissionForUser()` to delete a permission for a user.

Casbin has many APIs like this. Their calling methods and return values have the same style as the above APIs. You can find these APIs in the [next documents](#).

Management API

The primitive API that provides full support for Casbin policy management.

Filtered API

Almost all filtered api has the same parameters (`fieldIndex int, fieldValues ...string`). `fieldIndex` is the index where matching start, `fieldValues` denotes the values result should have. Note that empty string in `fieldValues` could be any word.

Example:

```
p, alice, book, read
p, bob, book, read
p, bob, book, write
p, alice, pen, get
p, bob, pen ,get
```

```
e.GetFilteredPolicy(1, "book") // will return: [["alice", "book", "read"], ["bob", "book", "read"], ["bob", "book", "write"]]

e.GetFilteredPolicy(1, "book", "read") // will return: [["alice", "book", "read"], ["bob", "book", "read"]]

e.GetFilteredPolicy(0, "alice", "", "read") // will return: [[alice book read]]

e.GetFilteredPolicy(0, "alice") // will return: [["alice", "book", "read"], ["alice", "pen", "get"]]
```

Reference

global variable `e` is Enforcer instance.

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e, err := NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

```
const e = await newEnforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv')
```

```
$e = new Enforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv');
```

```
e = casbin.Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

```
var e = new Enforcer("path/to/model.conf", "path/to/policy.csv");
```

```
let mut e = Enforce::new("examples/rbac_model.conf", "examples/rbac_policy.csv").await?;
```

```
Enforcer e = new Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv");
```

Enforce()

Enforce decides whether a "subject" can access a "object" with the operation "action", input parameters are usually: (sub, obj, act).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [Java](#)

```
ok, err := e.Enforce(request)

const ok = await e.enforce(request);

$ok = $e->enforcer($request);

ok = e.enforcer(request)

boolean ok = e.enforce(request);
```

EnforceWithMatcher()

EnforceWithMatcher use a custom matcher to decides whether a "subject" can access a "object" with the operation "action", input parameters are usually: (matcher, sub, obj, act), use model matcher by default when matcher is "".

For example:

[Go](#) [PHP](#) [Python](#) [Java](#)

```
ok, err := e.EnforceWithMatcher(matcher, request)

$ok = $e->enforceWithMatcher($matcher, $request);

ok = e.enforce_with_matcher(matcher, request)

boolean ok = e.enforceWithMatcher(matcher, request);
```

EnforceEx()

EnforceEx explain enforcement by informing matched rules.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#)

```
ok, reason, err := e.EnforceEx(request)

const ok = await e.enforceEx(request);

list($ok, $reason) = $e->enforceEx($request);

ok, reason = e.enforce_ex(request)
```

EnforceExWithMatcher()

EnforceExWithMatcher use a custom matcher and explain enforcement by informing matched rules.

For example:

[Go](#)

```
ok, reason, err := e.EnforceExWithMatcher(matcher, request)
```

BatchEnforce()

BatchEnforce enforces each request and returns result in a bool array

For example:

Go Node.js Java

```
boolArray, err := e.BatchEnforce(requests)

const boolArray = await e.batchEnforce(requests);

List<Boolean> boolArray = e.batchEnforce(requests);
```

GetAllSubjects()

GetAllSubjects gets the list of subjects that show up in the current policy.

For example:

Go Node.js PHP Python .NET Rust Java

```
allSubjects := e.GetAllSubjects()

const allSubjects = await e.getAllSubjects()

$allSubjects = $e->getAllSubjects();

all_subjects = e.get_all_subjects()

var allSubjects = e.GetAllSubjects();

let all_subjects = e.get_all_subjects();

List<String> allSubjects = e.getAllSubjects();
```

GetAllNamedSubjects()

GetAllNamedSubjects gets the list of subjects that show up in the current named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allNamedSubjects := e.GetAllNamedSubjects("p")  
  
const allNamedSubjects = await e.getAllNamedSubjects('p')  
  
$allNamedSubjects = $e->getAllNamedSubjects("p");  
  
all_named_subjects = e.get_all_named_subjects("p")  
  
var allNamedSubjects = e.GetAllNamedSubjects("p");  
  
let all_named_subjects = e.get_all_named_subjects("p");  
  
List<String> allNamedSubjects = e.getAllNamedSubjects("p");
```

GetAllObjects()

GetAllObjects gets the list of objects that show up in the current policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allObjects := e.GetAllObjects()

const allObjects = await e.getAllObjects()

$allObjects = $e->getAllObjects();

all_objects = e.get_all_objects()

var allObjects = e.GetAllObjects();

let all_objects = e.get_all_objects();

List<String> allObjects = e.getAllObjects();
```

GetAllNamedObjects()

GetAllNamedObjects gets the list of objects that show up in the current named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allNamedObjects := e.GetAllNamedObjects("p")

const allNamedObjects = await e.getAllNamedObjects('p')

$allNamedObjects = $e->getAllNamedObjects("p");

all_named_objects = e.get_all_named_objects("p")

var allNamedObjects = e.GetAllNamedObjects("p");
```

```
let all_named_objects = e.get_all_named_objects("p");

List<String> allNamedObjects = e.getAllNamedObjects("p");
```

GetAllActions()

GetAllActions gets the list of actions that show up in the current policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allActions := e.GetAllActions()

const allActions = await e.getAllActions()

$allActions = $e->getAllActions();

all_actions = e.get_all_actions()

var allActions = e.GetAllActions();

let all_actions = e.get_all_actions();

List<String> allActions = e.getAllActions();
```

GetAllNamedActions()

GetAllNamedActions gets the list of actions that show up in the current named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allNamedActions := e.GetAllNamedActions("p")

const allNamedActions = await e.getAllNamedActions('p')

$allNamedActions = $e->getAllNamedActions("p");

all_named_actions = e.get_all_named_actions("p")

var allNamedActions = e.GetAllNamedActions("p");

let all_named_actions = e.get_all_named_actions("p");

List<String> allNamedActions = e.getAllNamedActions("p");
```

GetAllRoles()

GetAllRoles gets the list of roles that show up in the current policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allRoles = e.GetAllRoles()

const allRoles = await e.getAllRoles()

$allRoles = $e->getAllRoles();
```

```
all_roles = e.get_all_roles()

var allRoles = e.GetAllRoles();

let all_roles = e.get_all_roles();

List<String> allRoles = e.getAllRoles();
```

GetAllNamedRoles()

GetAllNamedRoles gets the list of roles that show up in the current named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allNamedRoles := e.GetAllNamedRoles("g")

const allNamedRoles = await e.getAllNamedRoles('g')

$allNamedRoles = $e->getAllNamedRoles('g');

all_named_roles = e.get_all_named_roles("g")

var allNamedRoles = e.GetAllNamedRoles("g");

let all_named_roles = e.get_all_named_roles("g");

List<String> allNamedRoles = e.getAllNamedRoles("g");
```

GetPolicy()

GetPolicy gets all the authorization rules in the policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
policy = e.GetPolicy()

const policy = await e.getPolicy()

$policy = $e->getPolicy();

policy = e.get_policy()

var policy = e.GetPolicy();

let policy = e.get_policy();

List<List<String>> policy = e.getPolicy();
```

GetFilteredPolicy()

GetFilteredPolicy gets all the authorization rules in the policy, field filters can be specified.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
filteredPolicy := e.GetFilteredPolicy(0, "alice")

const filteredPolicy = await e.getFilteredPolicy(0, 'alice')

$filteredPolicy = $e->getFilteredPolicy(0, "alice");

filtered_policy = e.get_filtered_policy(0, "alice")

var filteredPolicy = e.GetFilteredPolicy(0, "alice");

let filtered_policy = e.get_filtered_policy(0,
vec!["alice".to_owned()]);

List<List<String>> filteredPolicy = e.getFilteredPolicy(0, "alice");
```

GetNamedPolicy()

GetNamedPolicy gets all the authorization rules in the named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
namedPolicy := e.GetNamedPolicy("p")

const namedPolicy = await e.getNamedPolicy('p')

$namedPolicy = $e->getNamedPolicy("p");

named_policy = e.get_named_policy("p")
```

```
var namedPolicy = e.GetNamedPolicy("p");

let named_policy = e.get_named_policy("p");

List<List<String>> namedPolicy = e.getNamedPolicy("p");
```

GetFilteredNamedPolicy()

GetFilteredNamedPolicy gets all the authorization rules in the named policy, field filters can be specified.

For example:

Go Node.js PHP Python .NET Rust Java

```
filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")

const filteredNamedPolicy = await e.getFilteredNamedPolicy('p', 0,
  'bob')

$filteredNamedPolicy = $e->getFilteredNamedPolicy("p", 0, "bob");

filtered_named_policy = e.get_filtered_named_policy("p", 0, "alice")

var filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "alice");

let filtered_named_policy = e.get_filtered_named_policy("p", 0,
  vec!["bob".to_owned()]);

List<List<String>> filteredNamedPolicy = e.getFilteredNamedPolicy("p",
  0, "bob");
```

GetGroupingPolicy()

GetGroupingPolicy gets all the role inheritance rules in the policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
groupingPolicy := e.GetGroupingPolicy()

const groupingPolicy = await e.getGroupingPolicy()

$groupingPolicy = $e->getGroupingPolicy();

grouping_policy = e.get_grouping_policy()

var groupingPolicy = e.GetGroupingPolicy();

let grouping_policy = e.get_grouping_policy();

List<List<String>> groupingPolicy = e.getGroupingPolicy();
```

GetFilteredGroupingPolicy()

GetFilteredGroupingPolicy gets all the role inheritance rules in the policy, field filters can be specified.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")

const filteredGroupingPolicy = await e.getFilteredGroupingPolicy(0,
  'alice')

$filteredGroupingPolicy = $e->getFilteredGroupingPolicy(0, "alice");

filtered_grouping_policy = e.get_filtered_grouping_policy(0, "alice")

var filteredGroupingPolicy = e.GetFilteredGroupingPolicy(0, "alice");

let filtered_grouping_policy = e.get_filtered_grouping_policy(0,
vec!["alice".to_owned()]);
List<List<String>> filteredGroupingPolicy =
e.getFilteredGroupingPolicy(0, "alice");
```

GetNamedGroupingPolicy()

GetNamedGroupingPolicy gets all the role inheritance rules in the policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
namedGroupingPolicy := e.GetNamedGroupingPolicy("g")

const namedGroupingPolicy = await e.getNamedGroupingPolicy('g')

$namedGroupingPolicy = $e->getNamedGroupingPolicy("g");
```

```
named_grouping_policy = e.get_named_grouping_policy("g")

var namedGroupingPolicy = e.GetNamedGroupingPolicy("g");

let named_grouping_policy = e.get_named_grouping_policy("g");

List<List<String>> namedGroupingPolicy = e.getNamedGroupingPolicy("g");
```

GetFilteredNamedGroupingPolicy()

GetFilteredNamedGroupingPolicy gets all the role inheritance rules in the policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0,
"alice")

const namedGroupingPolicy = await
e.getFilteredNamedGroupingPolicy('g', 0, 'alice')

$namedGroupingPolicy = $e->getFilteredNamedGroupingPolicy("g", 0,
"alice");

named_grouping_policy = e.get_filtered_named_grouping_policy("g", 0,
"alice")

var namedGroupingPolicy = e.GetFilteredNamedGroupingPolicy("g", 0,
"alice");

let named_grouping_policy = e.get_filtered_named_groupingPolicy("g",
```

```
List<List<String>> filteredNamedGroupingPolicy =  
e.getFilteredNamedGroupingPolicy("g", 0, "alice");
```

HasPolicy()

HasPolicy determines whether an authorization rule exists.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
hasPolicy := e.HasPolicy("data2_admin", "data2", "read")  
  
const hasPolicy = await e.hasPolicy('data2_admin', 'data2', 'read')  
  
$hasPolicy = $e->hasPolicy('data2_admin', 'data2', 'read');  
  
has_policy = e.has_policy("data2_admin", "data2", "read")  
  
var hasPolicy = e.HasPolicy("data2_admin", "data2", "read");  
  
let has_policy = e.has_policy(vec!["data2_admin".to_owned(),  
"data2".to_owned(), "read".to_owned()]);  
  
boolean hasPolicy = e.hasPolicy("data2_admin", "data2", "read");
```

HasNamedPolicy()

HasNamedPolicy determines whether a named authorization rule exists.

For example:

```
hasNamedPolicy := e.HasNamedPolicy("p", "data2_admin", "data2", "read")  
  
const hasNamedPolicy = await e.hasNamedPolicy('p', 'data2_admin',  
    'data2', 'read')  
  
$hasNamedPolicy = $e->hasNamedPolicy("p", "data2_admin", "data2",  
    "read");  
  
has_named_policy = e.has_named_policy("p", "data2_admin", "data2",  
    "read")  
  
var hasNamedPolicy = e.HasNamedPolicy("p", "data2_admin", "data2",  
    "read");  
  
let has_named_policy = e.has_named_policy("p",  
    vec!["data2_admin".to_owned(), "data2".to_owned(), "read".to_owned()]);  
  
boolean hasNamedPolicy = e.hasNamedPolicy("p", "data2_admin", "data2",  
    "read");
```

AddPolicy()

AddPolicy adds an authorization rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

For example:

```

added := e.AddPolicy('eve', 'data3', 'read')

const p = ['eve', 'data3', 'read']
const added = await e.addPolicy(...p)

$added = $e->addPolicy('eve', 'data3', 'read');

added = e.add_policy("eve", "data3", "read")

var added = e.AddPolicy("eve", "data3", "read");
or
var added = await e.AddPolicyAsync("eve", "data3", "read");

let added = e.add_policy(vec!["eve".to_owned(), "data3".to_owned(),
"read".to_owned()]);

boolean added = e.addPolicy("eve", "data3", "read");

```

AddPolicies()

AddPolicies adds authorization rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

rules := [][] string {
    []string {"jack", "data4", "read"},
```

```

const rules = [
  ['jack', 'data4', 'read'],
  ['katy', 'data4', 'write'],
  ['leyo', 'data4', 'read'],
  ['ham', 'data4', 'write']
];

const areRulesAdded = await e.addPolicies(rules);

rules = [
  ["jack", "data4", "read"],
  ["katy", "data4", "write"],
  ["leyo", "data4", "read"],
  ["ham", "data4", "write"]
]
are_rules_added = e.add_policies(rules)

let rules = vec![
  vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
  vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_added = e.add_policies(rules).await?

String[][] rules = {
  {"jack", "data4", "read"},
  {"katy", "data4", "write"},
  {"leyo", "data4", "read"},
  {"ham", "data4", "write"},
};

boolean areRulesAdded = e.addPolicies(rules);

```

AddPoliciesEx()

AddPoliciesEx adds authorization rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddPolicies, other non-existent rules are added instead of returning false directly

For example:

Go

```
ok, err := e.AddPoliciesEx([][]string{{"user1", "data1", "read"}, {"user2", "data2", "read"}})
```

AddNamedPolicy()

AddNamedPolicy adds an authorization rule to the current named policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

For example:

Go Node.js PHP Python .NET Rust Java

```
added := e.AddNamedPolicy("p", "eve", "data3", "read")  
  
const p = ['eve', 'data3', 'read']  
const added = await e.addNamedPolicy('p', ...p)  
  
$added = $e->addNamedPolicy("p", "eve", "data3", "read");
```

```

added = e.add_named_policy("p", "eve", "data3", "read")

var added = e.AddNamedPolicy("p", "eve", "data3", "read");
or
var added = await e.AddNamedPolicyAsync("p", "eve", "data3", "read");

let added = e.add_named_policy("p", vec!["eve".to_owned(),
"data3".to_owned(), "read".to_owned()]).await?;

boolean added = e.addNamedPolicy("p", "eve", "data3", "read");

```

AddNamedPolicies()

AddNamedPolicies adds authorization rules to the current named policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesAdded := e.AddNamedPolicies("p", rules)

const rules = [

```

```

rules = [
    ["jack", "data4", "read"],
    ["katy", "data4", "write"],
    ["leyo", "data4", "read"],
    ["ham", "data4", "write"]
]
are_rules_added = e.add_named_policies("p", rules)

let rules = vec![
    vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
    vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_added := e.add_named_policies("p", rules).await?;

List<List<String>> rules = Arrays.asList(
    Arrays.asList("jack", "data4", "read"),
    Arrays.asList("katy", "data4", "write"),
    Arrays.asList("leyo", "data4", "read"),
    Arrays.asList("ham", "data4", "write")
);
boolean areRulesAdded = e.addNamedPolicies("p", rules);

```

AddNamedPoliciesEx()

AddNamedPoliciesEx adds authorization rules to the current named policy. If the rule already exists, the rule will not be added. But unlike AddNamedPolicies, other non-existent rules are added instead of returning false directly

For example:

[Go](#)

```
ok, err := e.AddNamedPoliciesEx("p", [][]string{{"user1", "data1",  
"read"}, {"user2", "data2", "read"}})
```

SelfAddPoliciesEx()

SelfAddPoliciesEx adds authorization rules to the current named policy with autoNotifyWatcher disabled. If the rule already exists, the rule will not be added. But unlike SelfAddPolicies, other non-existent rules are added instead of returning false directly

For example:

[Go](#)

```
ok, err := e.SelfAddPoliciesEx("p", "p", [][]string{{"user1", "data1",  
"read"}, {"user2", "data2", "read"}})
```

RemovePolicy()

RemovePolicy removes an authorization rule from the current policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
removed := e.RemovePolicy("alice", "data1", "read")
```

```
const p = ['alice', 'data1', 'read']  
const removed = await e.removePolicy(...p)
```

```

$removed = $e->removePolicy("alice", "data1", "read");

removed = e.remove_policy("alice", "data1", "read")

var removed = e.RemovePolicy("alice", "data1", "read");
or
var removed = await e.RemovePolicyAsync("alice", "data1", "read");

let removed = e.remove_policy(vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removePolicy("alice", "data1", "read");

```

RemovePolicies()

RemovePolicies removes authorization rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesRemoved := e.RemovePolicies(rules)

```

```

const rules = [
  ['jack', 'data4', 'read'],
  ['katy', 'data4', 'write'],
  ['leyo', 'data4', 'read'],
  ['ham', 'data4', 'write']
];

const areRulesRemoved = await e.removePolicies(rules);

rules = [
  ["jack", "data4", "read"],
  ["katy", "data4", "write"],
  ["leyo", "data4", "read"],
  ["ham", "data4", "write"]
]
are_rules_removed = e.remove_policies(rules)

let rules = vec![
  vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
  vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_removed = e.remove_policies(rules).await?;

String[][] rules = {
  {"jack", "data4", "read"},
  {"katy", "data4", "write"},
  {"leyo", "data4", "read"},
  {"ham", "data4", "write"},
};
boolean areRulesRemoved = e.removePolicies(rules);

```

RemoveFilteredPolicy()

RemoveFilteredPolicy removes an authorization rule from the current policy, field filters

can be specified. RemovePolicy removes an authorization rule from the current policy.

For example:

Go Node.js PHP Python .NET Rust Java

```
removed := e.RemoveFilteredPolicy(0, "alice", "data1", "read")

const p = ['alice', 'data1', 'read']
const removed = await e.removeFilteredPolicy(0, ...p)

$removed = $e->removeFilteredPolicy(0, "alice", "data1", "read");

removed = e.remove_filtered_policy(0, "alice", "data1", "read")

var removed = e.RemoveFilteredPolicy("alice", "data1", "read");
or
var removed = await e.RemoveFilteredPolicyAsync("alice", "data1",
"read");

let removed = e.remove_filtered_policy(0, vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removeFilteredPolicy(0, "alice", "data1", "read");
```

RemoveNamedPolicy()

RemoveNamedPolicy removes an authorization rule from the current named policy.

For example:

Go Node.js PHP Python .NET Rust Java

```

removed := e.RemoveNamedPolicy("p", "alice", "data1", "read")

const p = ['alice', 'data1', 'read']
const removed = await e.removeNamedPolicy('p', ...p)

$removed = $e->removeNamedPolicy("p", "alice", "data1", "read");

removed = e.remove_named_policy("p", "alice", "data1", "read")

var removed = e.RemoveNamedPolicy("p", "alice", "data1", "read");
or
var removed = await e.RemoveNamedPolicyAsync("p", "alice", "data1",
"read");

let removed = e.remove_named_policy("p", vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removeNamedPolicy("p", "alice", "data1", "read");

```

RemoveNamedPolicies()

RemoveNamedPolicies removes authorization rules from the current named policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```
rules := [][] string {
```

```

const rules = [
  ['jack', 'data4', 'read'],
  ['katy', 'data4', 'write'],
  ['leyo', 'data4', 'read'],
  ['ham', 'data4', 'write']
];

const areRulesRemoved = await e.removeNamedPolicies('p', rules);

rules = [
  ["jack", "data4", "read"],
  ["katy", "data4", "write"],
  ["leyo", "data4", "read"],
  ["ham", "data4", "write"]
]
are_rules_removed = e.remove_named_policies("p", rules)

let rules = vec![
  vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
  vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];

```

`let areRulesRemoved = e.remove_named_policies("p", rules).await?;`

```

List<List<String>> rules = Arrays.asList(
  Arrays.asList("jack", "data4", "read"),
  Arrays.asList("katy", "data4", "write"),
  Arrays.asList("leyo", "data4", "read"),
  Arrays.asList("ham", "data4", "write")
);
boolean areRulesRemoved = e.removeNamedPolicies("p", rules);

```

RemoveFilteredNamedPolicy()

RemoveFilteredNamedPolicy removes an authorization rule from the current named

policy, field filters can be specified.

For example:

Go Node.js PHP Python .NET Rust Java

```
removed := e.RemoveFilteredNamedPolicy("p", 0, "alice", "data1",
"read")

const p = ['alice', 'data1', 'read']
const removed = await e.removeFilteredNamedPolicy('p', 0, ...p)

$removed = $e->removeFilteredNamedPolicy("p", 0, "alice", "data1",
"read");

removed = e.remove_filtered_named_policy("p", 0, "alice", "data1",
"read")

var removed = e.RemoveFilteredNamedPolicy("p", 0, "alice", "data1",
"read");
or
var removed = e.RemoveFilteredNamedPolicyAsync("p", 0, "alice",
"data1", "read");

let removed = e.remove_filtered_named_policy("p", 0,
vec!["alice".to_owned(), "data1".to_owned(),
"read".to_owned()]).await?;

boolean removed = e.removeFilteredNamedPolicy("p", 0, "alice",
"data1", "read");
```

HasGroupingPolicy()

HasGroupingPolicy determines whether a role inheritance rule exists.

For example:

Go Node.js PHP Python .NET Rust Java

```
has := e.HasGroupingPolicy("alice", "data2_admin")  
  
const has = await e.hasGroupingPolicy('alice', 'data2_admin')  
  
$has = $e->hasGroupingPolicy("alice", "data2_admin");  
  
has = e.has_grouping_policy("alice", "data2_admin")  
  
var has = e.HasGroupingPolicy("alice", "data2_admin");  
  
let has = e.has_grouping_policy(vec!["alice".to_owned(),  
"data2_admin".to_owned()]);  
  
boolean has = e.hasGroupingPolicy("alice", "data2_admin");
```

HasNamedGroupingPolicy()

HasNamedGroupingPolicy determines whether a named role inheritance rule exists.

For example:

Go Node.js PHP Python .NET Rust Java

```
has := e.HasNamedGroupingPolicy("g", "alice", "data2_admin")  
  
const has = await e.hasNamedGroupingPolicy('g', 'alice', 'data2_admin')
```

```
$has = $e->hasNamedGroupingPolicy("g", "alice", "data2_admin");  
  
has = e.has_named_grouping_policy("g", "alice", "data2_admin")  
  
var has = e.HasNamedGroupingPolicy("g", "alice", "data2_admin");  
  
let has = e.has_named_grouping_policy("g", vec!["alice".to_owned(),  
"data2_admin".to_owned()]);  
  
boolean has = e.hasNamedGroupingPolicy("g", "alice", "data2_admin");
```

AddGroupingPolicy()

AddGroupingPolicy adds a role inheritance rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
added := e.AddGroupingPolicy("group1", "data2_admin")  
  
const added = await e.addGroupingPolicy('group1', 'data2_admin')  
  
$added = $e->addGroupingPolicy("group1", "data2_admin");  
  
added = e.add_grouping_policy("group1", "data2_admin")  
  
var added = e.AddGroupingPolicy("group1", "data2_admin");  
or
```

```
let added = e.add_grouping_policy(vec!["group1".to_owned(),
"data2_admin".to_owned()]).await?;

boolean added = e.addGroupingPolicy("group1", "data2_admin");
```

AddGroupingPolicies()

AddGroupingPolicies adds role inheritance rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all authorization the rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesAdded := e.AddGroupingPolicies(rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesAdded = await e.addGroupingPolicies(groupingRules);

rules = [
    ["ham", "data4_admin"],
```

```

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];

let areRulesAdded = e.add_grouping_policies(rules).await?;

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesAdded = e.addGroupingPolicies(groupingRules);

```

AddGroupingPoliciesEx()

AddGroupingPoliciesEx adds role inheritance rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddGroupingPolicies, other non-existent rules are added instead of returning false directly

For example:

[Go](#)

```
ok, err := e.AddGroupingPoliciesEx([][]string{{"user1", "member"}, {"user2", "member"}})
```

AddNamedGroupingPolicy()

AddNamedGroupingPolicy adds a named role inheritance rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

For example:

```
added := e.AddNamedGroupingPolicy("g", "group1", "data2_admin")

const added = await e.addNamedGroupingPolicy('g', 'group1',
    'data2_admin')

$added = $e->addNamedGroupingPolicy("g", "group1", "data2_admin");

added = e.add_named_grouping_policy("g", "group1", "data2_admin")

var added = e.AddNamedGroupingPolicy("g", "group1", "data2_admin");
or
var added = await e.AddNamedGroupingPolicyAsync("g", "group1",
    "data2_admin");

let added = e.add_named_grouping_policy("g", vec![ "group1".to_owned(),
    "data2_admin".to_owned()]).await?;

boolean added = e.addNamedGroupingPolicy("g", "group1", "data2_admin");
```

AddNamedGroupingPolicies()

AddNamedGroupingPolicies adds named role inheritance rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

For example:

```

rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesAdded := e.AddNamedGroupingPolicies("g", rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesAdded = await e.addNamedGroupingPolicies('g',
groupingRules);

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]

are_rules_added = e.add_named_grouping_policies("g", rules)

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];

let are_rules_added = e.add_named_grouping_policies("g", rules).await?;

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesAdded = e.addNamedGroupingPolicies("g", groupingRules);

```

AddNamedGroupingPoliciesEx()

AddNamedGroupingPoliciesEx adds named role inheritance rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddNamedGroupingPolicies, other non-existent rules are added instead of returning false directly

For example:

Go

```
ok, err := e.AddNamedGroupingPoliciesEx("g", [][]string{{"user1", "member"}, {"user2", "member"}})
```

RemoveGroupingPolicy()

RemoveGroupingPolicy removes a role inheritance rule from the current policy.

For example:

Go Node.js PHP Python .NET Rust Java

```
removed := e.RemoveGroupingPolicy("alice", "data2_admin")  
  
const removed = await e.removeGroupingPolicy('alice', 'data2_admin')  
  
$removed = $e->removeGroupingPolicy("alice", "data2_admin");  
  
removed = e.remove_grouping_policy("alice", "data2_admin")  
  
var removed = e.RemoveGroupingPolicy("alice", "data2_admin");
```

```
let removed = e.remove_grouping_policy(vec!["alice".to_owned(),
"data2_admin".to_owned()]).await?;

boolean removed = e.removeGroupingPolicy("alice", "data2_admin");
```

RemoveGroupingPolicies()

RemoveGroupingPolicies removes role inheritance rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

For example:

[Go](#) [Node.js](#) [Rust](#) [Python](#) [Java](#)

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesRemoved := e.RemoveGroupingPolicies(rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesRemoved = await e.removeGroupingPolicies(groupingRules);

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
```

```

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]

are_rules_removed = e.remove_grouping_policies(rules)

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesRemoved = e.removeGroupingPolicies(groupingRules);

```

RemoveFilteredGroupingPolicy()

RemoveFilteredGroupingPolicy removes a role inheritance rule from the current policy, field filters can be specified.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```

removed := e.RemoveFilteredGroupingPolicy(0, "alice")

const removed = await e.removeFilteredGroupingPolicy(0, 'alice')

$removed = $e->removeFilteredGroupingPolicy(0, "alice");

removed = e.remove_filtered_grouping_policy(0, "alice")

var removed = e.RemoveFilteredGroupingPolicy(0, "alice");
or
var removed = await e.RemoveFilteredGroupingPolicyAsync(0, "alice");

```

```
let removed = e.remove_filtered_grouping_policy(0,
vec!["alice".to_owned()]).await?;

boolean removed = e.removeFilteredGroupingPolicy(0, "alice");
```

RemoveNamedGroupingPolicy()

RemoveNamedGroupingPolicy removes a role inheritance rule from the current named policy.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
removed := e.RemoveNamedGroupingPolicy("g", "alice")

const removed = await e.removeNamedGroupingPolicy('g', 'alice')

$removed = $e->removeNamedGroupingPolicy("g", "alice");

removed = e.remove_named_grouping_policy("g", "alice", "data2_admin")

var removed = e.RemoveNamedGroupingPolicy("g", "alice");
or
var removed = await e.RemoveNamedGroupingPolicyAsync("g", "alice");

let removed = e.remove_named_grouping_policy("g",
vec!["alice".to_owned()]).await?;

boolean removed = e.removeNamedGroupingPolicy("g", "alice");
```

RemoveNamedGroupingPolicies()

RemoveNamedGroupingPolicies removes named role inheritance rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesRemoved := e.RemoveNamedGroupingPolicies("g", rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesRemoved = await e.removeNamedGroupingPolicies('g',
groupingRules);

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]
are_rules_removed = e.remove_named_grouping_policies("g", rules)
```

```

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];

let are_rules_removed = e.remove_named_grouping_policies("g",
rules).await?;

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesRemoved = e.removeNamedGroupingPolicies("g",
groupingRules);

```

RemoveFilteredNamedGroupingPolicy()

RemoveFilteredNamedGroupingPolicy removes a role inheritance rule from the current named policy, field filters can be specified.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```

removed := e.RemoveFilteredNamedGroupingPolicy("g", 0, "alice")

const removed = await e.removeFilteredNamedGroupingPolicy('g', 0,
'alice')

$removed = $e->removeFilteredNamedGroupingPolicy("g", 0, "alice");

removed = e.remove_filtered_named_grouping_policy("g", 0, "alice")

```

```
var removed = e.RemoveFilteredNamedGroupingPolicy("g", 0, "alice");
or
var removed = await e.RemoveFilteredNamedGroupingPolicyAsync("g", 0,
"alice");

let removed = e.remove_filtered_named_groupingPolicy("g", 0,
vec!["alice".to_owned()]).await?;

boolean removed = e.removeFilteredNamedGroupingPolicy("g", 0, "alice");
```

UpdatePolicy()

UpdatePolicy update a old policy to new policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
updated, err := e.UpdatePolicy([]string{"eve", "data3", "read"}, 
[]string{"eve", "data3", "write"})

const update = await e.updatePolicy(["eve", "data3", "read"], ["eve",
"data3", "write"]);

updated = e.update_policy(["eve", "data3", "read"], ["eve", "data3",
"write"])

boolean updated = e.updatePolicy(Arrays.asList("eve", "data3",
"read"), Arrays.asList("eve", "data3", "write"));
```

UpdatePolicies()

UpdatePolicies updates all old policies to new policies.

For example:

[Go](#) [Python](#)

```
updated, err := e.UpdatePolicies([][]string{{"eve", "data3", "read"},  
{"jack", "data3", "read"}}, [][]string{{"eve", "data3", "write"},  
{"jack", "data3", "write"}})  
  
old_rules = [["eve", "data3", "read"], ["jack", "data3", "read"]]  
new_rules = [["eve", "data3", "write"], ["jack", "data3", "write"]]  
  
updated = e.update_policies(old_rules, new_rules)
```

UpdateNamedPolicy()

UpdateNamedPolicy update a old policy to new policy in the named policy.

For example:

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
updated, err := e.UpdateNamedPolicy("p", []string{"eve", "data3",  
"read"}, []string{"eve", "data3", "write"})  
  
const update = await e.updateNamedPolicy('p', ["eve", "data3",  
"read"], ["eve", "data3", "write"]);
```

```
updated = e.update_named_policy("p", ["eve", "data3", "read"], ["eve",  
"data3", "write"])
```



```
boolean updated = e.updateNamedPolicy("p", Arrays.asList("eve",  
"data3", "read"), Arrays.asList("eve", "data3", "write"));
```

UpdateNamedPolicies()

UpdateNamedPolicies updates all old policies to new policies in the named policy.

For example:

[Go](#) [Python](#)

```
updated, err := e.UpdateNamedPolicies("p", [][]string{{"eve", "data3",  
"read"}, {"jack", "data3", "read"}}, [][]string{{"eve", "data3",  
"write"}, {"jack", "data3", "write"}})
```



```
old_rules = [[{"eve", "data3", "read"}, {"jack", "data3", "read"}]]  
new_rules = [[{"eve", "data3", "write"}, {"jack", "data3", "write"}]]
```



```
updated = e.update_named_policies("p", old_rules, new_rules)
```

AddFunction()

AddFunction adds a customized function.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [Rust](#) [Java](#)

```

func CustomFunction(key1 string, key2 string) bool {
    if key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data/:resource" {
        return true
    } else if key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data2/:id/using/:resId" {
        return true
    } else {
        return false
    }
}

func CustomFunctionWrapper(args ...interface{}) (interface{}, error) {
    key1 := args[0].(string)
    key2 := args[1].(string)

    return bool(CustomFunction(key1, key2)), nil
}

e.AddFunction("keyMatchCustom", CustomFunctionWrapper)

function customFunction(key1, key2){
    if(key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data/:resource") {
        return true
    } else if(key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data2/:id/using/:resId") {
        return true
    } else {
        return false
    }
}

e.addFunction("keyMatchCustom", customFunction);

func customFunction($key1, $key2) {
    if ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data/:resource") {

```

```

def custom_function(key1, key2):
    return ((key1 == "/alice_data2/myid/using/res_id" and key2 ==
"/alice_data/:resource") or (key1 == "/alice_data2/myid/using/res_id"
and key2 == "/alice_data2/:id/using/:resId"))

e.add_function("keyMatchCustom", custom_function)

fn custom_function(key1: SString, key2: String) {
    key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data/:resource" || key1 == "/alice_data2/myid/using/res_id" &&
key2 == "/alice_data2/:id/using/:resId"
}

e.add_function("keyMatchCustom", custom_function);

public static class CustomFunc extends CustomFunction {
    @Override
    public AviatorObject call(Map<String, Object> env, AviatorObject
arg1, AviatorObject arg2) {
        String key1 = FunctionUtils.getStringValue(arg1, env);
        String key2 = FunctionUtils.getStringValue(arg2, env);
        if (key1.equals("/alice_data2/myid/using/res_id") &&
key2.equals("/alice_data/:resource")) {
            return AviatorBoolean.valueOf(true);
        } else if (key1.equals("/alice_data2/myid/using/res_id") &&
key2.equals("/alice_data2/:id/using/:resId")) {
            return AviatorBoolean.valueOf(true);
        } else {
            return AviatorBoolean.valueOf(false);
        }
    }

    @Override
    public String getName() {
        return "keyMatchCustom";
    }
}

```

LoadFilteredPolicy()

LoadFilteredPolicy loads filtered policies from file/database.

For example:

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
err := e.LoadFilteredPolicy()

const ok = await e.loadFilteredPolicy();

class Filter:
    P = []
    G = []

adapter =
casbin.persist.adapters.FilteredAdapter("rbac_with_domains_policy.csv")
e = casbin.Enforcer("rbac_with_domains_model.conf", adapter)
filter = Filter()
filter.P = ["", "domain1"]
filter.G = ["", "", "domain1"]
e.load_filtered_policy(filter)

e.loadFilteredPolicy(new String[] { "", "domain1" });
```

LoadIncrementalFilteredPolicy()

LoadIncrementalFilteredPolicy append a filtered policy from file/database.

For example:

[Go](#) [Node.js](#) [Python](#)

```
err := e.LoadIncrementalFilteredPolicy()

const ok = await e.loadIncrementalFilteredPolicy();

adapter =
casbin.persist.adapters.FilteredAdapter("rbac_with_domains_policy.csv")
e = casbin.Enforcer("rbac_with_domains_model.conf", adapter)
filter = Filter()
filter.P = ['', "domain1"]
filter.G = ['', "", "domain1"]
e.load_increment_filtered_policy(filter)
```

UpdateGroupingPolicy()

UpdateGroupingPolicy updates oldRule to newRule in `g` section

For example:

[Go](#) [Java](#)

```
succeed, err := e.UpdateGroupingPolicy([]string{"data3_admin",
"data4_admin"}, []string{"admin", "data4_admin"})

boolean succeed = e.updateGroupingPolicy(Arrays.asList("data3_admin",
"data4_admin"), Arrays.asList("admin", "data4_admin"));
```

UpdateNamedGroupingPolicy()

UpdateNamedGroupingPolicy updates oldRule named `ptype` to newRule in `g` section

For example:

[Go](#) [Java](#)

```
succeed, err := e.UpdateGroupingPolicy("g1", []string{"data3_admin",  
"data4_admin"}, []string{"admin", "data4_admin"})  
  
boolean succeed = e.updateNamedGroupingPolicy("g1",  
Arrays.asList("data3_admin", "data4_admin"), Arrays.asList("admin",  
"data4_admin"));
```

SetFieldIndex()

SetFieldIndex support customization of conventional name and position of `sub`, `obj`, `domain` and `priority`.

```
[policy_definition]  
p = customized_priority, obj, act, eft, subject
```

For example:

Go

```
e.SetFieldIndex("p", constant.PriorityIndex, 0)  
e.SetFieldIndex("p", constant.SubjectIndex, 4)
```

RBAC API

A more friendly API for RBAC. This API is a subset of Management API. The RBAC users could use this API to simplify the code.

Reference

global variable `e` is Enforcer instance.

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e, err := NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

```
const e = await newEnforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv')
```

```
$e = new Enforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv');
```

```
e = casbin.Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

```
var e = new Enforcer("path/to/model.conf", "path/to/policy.csv");
```

```
let mut e = Enforcer::new("examples/rbac_model.conf", "examples/rbac_policy.csv").await?;
```

```
Enforcer e = new Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv");
```

GetRolesForUser()

GetRolesForUser gets the roles that a user has.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

[Go](#)

```
res := e.GetRolesForUser("alice")
```

[Node.js](#)

```
const res = await e.getRolesForUser('alice')
```

[PHP](#)

```
$res = $e->getRolesForUser("alice");
```

[Python](#)

```
roles = e.get_roles_for_user("alice")
```

[.NET](#)

```
var res = e.GetRolesForUser("alice");
```

[Rust](#)

```
let roles = e.get_roles_for_user("alice", None); // No domain
```

[Java](#)

```
List<String> res = e.getRolesForUser("alice");
```

GetUsersForRole()

GetUsersForRole gets the users that has a role.

For example:

Go

Node.js

PHP

Python

.NET

Rust

Java

```
res := e.GetUsersForRole("data1_admin")  
  
const res = await e.getUsersForRole('data1_admin')  
  
$res = $e->getUsersForRole("data1_admin");  
  
users = e.get_users_for_role("data1_admin")  
  
var res = e.GetUsersForRole("data1_admin");  
  
let users = e.get_users_for_role("data1_admin", None); // No  
domain  
  
List<String> res = e.getUsersForRole("data1_admin");
```

HasRoleForUser()

HasRoleForUser determines whether a user has a role.

For example:

Go

Node.js

PHP

Python

.NET

Rust

Java

```
res := e.HasRoleForUser("alice", "data1_admin")
```

```
const res = await e.hasRoleForUser('alice', 'data1_admin')

$res = $e->hasRoleForUser("alice", "data1_admin");

has = e.has_role_for_user("alice", "data1_admin")

var res = e.HasRoleForUser("alice", "data1_admin");

let has = e.has_role_for_user("alice", "data1_admin", None); //  
No domain

boolean res = e.hasRoleForUser("alice", "data1_admin");
```

AddRoleForUser()

AddRoleForUser adds a role for a user. Returns false if the user already has the role (aka not affected).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.AddRoleForUser("alice", "data2_admin")

await e.addRoleForUser('alice', 'data2_admin')

$e->addRoleForUser("alice", "data2_admin");
```

```
e.add_role_for_user("alice", "data2_admin")  
  
var added = e.AddRoleForUser("alice", "data2_admin");  
or  
var added = await e.AddRoleForUserAsync("alice", "data2_admin");  
  
let added = e.add_role_for_user("alice", "data2_admin",  
None).await?; // No domain  
  
boolean added = e.addRoleForUser("alice", "data2_admin");
```

AddRolesForUser()

AddRolesForUser adds multiple roles for a user. Returns false if the user already has one of these roles (aka not affected).

For example:

[Go](#) [Node.js](#) [Rust](#)

```
var roles = []string{"data2_admin", "data1_admin"}  
e.AddRolesForUser("alice", roles)  
  
const roles = ["data1_admin", "data2_admin"];  
roles.map((role) => e.addRoleForUser("alice", role));  
  
let roles = vec!["data1_admin".to_owned(),  
"data2_admin".to_owned()];  
let all_added = e.add_roles_for_user("alice", roles,
```

DeleteRoleForUser()

DeleteRoleForUser deletes a role for a user. Returns false if the user does not have the role (aka not affected).

For example:

Go

Node.js

PHP

Python

.NET

Rust

Java

```
e.DeleteRoleForUser("alice", "data1_admin")  
  
await e.deleteRoleForUser('alice', 'data1_admin')  
  
$e->deleteRoleForUser("alice", "data1_admin");  
  
e.delete_role_for_user("alice", "data1_admin")  
  
var deleted = e.DeleteRoleForUser("alice", "data1_admin");  
or  
var deleted = await e.DeleteRoleForUser("alice", "data1_admin");  
  
let deleted = e.delete_role_for_user("alice", "data1_admin",  
None).await?; // No domain  
  
boolean deleted = e.deleteRoleForUser("alice", "data1_admin");
```

DeleteRolesForUser()

DeleteRolesForUser deletes all roles for a user. Returns false if the user does not

have any roles (aka not affected).

For example:

Go Node.js PHP Python .NET Rust Java

```
e.DeleteRolesForUser("alice")

await e.deleteRolesForUser('alice')

$e->deleteRolesForUser("alice");

e.delete_roles_for_user("alice")

var deletedAtLeastOne = e.DeleteRolesForUser("alice");
or
var deletedAtLeastOne = await
e.DeleteRolesForUserAsync("alice");

let deleted_at_least_one = e.delete_roles_for_user("alice",
None).await?; // No domain

boolean deletedAtLeastOne = e.deleteRolesForUser("alice");
```

DeleteUser()

DeleteUser deletes a user. Returns false if the user does not exist (aka not affected).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.DeleteUser("alice")  
  
await e.deleteUser('alice')  
  
$e->deleteUser("alice");  
  
e.delete_user("alice")  
  
var deleted = e.DeleteUser("alice");  
or  
var deleted = await e.DeleteUserAsync("alice");  
  
let deleted = e.delete_user("alice").await?;  
  
boolean deleted = e.deleteUser("alice");
```

DeleteRole()

DeleteRole deletes a role.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.DeleteRole("data2_admin")
```

```
await e.deleteRole("data2_admin")  
  
$e->deleteRole("data2_admin");  
  
e.delete_role("data2_admin")  
  
var deleted = e.DeleteRole("data2_admin");  
or  
var deleted = await e.DeleteRoleAsync("data2_admin");  
  
let deleted = e.delete_role("data2_admin").await?;  
  
e.deleteRole("data2_admin");
```

DeletePermission()

DeletePermission deletes a permission. Returns false if the permission does not exist (aka not affected).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.DeletePermission("read")  
  
await e.deletePermission('read')  
  
$e->deletePermission("read");
```

```
e.delete_permission("read")

var deleted = e.DeletePermission("read");
or
var deleted = await e.DeletePermissionAsync("read");

let deleted =
e.delete_permission(vec!["read".to_owned()]).await?;

boolean deleted = e.deletePermission("read");
```

AddPermissionForUser()

AddPermissionForUser adds a permission for a user or role. Returns false if the user or role already has the permission (aka not affected).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.AddPermissionForUser("bob", "read")

await e.addPermissionForUser('bob', 'read')

$e->addPermissionForUser("bob", "read");

e.add_permission_for_user("bob", "read")
```

```
var added = e.AddPermissionForUser("bob", "read");
or
var added = await e.AddPermissionForUserAsync("bob", "read");

let added = e.add_permission_for_user("bob",
vec!["read".to_owned()]).await?;

boolean added = e.addPermissionForUser("bob", "read");
```

AddPermissionsForUser()

AddPermissionsForUser adds multiple permissions for a user or role. Returns false if the user or role already has one of the permissions (aka not affected).

For example:

[Go](#) [Node.js](#) [Rust](#)

```
var permissions = [][]string{{"data1",
"read"}, {"data2", "write"}}
for i := 0; i < len(permissions); i++ {
    e.AddPermissionsForUser("alice", permissions[i])
}

const permissions = [
    ["data1", "read"],
    ["data2", "write"],
];

permissions.map((permission) => e.addPermissionForUser("bob",
```

```
let permissions = vec![
    vec!["data1".to_owned(), "read".to_owned()],
    vec!["data2".to_owned(), "write".to_owned()],
];
let all_added = e.add_permissions_for_user("bob",
permissions).await?;
```

DeletePermissionForUser()

DeletePermissionForUser deletes a permission for a user or role. Returns false if the user or role does not have the permission (aka not affected).

For example:

Go Node.js PHP Python .NET Rust Java

```
e.DeletePermissionForUser("bob", "read")
```

```
await e.deletePermissionForUser("bob", "read")
```

```
$e->deletePermissionForUser("bob", "read");
```

```
e.delete_permission_for_user("bob", "read")
```

```
var deleted = e.DeletePermissionForUser("bob", "read");
```

or

```
var deleted = await e.DeletePermissionForUserAsync("bob",
"read");
```

```
let deleted = e.delete_permission_for_user("bob",
vec!["read"].to_owned()).await?;

boolean deleted = e.deletePermissionForUser("bob", "read");
```

DeletePermissionsForUser()

DeletePermissionsForUser deletes permissions for a user or role. Returns false if the user or role does not have any permissions (aka not affected).

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.DeletePermissionsForUser("bob")

await e.deletePermissionsForUser('bob')

$e->deletePermissionsForUser("bob");

e.delete_permissions_for_user("bob")

var deletedAtLeastOne = e.DeletePermissionsForUser("bob");
or
var deletedAtLeastOne = await
e.DeletePermissionsForUserAsync("bob");

let deleted_at_least_one =
e.delete_permissions_for_user("bob").await?;
```

```
boolean deletedAtLeastOne = e.deletePermissionForUser("bob");
```

GetPermissionsForUser()

GetPermissionsForUser gets permissions for a user or role.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Java](#)

```
e.GetPermissionsForUser("bob")
```

```
await e.getPermissionsForUser('bob')
```

```
$e->getPermissionsForUser("bob");
```

```
e.get_permissions_for_user("bob")
```

```
var permissions = e.GetPermissionsForUser("bob");
```

```
List<List<String>> permissions = e.getPermissionsForUser("bob");
```

GetNamedPermissionsForUser()

GetNamedPermissionsForUser gets permissions for a user or role by named policy.

For example:

```
p, alice, data1, read  
p, bob, data2, write  
p2, admin, create  
g, alice, admin
```

GetNamedPermissionsForUser("p", "alice") will return `[["alice", "data1", "read"]]`. GetNamedPermissionsForUser("p2", "alice") will return `[["admin", "create"]]`.

[Go](#) [Python](#)

```
permissions, err := e.GetNamedPermissionsForUser("p", "alice")  
  
permissions = e.get_named_permissions_for_user("p", "alice")
```

HasPermissionForUser()

HasPermissionForUser determines whether a user has a permission.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.HasPermissionForUser("alice", []string{"read"})  
  
await e.hasPermissionForUser('alice', 'read')  
  
$e->hasPermissionForUser("alice", []string{"read"});
```

```
has = e.has_permission_for_user("alice", "read")  
  
var has = e.HasPermissionForUser("bob", "read");  
  
let has = e.has_permission_for_user("alice",  
vec!["data1".to_owned(), "read".to_owned()]);  
  
boolean has = e.hasPermissionForUser("alice", "read");
```

GetImplicitRolesForUser()

GetImplicitRolesForUser gets implicit roles that a user has. Compared to GetRolesForUser(), this function retrieves indirect roles besides direct roles.

For example:

```
g, alice, role:admin  
g, role:admin, role:user
```

GetRolesForUser("alice") can only get: ["role:admin"].

But GetImplicitRolesForUser("alice") will get: ["role:admin", "role:user"].

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.GetImplicitRolesForUser("alice")
```

```
await e.getImplicitRolesForUser("alice")  
  
$e->getImplicitRolesForUser("alice");  
  
e.get_implicit_roles_for_user("alice")  
  
var implicitRoles = e.GetImplicitRolesForUser("alice");  
  
e.get_implicit_roles_for_user("alice", None); // No domain  
  
List<String> implicitRoles = e.getImplicitRolesForUser("alice");
```

GetNamedImplicitRolesForUser()

GetNamedImplicitRolesForUser gets implicit roles that a user has by named policy.

For example:

```
g, alice, admin  
g, admin, super_admin  
g2, alice, user  
g2, user, guest
```

GetNamedImplicitRolesForUser("g", "alice") will return `["admin", "super_admin"]`. GetNamedImplicitRolesForUser("g2", "alice") will return `["user", "guest"]`.

[Go](#) [Python](#)

```
roles, err := e.GetNamedImplicitRolesForUser("g", "alice")
```

```
roles = e.get_named_implicit_roles_for_user("g", "alice")
```

GetImplicitUsersForRole()

GetImplicitUsersForRole gets all users inheriting the role. Compared to GetUsersForRole(), this function retrieves indirect users.

For example:

```
g, alice, role:admin  
g, role:admin, role:user
```

GetUsersForRole("role:user") can only get: ["role:admin"].

But GetImplicitUsersForRole("role:user") will get: ["role:admin", "alice"].

For example:

[Go](#) [Node.js](#) [Java](#)

```
users := e.GetImplicitUsersForRole("role:user")
```

```
const users = e.getImplicitUsersForRole("role:user");
```

```
List<String> users = e.getImplicitUsersForRole("role:user");
```

GetImplicitPermissionsForUser()

GetImplicitPermissionsForUser gets implicit permissions for a user or role. Compared to GetPermissionsForUser(), this function retrieves permissions for inherited roles.

For example:

```
p, admin, data1, read  
p, alice, data2, read  
g, alice, admin
```

GetPermissionsForUser("alice") can only get: `[[{"alice", "data2", "read"}]]`.

But GetImplicitPermissionsForUser("alice") will get: `[[{"admin", "data1", "read"}, {"alice", "data2", "read"}]]`.

For example:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.GetImplicitPermissionsForUser("alice")  
  
await e.getImplicitPermissionsForUser("alice")  
  
$e->getImplicitPermissionsForUser("alice");  
  
e.get_implicit_permissions_for_user("alice")  
  
var implicitPermissions =
```

```
e.get_implicit_permissions_for_user("alice", None); // No domain  
  
List<List<String>> implicitPermissions =  
e.getImplicitPermissionsForUser("alice");
```

GetNamedImplicitPermissionsForUser()

GetNamedImplicitPermissionsForUser gets implicit permissions for a user or role by named policy Compared to GetImplicitPermissionsForUser(), this function allow you to specify the policy name.

For example:

```
p, admin, data1, read  
p2, admin, create  
g, alice, admin
```

GetImplicitPermissionsForUser("alice") only get: [[{"admin": "data1", "read"}]], whose policy is default "p"

But you can specify the policy as "p2" to get: [[{"admin": "create"}]] by GetNamedImplicitPermissionsForUser("p2","alice")

For example:

[Go](#) [Python](#)

```
e.GetNamedImplicitPermissionsForUser("p2", "alice")
```

```
e.get_named_implicit_permissions_for_user("p2", "alice")
```

GetDomainsForUser()

GetDomainsForUser gets all domains which a user has.

For example:

```
p, admin, domain1, data1, read
p, admin, domain2, data2, read
p, admin, domain2, data2, write
g, alice, admin, domain1
g, alice, admin, domain2
```

GetDomainsForUser("alice") could get ["domain1", "domain2"]

For example:

[Go](#)

```
result, err := e.GetDomainsForUser("alice")
```

GetImplicitResourcesForUser()

GetImplicitResourcesForUser returns all policies that should be true for user.

For example:

```
p, alice, data1, read  
p, bob, data2, write  
p, data2_admin, data2, read  
p, data2_admin, data2, write  
  
g, alice, data2_admin
```

GetImplicitResourcesForUser("alice") will return `[[alice data1 read] [alice data2 read] [alice data2 write]]`

[Go](#)

```
resources, err := e.GetImplicitResourcesForUser("alice")
```

GetImplicitUsersForPermission()

GetImplicitUsersForPermission gets implicit users for a permission.

For example:

```
p, admin, data1, read  
p, bob, data1, read  
g, alice, admin
```

GetImplicitUsersForPermission("data1", "read") will return: `["alice", "bob"]`.

Note: only users will be returned, roles (2nd arg in "g") will be excluded.

[Go](#)

```
users, err := e.GetImplicitUsersForPermission("data1", "read")
```

GetImplicitObjectPatternsForUser()

GetImplicitObjectPatternsForUser returns all object patterns (with wildcards) that a user has for a given domain and action.

For example:

```
p, admin, chronicle/123, location/*, read
p, user, chronicle/456, location/789, read
g, alice, admin
g, bob, user
```

GetImplicitObjectPatternsForUser("alice", "chronicle/123", "read") will return `["location/*"]`. GetImplicitObjectPatternsForUser("bob", "chronicle/456", "read") will return `["location/789"]`.

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
patterns, err := e.GetImplicitObjectPatternsForUser("alice",
"chronicle/123", "read")
```

```
const patterns = await
e.getImplicitObjectPatternsForUser("alice", "chronicle/123",
"read")
```

```
$patterns = $e->getImplicitObjectPatternsForUser("alice",
"chronicle/123", "read");
```

```
patterns = e.get_implicit_object_patterns_for_user("alice",
"chronicle/123", "read")

var patterns = e.GetImplicitObjectPatternsForUser("alice",
"chronicle/123", "read");

let patterns = e.get_implicit_object_patterns_for_user("alice",
"chronicle/123", "read").await?;

List<String> patterns =
e.getImplicitObjectPatternsForUser("alice", "chronicle/123",
"read");
```

GetAllowedObjectConditions()

GetAllowedObjectConditions returns a string array of object conditions that the user can access.

For example:

```
p, alice, r.obj.price < 25, read
p, admin, r.obj.category_id = 2, read
p, bob, r.obj.author = bob, write

g, alice, admin
```

e.GetAllowedObjectConditions("alice", "read", "r.obj.") will return ["price < 25", "category_id = 2"], nil

Note:

0. prefix: You can customize the prefix of the object conditions, and "r.obj." is commonly used as a prefix. After removing the prefix, the remaining part is the condition of the object. If there is an obj policy that does not meet the prefix requirement, an `errors.ERR_OBJ_CONDITION` will be returned.
1. If the 'objectConditions' array is empty, return `errors.ERR_EMPTY_CONDITION` This error is returned because some data adapters' ORM return full table data by default when they receive an empty condition, which tends to behave contrary to expectations.(e.g. GORM) If you are using an adapter that does not behave like this, you can choose to ignore this error.

Go

```
conditions, err := e.GetAllowedObjectConditions("alice",
"read", "r.obj.")
```

GetImplicitUsersForResource()

GetImplicitUsersForResource return implicit user based on resource.

For example:

```
p, alice, data1, read
p, bob, data2, write
p, data2_admin, data2, read
p, data2_admin, data2, write
g, alice, data2_admin
```

GetImplicitUsersForResource("data2") will return `[["bob", "data2", "write"], ["alice", "data2", "read"] ["alice", "data2", "write"]], nil`.

GetImplicitUsersForResource("data1") will return `[["alice", "data1", "read"]]`, nil.

[Go](#)

```
ImplicitUsers, err := e.GetImplicitUsersForResource("data2")
```

 NOTE

Only users will be returned, roles (2nd arg in "g") will be excluded.

GetNamedImplicitUsersForResource()

GetNamedImplicitUsersForResource return implicit user based on resource with named policy support. This function handles resource role relationships through named policies (e.g., g2, g3, etc.).

For example:

```
p, admin_group, admin_data, *
g, admin, admin_group
g2, app, admin_data
```

GetNamedImplicitUsersForResource("g2", "app") will return users who have access to admin_data through g2 relationship.

[Go](#)

```
ImplicitUsers, err := e.GetNamedImplicitUsersForResource("g2",
"app")
```


RBAC with Domains API

A more user-friendly API for RBAC with domains. This API is a subset of the Management API. RBAC users can use this API to simplify their code.

Reference

The global variable `e` represents the Enforcer instance.

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e, err := NewEnforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```
const e = await newEnforcer('examples/
rbac_with_domains_model.conf', 'examples/
rbac_with_domains_policy.csv')
```

```
$e = new Enforcer('examples/rbac_with_domains_model.conf',
'examples/rbac_with_domains_policy.csv');
```

```
e = casbin.Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```
var e = new Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv");
```

```
let mut e = Enforcer::new("examples/
rbac_with_domains_model.conf", "examples/
```

```
Enforcer e = new Enforcer("examples/  
rbac_with_domains_model.conf", "examples/  
rbac_with_domains_policy.csv");
```

GetUsersForRoleInDomain()

The `GetUsersForRoleInDomain()` function retrieves the users that have a role within a domain.

For example:

[Go](#) [Node.js](#) [Python](#)

```
res := e.GetUsersForRoleInDomain("admin", "domain1")
```

```
const res = e.getUsersForRoleInDomain("admin", "domain1")
```

```
res = e.get_users_for_role_in_domain("admin", "domain1")
```

GetRolesForUserInDomain()

The `GetRolesForUserInDomain()` function retrieves the roles that a user has within a domain.

For example:

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
res := e.GetRolesForUserInDomain("admin", "domain1")  
  
const res = e.getRolesForUserInDomain("alice", "domain1")  
  
res = e.get_roles_for_user_in_domain("alice", "domain1")  
  
List<String> res = e.getRolesForUserInDomain("admin",  
"domain1");
```

GetPermissionsForUserInDomain()

The `GetPermissionsForUserInDomain()` function retrieves the permissions for a user or role within a domain.

For example:

[Go](#) [Java](#)

```
res := e.GetPermissionsForUserInDomain("alice", "domain1")  
  
List<List<String>> res =  
e.getPermissionsForUserInDomain("alice", "domain1");
```

AddRoleForUserInDomain()

The `AddRoleForUserInDomain()` function adds a role for a user within a domain. It returns `false` if the user already has the role (no changes made).

For example:

[Go](#) [Python](#) [Java](#)

```
ok, err := e.AddRoleForUserInDomain("alice", "admin", "domain1")  
  
ok = e.add_role_for_user_in_domain("alice", "admin", "domain1")  
  
boolean ok = e.addRoleForUserInDomain("alice", "admin",  
"domain1");
```

DeleteRoleForUserInDomain()

The `DeleteRoleForUserInDomain()` function removes a role for a user within a domain. It returns `false` if the user does not have the role (no changes made).

For example:

[Go](#) [Java](#)

```
ok, err := e.DeleteRoleForUserInDomain("alice", "admin",  
"domain1")  
  
boolean ok = e.deleteRoleForUserInDomain("alice", "admin",  
"domain1");
```

DeleteRolesForUserInDomain()

The `DeleteRolesForUserInDomain()` function removes all roles for a user within a domain. It returns `false` if the user does not have any roles (no changes made).

For example:

[Go](#)

```
ok, err := e.DeleteRolesForUserInDomain("alice", "domain1")
```

GetAllUsersByDomain()

The `GetAllUsersByDomain()` function retrieves all users associated with the given domain. It returns an empty string array if no domain is defined in the model.

For example:

[Go](#)

```
res := e.GetAllUsersByDomain("domain1")
```

DeleteAllUsersByDomain()

The `DeleteAllUsersByDomain()` function deletes all users associated with the given domain. It returns `false` if no domain is defined in the model.

For example:

[Go](#)

```
ok, err := e.DeleteAllUsersByDomain("domain1")
```

DeleteDomains()

DeleteDomains would delete all associated users and roles. It would delete all domains if parameter is not provided.

For example:

[Go](#)

```
ok, err := e.DeleteDomains("domain1", "domain2")
```

GetAllDomains()

GetAllDomains would get all domains.

For example:

[Go](#)

```
res, _ := e.GetAllDomains()
```

ⓘ NOTE

If you are handling a domain like `name::domain`, it may lead to unexpected behavior. In Casbin, `::` is a reserved keyword, just like `for`, `if` in a programming language, we should never put `::` in a domain.

GetAllRolesByDomain()

`GetAllRolesByDomain` would get all roles associated with the domain.

For example:

[Go](#)

```
res := e.GetAllRolesByDomain("domain1")
```

ⓘ NOTE

This method does not apply to domains that have an inheritance relationship, also known as implicit roles.

GetImplicitUsersForResourceByDomain()

`GetImplicitUsersForResourceByDomain` return implicit user based on resource and domain.

For example:

```
p, admin, domain1, data1, read
```

```
GetImplicitUsersForResourceByDomain("data1", "domain1") will return [[{"alice",  
"domain1", "data1", "read"}, {"alice", "domain1", "data1", "write"}]],  
nil
```

[Go](#)

```
ImplicitUsers, err :=  
e.GetImplicitUsersForResourceByDomain("data1", "domain1")
```

 NOTE

Only users will be returned, roles (2nd arg in "g") will be excluded.

RBAC with Conditions API

A more user-friendly API for [RBAC with conditions](#).

Reference

AddNamedLinkConditionFunc

`AddNamedLinkConditionFunc` Add condition function fn for Link `userName->roleName`, when fn returns true, Link is valid, otherwise invalid

[Go](#)

```
e.AddNamedLinkConditionFunc("g", "userName", "roleName",  
YourLinkConditionFunc)
```

AddNamedDomainLinkConditionFunc

`AddNamedDomainLinkConditionFunc` Add condition function fn for Link `userName-> {roleName, domain}`, when fn returns true, Link is valid, otherwise invalid

[Go](#)

```
e.AddNamedDomainLinkConditionFunc("g", "userName", "roleName",  
"domainName", YourLinkConditionFunc)
```

SetNamedLinkConditionFuncParams

`SetNamedLinkConditionFuncParams` Sets the parameters of the condition function fn for Link `userName->roleName`

[Go](#)

```
e.SetNamedLinkConditionFuncParams("g", "userName", "roleName",
"YourConditionFuncParam")
e.SetNamedLinkConditionFuncParams("g", "userName2",
"roleName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```

SetNamedDomainLinkConditionFuncParams

`SetNamedDomainLinkConditionFuncParams` Sets the parameters of the condition function fn for Link `userName->{roleName, domain}`

[Go](#)

```
e.SetNamedDomainLinkConditionFuncParams("g", "userName",
"roleName", "domainName", "YourConditionFuncParam")
e.SetNamedDomainLinkConditionFuncParams("g", "userName2",
"roleName2", "domainName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```

RoleManager API

RoleManager

The RoleManager provides an interface for defining operations to manage roles. The addition of a matching function to the RoleManager allows the use of wildcards in role names and domains.

AddNamedMatchingFunc()

The `AddNamedMatchingFunc` function adds a `MatchingFunc` by Ptype to the RoleManager. The `MatchingFunc` will be used when performing role matching.

[Go](#) Node.js

```
e.AddNamedMatchingFunc("g", "", util.KeyMatch)
_, _ = e.AddGroupingPolicies([][]string{{"*", "admin",
"domain1"}})
_, _ = e.GetRoleManager().HasLink("bob", "admin",
"domain1") // -> true, nil

await e.addNamedMatchingFunc('g', Util.keyMatchFunc);
await e.addGroupingPolicies([["*", 'admin', 'domain1']]);
await e.getRoleManager().hasLink('bob', 'admin', 'domain1');
```

For example:

[Go](#) Node.js

```
e, _ := casbin.NewEnforcer("path/to/model", "path/to/
policy")
e.AddNamedMatchingFunc("g", "", util.MatchKey)

const e = await newEnforcer('path/to/model', 'path/to/
policy');
await e.addNamedMatchingFunc('g', Util.keyMatchFunc);
```

AddNamedDomainMatchingFunc()

The `AddNamedDomainMatchingFunc` function adds a `MatchingFunc` by Ptype to the RoleManager. The `DomainMatchingFunc` is similar to the `MatchingFunc` listed above.

For example:

[Go](#) [Node.js](#)

```
e, _ := casbin.NewEnforcer("path/to/model", "path/to/
policy")
e.AddNamedDomainMatchingFunc("g", "", util.MatchKey)

const e = await newEnforcer('path/to/model', 'path/to/
policy');
await e.addNamedDomainMatchingFunc('g', Util.keyMatchFunc);
```

GetRoleManager()

The `GetRoleManager` function gets the current role manager for `g`.

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm := e.GetRoleManager()  
  
const rm = await e.getRoleManager();  
  
rm = e.get_role_manager()
```

GetNamedRoleManager()

The `GetNamedRoleManager` function gets the role manager by named Ptype.

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm := e.GetNamedRoleManager("g2")  
  
const rm = await e.getNamedRoleManager("g2");  
  
rm = e.get_named_role_manager("g2")
```

SetRoleManager()

The `SetRoleManager` function sets the current role manager for `g`.

For example:

[Go](#) [Node.js](#) [Python](#)

```
e.SetRoleManager(rm)  
  
e.setRoleManager(rm);  
  
rm = e.set_role_manager(rm)
```

SetNamedRoleManager()

The `SetNamedRoleManager` function sets the role manager by named Ptype.

For example:

[Go](#) [Python](#)

```
rm := e.SetNamedRoleManager("g2", rm)  
  
rm = e.set_role_manager("g2", rm)
```

Clear()

The `Clear` function clears all stored data and resets the role manager to its initial state.

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.Clear()  
  
await rm.clear();  
  
rm.clear()
```

AddLink()

AddLink adds the inheritance link between two roles. role: name1 and role: name2. Domain is a prefix to the roles (can be used for other purposes).

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.AddLink("u1", "g1", "domain1")  
  
await rm.addLink('u1', 'g1', 'domain1');  
  
rm.add_link("u1", "g1", "domain1")
```

DeleteLink()

DeleteLink deletes the inheritance link between two roles. role: name1 and role: name2. Domain is a prefix to the roles (can be used for other purposes).

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.DeleteLink("u1", "g1", "domain1")  
  
await rm.deleteLink('u1', 'g1', 'domain1');  
  
rm.delete_link("u1", "g1", "domain1")
```

HasLink()

HasLink determines whether a link exists between two roles. role: name1 inherits role: name2. Domain is a prefix to the roles (can be used for other purposes).

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.HasLink("u1", "g1", "domain1")  
  
await rm.hasLink('u1', 'g1', 'domain1');  
  
rm.has_link("u1", "g1", "domain1")
```

GetRoles()

GetRoles gets the roles that a user inherits. Domain is a prefix to the roles (can be

used for other purposes).

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.GetRoles("u1", "domain1")  
  
await rm.getRoles('u1', 'domain1');  
  
rm.get_roles("u1", "domain")
```

GetUsers()

GetUsers gets the users that inherits a role. Domain is a prefix to the users (can be used for other purposes).

For example:

[Go](#) [Node.js](#) [Python](#)

```
rm.GetUsers("g1")  
  
await rm.getUsers('g1');  
  
rm.get_users("g1")
```

PrintRoles()

PrintRoles prints all the roles to log.

For example:

Go Node.js Python

```
rm.PrintRoles()  
  
await rm.printRoles();  
  
rm.print_roles()
```

SetLogger()

SetLogger sets role manager's logger.

For example:

Go

```
logger := log.DefaultLogger{}  
logger.EnableLog(true)  
rm.SetLogger(&logger)  
_ = rm.PrintRoles()
```

GetDomains()

GetDomains gets domains that a user has

For example:

Go

```
result, err := rm.GetDomains(name)
```

Advanced usage

Multi-threading

Utilizing Casbin in a multi-threading environment

Benchmarks

Overhead of Policy Enforcement in Casbin

Performance Optimization

Casbin performance optimization

Authorization of Kubernetes

Kubernetes (k8s) RBAC & ABAC authorization middleware based on Casbin



Admission Webhook for K8s

Kubernetes (K8s) RBAC & ABAC Authorization Middleware based on Casbin



Authorization of Service Mesh through Envoy

Authorization of Service Mesh through Envoy

Multi-threading

When using Casbin in a multi-threading environment, you can employ the synchronized wrapper of the Casbin enforcer: https://github.com/casbin/casbin/blob/master/enforcer_synced.go (GoLang) and https://github.com/casbin/casbin-cpp/blob/master/casbin/enforcer_synced.cpp (C++).

Furthermore, it also provides support for the "AutoLoad" feature, allowing the Casbin enforcer to automatically load the latest policy rules from the database if any changes occur. To initiate the automatic loading of policies periodically, call the "StartAutoLoadPolicy()" function. Likewise, to stop this automatic loading, call the "StopAutoLoadPolicy()" function.

Benchmarks

[Go](#) [Python](#) [C++](#) [Lua \(JIT\)](#)

The overhead of policy enforcement has been benchmarked in [model_b_test.go](#). The testbed configuration is as follows:

```
Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core(s), 8 Logical Processor(s)
```

Following are the benchmarking results obtained by running `go test -bench=. -benchmem` (op = an `Enforce()` call, ms = millisecond, KB = kilobytes):

Test case	Rule size	Time overhead (ms/op)	Memory overhead (KB)
ACL	2 rules (2 users)	0.015493	5.649
RBAC	5 rules (2 users, 1 role)	0.021738	7.522
RBAC (small)	1100 rules (1000 users, 100 roles)	0.164309	80.620
RBAC (medium)	11000 rules (10000 users, 1000 roles)	2.258262	765.152
RBAC (large)	110000 rules (100000 users, 10000 roles)	23.916776	7,606
RBAC with resource roles	6 rules (2 users, 2 roles)	0.021146	7.906
RBAC with domains/tenants	6 rules (2 users, 1 role, 2 domains)	0.032696	10.755
ABAC	0 rule (0 user)	0.007510	2.328
RESTful	5 rules (3 users)	0.045398	91.774
Deny-override	6 rules (2 users, 1 role)	0.023281	8.370
Priority	9 rules (2 users, 2 roles)	0.016389	5.313

The overhead of policy enforcement in [Pycasbin](#) has been benchmarked in the [tests/benchmarks](#) directory. The testbed configuration is as follows:

```
Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz (Runned by Github actions)
platform linux -- Python 3.11.4, pytest-7.0.1, pluggy-1.2.0
```

Here are the benchmarking results obtained from executing `casbin_benchmark` (op = an `enforce()` call, ms = millisecond):

Test case	Rule size	Time overhead (ms/op)
ACL	2 rules (2 users)	0.067691
RBAC	5 rules (2 users, 1 role)	0.080045

Test case	Rule size	Time overhead (ms/op)
RBAC (small)	1100 rules (1000 users, 100 roles)	0.853590
RBAC (medium)	11000 rules (10000 users, 1000 roles)	6.986668
RBAC (large)	110000 rules (100000 users, 10000 roles)	77.922851
RBAC with resource roles	6 rules (2 users, 2 roles)	0.106090
RBAC with domains/tenants	6 rules (2 users, 1 role, 2 domains)	0.103628
ABAC	0 rule (0 user)	0.053213
RESTful	5 rules (3 users)	NA
Deny-override	6 rules (2 users, 1 role)	NA
Priority	9 rules (2 users, 2 roles)	0.084684

The overhead of policy enforcement in Casbin CPP has been benchmarked in the [tests/benchmarks](#) directory using Google's [benchmarking tool](#). The testbed configuration is as follows:

```
Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 4 cores, 4 threads
```

Here are the benchmarking results obtained from executing the `casbin_benchmark` target built in the `Release` configuration (`op = an enforce() call, ms = millisecond`):

Test case	Rule size	Time overhead (ms/op)
ACL	2 rules (2 users)	0.0195
RBAC	5 rules (2 users, 1 role)	0.0288
RBAC (small)	1100 rules (1000 users, 100 roles)	0.300
RBAC (medium)	11000 rules (10000 users, 1000 roles)	2.113
RBAC (large)	110000 rules (100000 users, 10000 roles)	21.450
RBAC with resource roles	6 rules (2 users, 2 roles)	0.03
RBAC with domains/tenants	6 rules (2 users, 1 role, 2 domains)	0.041
ABAC	0 rule (0 user)	NA
RESTful	5 rules (3 users)	NA
Deny-override	6 rules (2 users, 1 role)	0.0246

Test case	Rule size	Time overhead (ms/op)
Priority	9 rules (2 users, 2 roles)	0.035

The overhead of policy enforcement in [Lua Casbin](#) has been benchmarked in [bench.lua](#). The testbed configuration is as follows:

AMD Ryzen(TM) 5 4600H CPU @ 3.0GHz, 6 Cores, 12 Threads

Here are the benchmarking results obtained by running `luajit bench.lua` (op = an `enforce()` call, ms = millisecond):

Test case	Rule size	Time overhead (ms/op)
ACL	2 rules (2 users)	0.0533
RBAC	5 rules (2 users, 1 role)	0.0972
RBAC (small)	1100 rules (1000 users, 100 roles)	0.8598
RBAC (medium)	11000 rules (10000 users, 1000 roles)	8.6848
RBAC (large)	110000 rules (100000 users, 10000 roles)	90.3217
RBAC with resource roles	6 rules (2 users, 2 roles)	0.1124
RBAC with domains/tenants	6 rules (2 users, 1 role, 2 domains)	0.1978
ABAC	0 rule (0 user)	0.0305
RESTful	5 rules (3 users)	0.1085
Deny-override	6 rules (2 users, 1 role)	0.1934
Priority	9 rules (2 users, 2 roles)	0.1437

Benchmark monitoring

In the embedded web page below, you can see the performance changes of Casbin for each commit.

You can also directly browse it at: <https://v1.casbin.org/casbin/benchmark-monitoring>

Last Update:
Repository:

[Download data as JSON](#)

Powered by [github-action-benchmark](#)

Performance Optimization

When applied in a production environment with millions of users or permissions, you may encounter a performance downgrade in Casbin enforcement. There are usually two causes:

High Volume Traffic

The number of incoming requests per second is too large, for example, 10,000 requests/s for a single Casbin instance. In such cases, a single Casbin instance is usually not enough to handle all the requests. There are two possible solutions:

1. Use multi-threading to enable multiple Casbin instances, so you can fully utilize all the cores in the machine. For more details, see: [Multi-threading](#).
2. Deploy Casbin instances to a cluster (multiple machines) and use Watcher to ensure all Casbin instances are consistent. For more details, see: [Watchers](#).

NOTE

You can use both of the above methods at the same time, for example, deploy Casbin to a 10-machine cluster where each machine has 5 threads simultaneously serving Casbin enforcement requests.

High Number of Policy Rules

In a cloud or multi-tenant environment, millions of policy rules may be required. Each enforcement call or even loading the policy rules at the initial time can be very slow. Such cases can usually be mitigated in several ways:

1. Check if your Casbin model or policy is well-designed. A well-written model and policy abstracts out the duplicated logic for each user/tenant and reduces

the number of rules to a very small level (< 100). For example, you can share some default rules across all tenants and allow users to customize their rules later. Customized rules can override the default rules. If you have any further questions, please open a GitHub issue on the Casbin repository.

2. Do sharding to let a Casbin enforcer only load a small set of policy rules. For example, enforcer_0 can serve tenant_0 to tenant_99, while enforcer_1 can serve tenant_100 to tenant_199. To load only a subset of all policy rules, see: [Policy Subset Loading](#).
3. Grant permissions to RBAC roles instead of users directly. Casbin's RBAC is implemented by a role inheritance tree (as a cache). So, given a user like Alice, Casbin only takes O(1) time to query the RBAC tree for the role-user relationship and perform enforcement. If your g rules don't change often, then the RBAC tree won't need to be constantly updated. See the details of this discussion here: <https://github.com/casbin/casbin/issues/681#issuecomment-763801583>

 NOTE

You can try all of the above methods at the same time.

Authorization of Kubernetes

[K8s-authz](#) is a Kubernetes (k8s) authorization middleware based on Casbin that utilizes RBAC (Role-Based Access Control) and ABAC (Attribute-Based Access Control) for policy enforcement. This middleware integrates with the K8s validation admission webhook to validate the policies defined by Casbin for each request made to K8s resources. Custom admission controllers are registered with Kubernetes using the `ValidatingAdmissionWebhook` to perform validations on request objects forwarded by the API server and provide a response indicating whether the request should be allowed or rejected.

To determine when to send incoming requests to the admission controller, a validation webhook has been implemented. This webhook proxies requests for any type of K8s resource or sub-resource and performs policy verification. Users are only allowed to perform operations on these resources if they are authorized by the Casbin enforcer. The [enforcer](#) checks the roles of the user as defined in the policies. The K8s cluster is the deployment target for this middleware.

Requirements

Before proceeding, ensure that you have the following:

- A running Kubernetes cluster. You can set up a local cluster using Docker or set up a complete Kubernetes ecosystem on your server. For detailed instructions, refer to this [guide](#) for setting up a local Kubernetes cluster on Windows or this [guide](#) for setting up a cluster on Linux.
- Kubectl CLI. Instructions for installing Kubectl on Windows can be found [here](#),

and for Linux [here](#).

- OpenSSL

Usage

Follow these steps to use K8s-authz:

1. Generate certificates and keys for each user using OpenSSL. Run the script below:

```
./gen_cert.sh
```

2. Build the Docker image from the [Dockerfile](#) manually by running the following command. Remember to change the build version in the command and in the deployment [file](#) accordingly.

```
docker build -t casbin/k8s_authz:0.1 .
```

3. Define the Casbin policies in the [model.conf](#) and [policy.csv](#) files. For more information on how these policies work, refer to the [documentation](#).
4. Before deploying, you can modify the ports in the [main.go](#) file, as well as in the validation webhook configuration [file](#), based on your specific requirements.
5. Deploy the validation controller and the webhook on the Kubernetes cluster by running the following command:

```
kubectl apply -f deployment.yaml
```

6. For a production server, it is recommended to create a Kubernetes `secret` to secure the certificates:

```
kubectl create secret generic casbin -n default \
--from-file=key.pem=certs/casbin-key.pem \
--from-file=cert.pem=certs/casbin-crt.pem
```

7. After completing the above steps, you need to update the certificate directory in `main.go` and the `manifests` with the directory of the created `secret`.

Now, the server should be up and running, ready to validate requests made to K8s resources and enforce policies accordingly.

Admission Webhook for K8s

1. Overview & Documents for Casbin K8s-Gatekeeper

Casbin K8s-GateKeeper is a Kubernetes admission webhook that integrates Casbin as the Access Control tool. By using Casbin K8s-GateKeeper, you can establish flexible rules to authorize or intercept any operation on K8s resources, WITHOUT writing any piece of code, but only several lines of declarative configurations of Casbin models and policies, which are part of the Casbin ACL (Access Control List) language.

Casbin K8s-GateKeeper is developed and maintained by the Casbin community. The repository of this project is available here: <https://github.com/casbin/k8s-gatekeeper>

0.1 A Simple Example

For example, you don't need to write any code, but use the following lines of configuration to achieve this function: "Forbid images with some specified tags to be used in any deployments":

Model:

```
[request_definition]
r = obj

[policy_definition]
p = obj,eft

[policy_effect]
e = !some(where (p.eft == deny))

[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
== "deployments" && \
contain(split(accessWithWildcard(${OBJECT}.Spec.Template.Spec.Containers , "*",
"Image"),":",1) , p.obj)
```

And Policy:

```
p, "1.14.1", deny
```

These are in ordinary Casbin ACL language. Suppose you have already read chapters about them, it will be very easy to understand.

Casbin K8s-Gatekeeper has the following advantages:

- Easy to use. Writing several lines of ACL is far better than writing lots of code.

- It allows hot updates of configurations. You don't need to shut down the whole plugin to modify configurations.
- It is flexible. Arbitrary rules can be made on any K8s resource, which can be explored with `kubectl gatekeeper`.
- It simplifies the implementation of K8s admission webhook, which is very complicated. You don't need to know what K8s admission webhook is or how to write code for it. All you need to do is to know the resource on which you want to put constraints and then write Casbin ACL. Everyone knows that K8s is complex, but by using Casbin K8s-Gatekeeper, your time can be saved.
- It is maintained by the Casbin community. Feel free to contact us if anything about this plugin confuses you or if you encounter any problems when trying this.

1.1 How Casbin K8s-Gatekeeper Works?

K8s-Gatekeeper is an admission webhook for K8s that uses [Casbin](#) to apply arbitrary user-defined access control rules to help prevent any operation on K8s that the administrator doesn't want.

Casbin is a powerful and efficient open-source access control library. It provides support for enforcing authorization based on various access control models. For more details about Casbin, see [Overview](#).

Admission webhooks in K8s are HTTP callbacks that receive 'admission requests' and do something with them. In particular, K8s-Gatekeeper is a special type of admission webhook: 'ValidatingAdmissionWebhook', which can decide whether to accept or reject this admission request or not. As for admission requests, they are HTTP requests describing an operation on specified resources of K8s (for example, creating/deleting a deployment). For more about admission webhooks, see [K8s official documentation](#).

1.2 An Example Illustrating How It Works

For example, when somebody wants to create a deployment containing a pod running nginx (using kubectl or K8s clients), K8s will generate an admission request, which (if translated into YAML format) can be something like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

This request will go through the process of all the middleware shown in the picture, including our K8s-Gatekeeper. K8s-Gatekeeper can detect all the Casbin enforcers stored in K8s's etcd, which is created and maintained by the user (via `kubectl` or the Go client we provide). Each enforcer contains a Casbin model and a Casbin policy. The admission request will be processed by every enforcer, one by one, and only by passing all enforcers can a request be accepted by this K8s-Gatekeeper.

(If you do not understand what a Casbin enforcer, model, or policy is, see this document: [Get Started](#)).

For example, for some reason, the administrator wants to forbid the appearance of the image 'nginx:1.14.1' while allowing 'nginx:1.3.1'. An enforcer containing the following rule and policy can be created (We will explain how to create an enforcer, what these models and policies are, and how to write them in the following chapters).

Model:

```
[request_definition]
r = obj

[policy_definition]
p = obj,eft

[policy_effect]
e = !some(where (p.eft == deny))

[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
== "deployments" && \
access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image") == p.obj
```

Policy:

```
p, "nginx:1.13.1",allow
p, "nginx:1.14.1",deny
```

By creating an enforcer containing the model and policy above, the previous admission request will be rejected by this enforcer, which means K8s won't create this deployment.

2 Install K8s-gatekeeper

There are three methods available for installing K8s-gatekeeper: External webhook, Internal webhook, and Helm.

 NOTE

Note: These methods are only meant for users to try out K8s-gatekeeper and are not secure. If you wish to use it in a productive environment, please ensure that you read [Chapter 5. Advanced settings](#) and make

any necessary modifications before installation.

2.1 Internal webhook

2.1.1 Step 1: Build the image

For the internal webhook method, the webhook itself will be implemented as a service within Kubernetes. To create the necessary service and deployment, you need to build an image of K8s-gatekeeper. You can build your own image by running the following command:

```
docker build --target webhook -t k8s-gatekeeper .
```

This command will create a local image called 'k8s-gatekeeper:latest'.

NOTE

Note: If you are using minikube, please execute `eval $(minikube -p minikube docker-env)` before running 'docker build'.

2.1.2 Step 2: Set up services and deployments for K8s-gatekeeper

Run the following commands:

```
kubectl apply -f config/rbac.yaml  
kubectl apply -f config/webhook_deployment.yaml  
kubectl apply -f config/webhook_internal.yaml
```

This will start running K8s-gatekeeper, and you can confirm this by running `kubectl get pods`.

2.1.3 Step 3: Install CRD Resources for K8s-gatekeeper

Run the following commands:

```
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml  
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
```

2.2 External webhook

For the external webhook method, K8s-gatekeeper will be running outside of Kubernetes, and Kubernetes will access K8s-gatekeeper as it would access a regular website. Kubernetes has a mandatory requirement that the admission webhook must be HTTPS. For the purpose of trying out K8s-gatekeeper, we have provided a set of certificates and a private key (although this is not secure). If you prefer to use your own certificate, please refer to

[Chapter 5. Advanced settings](#) for instructions on adjusting the certificate and private key.

The certificate we provide is issued for 'webhook.domain.local'. So, modify the host (e.g., /etc/hosts) and point 'webhook.domain.local' to the IP address on which K8s-gatekeeper is running.

Then execute the following command:

```
go mod tidy
go mod vendor
go run cmd/webhook/main.go
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
kubectl apply -f config/webhook_external.yaml
```

2.3 Install K8s-gatekeeper via Helm

2.3.1 Step 1: Build the image

Please refer to [Chapter 2.1.1](#).

2.3.2 Helm installation

Run the command `helm install k8sgatekeeper ./k8sgatekeeper`.

3. Try K8s-gatekeeper

3.1 Create Casbin Model and Policy

You have two methods to create a model and policy: via kubectl or via the go-client we provide.

3.1.1 Create/Update Casbin Model and Policy via kubectl

In K8s-gatekeeper, the Casbin model is stored in a CRD resource called 'CasbinModel'. Its definition is located in `config/auth.casbin.org_casbinmodels.yaml`.

There are examples in `example/allowed_repo/model.yaml`. Pay attention to the following fields:

- `metadata.name`: the name of the model. This name MUST be the same as the name of the CasbinPolicy object related to this model, so that K8s-gatekeeper can pair them and create an enforcer.
- `spec.enable`: if this field is set to "false", this model (as well as the CasbinPolicy object related to this model) will be ignored.
- `spec.modelText`: a string that contains the model text of a Casbin model.

The Casbin Policy is stored in another CRD resource called 'CasbinPolicy', whose definition can be found in `config/auth.casbin.org_casbinpolicies.yaml`.

There are examples in `example/allowed_repo/policy.yaml`. Pay attention to the following fields:

- `metadata.name`: the name of the policy. This name MUST be the same as the name of the CasbinModel object related to this policy, so that K8s-gatekeeper can pair them and create an enforcer.
- `spec.policyItem`: a string that contains the policy text of a Casbin model.

After creating your own CasbinModel and CasbinPolicy files, use the following command to apply them:

```
kubectl apply -f <filename>
```

Once a pair of CasbinModel and CasbinPolicy is created, K8s-gatekeeper will be able to detect it within 5 seconds.

3.1.2 Create/Update Casbin Model and Policy via the go-client we provide

We understand that there may be situations where it is not convenient to use the shell to execute commands directly on a node of the K8s cluster, such as when you are building an automatic cloud platform for your corporation. Therefore, we have developed a go-client to create and maintain CasbinModel and CasbinPolicy.

The go-client library is located in `pkg/client`.

In `client.go`, we provide a function to create a client.

```
func NewK8sGateKeeperClient(externalClient bool) (*K8sGateKeeperClient, error)
```

The `externalClient` parameter determines whether K8s-gatekeeper is running inside the K8s cluster or not.

In `model.go`, we provide various functions to create, delete, and modify CasbinModel. You can find out how to use these interfaces in `model_test.go`.

In `policy.go`, we provide various functions to create, delete, and modify CasbiPolicy. You can find out how to use these interfaces in `policy_test.go`.

3.1.2 Try Whether K8s-gatekeeper Works

Suppose you have already created the exact model and policy in `example/allowed_repo`. Now, try the following command:

```
kubectl apply -f example/allowed_repo/testcase/reject_1.yaml
```

You should find that K8s will reject this request and mention that the webhook was the reason why this request is rejected. However, when you try to apply `example/allowed_repo/testcase/approve_2.yaml`, it will be accepted.

4. How to Write Model and Policy with K8s-gatekeeper

First of all, make sure you are familiar with the basic grammar of Casbin Models and Policies. If you are not, please read the [Get Started](#) section first. In this chapter, we assume that you already understand what Casbin Models and Policies are.

4.1 Request Definition of Model

When K8s-gatekeeper is authorizing a request, the input is always an object: the Go object of the Admission Request. This means that the enforcer will always be used like this:

```
ok, err := enforcer.Enforce(admission)
```

where `admission` is an `AdmissionReview` object defined by K8s's official go api "`"k8s.io/api/admission/v1"`". You can find the definition of this struct in this repository: <https://github.com/kubernetes/api/blob/master/admission/v1/types.go>. For more information, you can also refer to <https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-request-and-response>.

Therefore, for any model used by K8s-gatekeeper, the definition of the `request_definition` should always be like this:

```
[request_definition]
r = obj
```

The name 'obj' is not mandatory, as long as the name is consistent with the name used in the `[matchers]` part.

4.2 Matchers of Model

You are supposed to use the ABAC feature of Casbin to write your rules. However, the expression evaluator integrated in Casbin does not support indexing in maps or arrays(slices), nor the expansion of arrays. Therefore, K8s-gatekeeper provides various 'Casbin functions' as extensions to implement these features. If you still find that your demand cannot be fulfilled by these extensions, feel free to start an issue, or create a pull request.

If you are not familiar with Casbin functions, you can refer to [Function](#) for more information.

Here are the extension functions:

4.2.1 Extension functions

4.2.1.1 access

Access is used to solve the problem that Casbin does not support indexing in maps or arrays. The example

`example/allowed_repo/model.yaml` demonstrates the usage of this function:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image") == p.obj
```

In this matcher, `access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image")` is equivalent to `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Image`, where `r.obj.Request.Object.Object.Spec.Template.Spec.Containers` is a slice.

Access can also call simple functions that have no parameters and return a single value. The example `example/container_resource_limit/model.yaml` demonstrates this:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
parseFloat(access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0,
"Resources","Limits","cpu","Value")) >= parseFloat(p.cpu) && \
parseFloat(access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0,
"Resources","Limits","memory","Value")) >= parseFloat(p.memory)
```

In this matcher, `access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Resources","Limits","cpu","Value")` is equivalent to `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits["cpu"].Value()`, where `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits` is a map, and `Value()` is a simple function that has no parameters and returns a single value.

4.2.1.2 accessWithWildcard

Sometimes, you may have a demand like this: all elements in an array must have a prefix "aaa". However, Casbin does not support `for` loops. With `accessWithWildcard` and the "map/slice expansion" feature, you can easily implement such a demand.

For example, suppose `a.b.c` is an array `[aaa, bbb, ccc, ddd, eee]`, then the result of `accessWithWildcard(a, "b", "c", "*", "*")` will be a slice `[aaa, bbb, ccc, ddd, eee]`. By using the wildcard `*`, the slice is expanded.

Similarly, the wildcard can be used more than once. For example, the result of `accessWithWildcard(a, "b", "c", "*", "*", "*")` will be `[a.b.c[0][0], a.b.c[0][1], ..., a.b.c[1][0], a.b.c[1][1], ...]`.

4.2.1.3 Functions Supporting Variable-length Arguments

In the expression evaluator of Casbin, when a parameter is an array, it will be automatically expanded as a

variable-length argument. Utilizing this feature to support array/slice/map expansion, we have also integrated several functions that accept an array/slice as a parameter:

- `contain()`: accepts multiple parameters and returns whether any parameter (except the last parameter) equals the last parameter.
- `split(a,b,c...,sep,index)`: returns a slice that contains `[splits(a,sep)[index], splits(b,sep)[index], splits(a,sep)[index], ...]`.
- `len()`: returns the length of the variable-length argument.
- `matchRegex(a,b,c...,regex)`: returns whether all of the given parameters (`a`, `b`, `c`, ...) match the given regex.

Here is an example in `example/disallowed_tag/model.yaml`:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1) , p.obj)
```

Assuming that `accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image")` returns `["a:b", "c:d", "e:f", "g:h"]`, because `splits` supports variable-length arguments and performs the `splits` operation on each element, the element at index 1 will be selected and returned. Therefore, `split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1)` returns `["b", "d", "f", "h"]`. And `contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1) , p.obj)` returns whether `p.obj` is contained in `["b", "d", "f", "h"]`.

4.2.1.2 Type Conversion Functions

- `ParseFloat()`: Parses an integer to a float (this is necessary because any number used in comparison must be converted into a float).
- `ToString()`: Converts an object to a string. This object must have a basic type of string (for example, an object of type `XXX` when there is a statement `type XXX string`).
- `IsNil()`: Returns whether the parameter is nil.

5. Advanced Settings

5.1 About Certificates

In Kubernetes (k8s), it is mandatory that a webhook should use HTTPS. There are two approaches to achieve this:

- Use self-signed certificates (examples in this repository use this method)
- Use a normal certificate

5.1.1 Self-signed certificates

Using a self-signed certificate means that the Certificate Authority (CA) issuing the certificate is not one of the well-known CAs. Therefore, you must let k8s know about this CA.

Currently, the example in this repository uses a self-made CA, whose private key and certificate are stored in `config/certificate/ca.crt` and `config/certificate/ca.key` respectively. The certificate for the webhook is `config/certificate/server.crt`, which is issued by the self-made CA. The domains of this certificate are "webhook.domain.local" (for external webhook) and "casbin-webhook-svc.default.svc" (for internal webhook).

Information about the CA is passed to k8s via webhook configuration files. Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", which contains a base64 encoded string of the CA's certificate.

In case you need to change the certificate/domain (for example, if you want to put this webhook into another namespace of k8s while using an internal webhook, or if you want to change the domain while using an external webhook), the following procedures should be followed:

1. Generate a new CA:

- Generate the private key for the fake CA:

```
openssl genrsa -des3 -out ca.key 2048
```

- Remove the password protection of the private key:

```
openssl rsa -in ca.key -out ca.key
```

2. Generate a private key for the webhook server:

```
openssl genrsa -des3 -out server.key 2048  
openssl rsa -in server.key -out server.key
```

3. Use the self-generated CA to sign the certificate for the webhook:

- Copy your system's openssl config file for temporary use. You can find out the location of the config file by running `openssl version -a`, usually called `openssl.cnf`.
- In the config file:
 - Find the `[req]` paragraph and add the following line: `req_extensions = v3_req`
 - Find the `[v3_req]` paragraph and add the following line: `subjectAltName = @alt_names`

- Append the following lines to the file:

```
[alt_names]
DNS.2=<The domain you want>
```

Note: Replace 'casbin-webhook-svc.default.svc' with the real service name of your own service if you decide to modify the service name.

- Use the modified config file to generate a certificate request file:

```
openssl req -new -nodes -keyout server.key -out server.csr -config openssl.cnf
```

- Use the self-made CA to respond to the request and sign the certificate:

```
openssl x509 -req -days 3650 -in server.csr -out server.crt -CA ca.crt -CAkey ca.key -CAcreateserial -extensions v3_req -extensions SAN -extfile openssl.cnf
```

4. Replace the 'CABundle' field: Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", which contains a base64 encoded string of the certificate of the CA. Update this field with the new certificate.
5. If you are using helm, similar changes need to be applied to the helm charts.

5.1.2 Legal certificates

If you use legal certificates, you do not need to go through all these procedures. Remove the "CABundle" field in `config/webhook_external.yaml` and `config/webhook_internal.yaml`, and change the domain in these files to the domain you own.

Authorization of Service Mesh through Envoy

[Envoy-authz](#) is a middleware for Envoy that performs external RBAC & ABAC authorization through casbin. This middleware uses [Envoy's external authorization API](#) via a gRPC server. This proxy can be deployed on any type of Envoy-based service mesh, such as Istio.

Requirements

- Envoy 1.17+
- Istio or any other type of service mesh
- grpc dependencies

Dependencies are managed using `go.mod`.

Working of the Middleware

- A client makes an HTTP request.
- The Envoy proxy sends the request to the gRPC server.
- The gRPC server authorizes the request based on casbin policies.
- If authorized, the request is forwarded; otherwise, it is denied.

The gRPC server is based on protocol buffer from [external_auth.proto](#) in Envoy.

```
// A generic interface for performing authorization checks on
```

From the above proto file, we need to use the `Check()` service in the authorization server.

Usage

- Define the Casbin policies in the config files following this [guide](#).

You can verify/test your policies using the online [casbin-editor](#).

- Start the authentication server by running:

```
go build .
./authz
```

- Load the Envoy configuration:

```
envoy -c authz.yaml -l info
```

Once Envoy starts, it will intercept requests for the authorization process.

Integrating with Istio

To make this middleware work, you need to send custom headers containing usernames in the JWT token or headers. You can refer to the official [Istio documentation](#) for more information on modifying `Request Headers`.

Management

Admin Portal

Admin portal for Casbin

Casbin Service

Using Casbin as a Service

Command-line Tools

Command-line Tools

Log & Error Handling

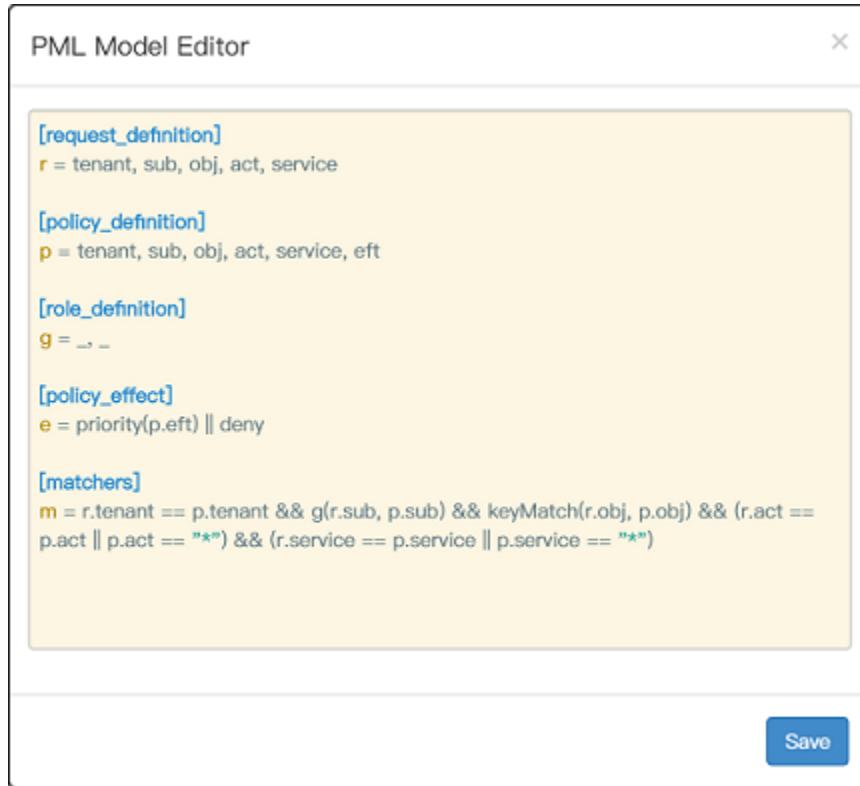
Logging and error handling in Casbin

Frontend Usage

Casbin.js is a Casbin addon that facilitates your access-control management in the frontend application

Admin Portal

We provide a web-based portal called [Casdoor](#) for model management and policy management:



Rule Type	Tenant	User	Resource Path	Action	Service	Auth Effect	Option
p	tenant1	admin1	/*	*	*	allow	
p	tenant1	user12	/*	*	nova	allow	
p	tenant1	user13	/*	*	glance	allow	
g	user11	admin1					

Save

There are also third-party admin portal projects that use Casbin as an authorization engine. You can get started building your own Casbin service based on these projects.

[Go](#) [Java](#) [Node.js](#) [Python](#) [PHP](#)

Project	Author	Frontend	Backend	Description
Casdoor	Casbin	React + Ant Design	Beego	Based on Beego + XORM + React
go-admin-team/go-admin	@go-admin-team	Vue + Element UI	Gin	go-admin Based on Gin + Casbin + GORM
gin-vue-flipped-		Vue +	Gin	Based on Gin +

Project	Author	Frontend	Backend	Description
admin	aurora	Element UI		GORM + Vue
gin-admin	@LyricTian	React + Ant Design	Gin	RBAC scaffolding based on Gin + GORM + Casbin + Ant Design React
go-admin	@hequan2017	None	Gin	Go RESTful API gateway based on Gin + GORM + JWT + RBAC (Casbin)
zeus-admin	bullteam	Vue + Element UI	Gin	Unified Permission management platform based on JWT + Casbin
IrisAdminApi	@snowlyg	Vue + Element UI	Iris	Backend API based on Iris + Casbin
Gfast	@tiger1103	Vue + Element UI	Go Frame	Admin portal based on GF (Go Frame)
echo-admin (Frontend, Backend)	@RealLiuSha	Vue 2.x + Element	Echo	Admin portal based on Echo + Gorm + Casbin + Uber-FX

Project		Author	Frontend	Backend	Description
		UI			
Spec-Center		@atul-wankhade	None	Mux	Golang RESTful platform based on Casbin + MongoDB
Project	Author	Frontend	Backend	Description	
spring-boot-web	@BazookaW	None	SpringBoot	Admin portal based on SpringBoot 2.0 + MyBatisPlus + Casbin	
Project		Author	Frontend	Backend	Description
node-mysql-rest-api		@JoemaNequinto	None	Express	A boilerplate application for building RESTful APIs Microservice in Node.js using Express, Sequelize, JWT and Casbin.
Casbin-Role-Mgt-Dashboard-RBAC		@alikhan866	React + Material UI	Express	Beginner friendly RBAC management with Enforcer integration to check

Project	Author	Frontend	Backend	Description
				enforcement result on the go
Project	Author	Frontend	Backend	Description
fastapi-best-architecture	@WuClan	Vue + Arco-design	FastAPI	Admin portal based on FastAPI, SQLAlchemy, JWT and RBAC
fastapi-mysql-generator	@CoderCharm	None	FastAPI	FastAPI + MySQL + JWT + Casbin
FastAPI-MySQL-Tortoise-Casbin	@xingxingzaixian	None	FastAPI	FastAPI + MySQL + Tortoise + Casbin
openstack-policy-editor	Casbin	Bootstrap	Django	The Web UI for Casbin
Project	Author	Frontend	Backend	Description
Tadmin	@leeqvip	AmazeUI	ThinkPHP	Non-intrusive backend framework based on

Project	Author	Frontend	Backend	Description
				ThinkPHP
video.tinywan.com	@Tinywanner	LayUI	ThinkPHP	RESTful API gateway based on ThinkPHP5 + ORM + JWT + RBAC (Casbin)
laravel-casbin-admin	@pl1998	Vue + Element UI	Laravel	RBAC permission management system based on vue-element-admin and Laravel
larke-admin (Frontend, Backend)	@deatil	Vue 2 + Element UI	Laravel 8	Admin portal based on Laravel 8, JWT and RBAC
hyperf-vuetify-admin	@TragicMale	Vue + Vuetify 2.x	Hyperf	Admin portal based on Hyperf, Vuetify and Casbin

Casbin Service

How to Use Casbin as a Service?

Name	Description
Casbin Server	The official "Casbin as a Service" solution based on gRPC . Both Management API and RBAC API are provided.
middleware-acl	RESTful access control middleware based on Casbin.
auth-server	Auth Server for proofreading services.

Command-line Tools

Casbin CLIs are command-line tools that provide a command-line interface for Casbin, enabling you to use all of Casbin APIs in the shell. This documentation covers the usage of Casbin CLI for various languages including Rust, Java, Go, Python, .NET, and NodeJs.

Installation

Go (casbin-go-cli)

1. Clone project from repository

```
git clone https://github.com/casbin/casbin-go-cli.git
```

2. Build project

```
cd casbin-go-cli  
go build -o casbin
```

Rust (casbin-rust-cli)

From crates.io

```
cargo install casbin-rust-cli
```

From source

1. Clone project from repository

```
git clone https://github.com/casbin-rs/casbin-rust-cli.git
```

2. Build project

```
cd casbin-rust-cli  
cargo build --release
```

Java (casbin-java-cli)

1. Clone project from repository

```
git clone https://github.com/jcasbin/casbin-java-cli.git
```

2. Build project, the jar package will be generated in the target directory

```
cd casbin-java-cli  
mvn clean install
```

Python (casbin-python-cli)

1. Clone project from repository

```
git clone https://github.com/casbin/casbin-python-cli.git
```

2. Install dependencies

```
cd casbin-python-cli  
pip install -r requirements.txt
```

3. Run CLI

```
python -m casbin_cli.client [command] [options] [args]
```

.NET (casbin-dotnet-cli)

1. Clone project from repository

```
git clone https://github.com/casbin-net/casbin-dotnet-cli.git
```

2. Build project

```
cd casbin-dotnet-cli  
dotnet build
```

Usage

Options

options	description	must	remark
<code>-m, --model</code>	The path of the model file or model text	y	Please wrap it with <code>'''</code> and separate each line with <code>\ </code>
<code>-p, --policy</code>	The path of the policy file or policy text	y	Please wrap it with <code>'''</code> and separate each line with <code>\ </code>
<code>-e, --enforce</code>	Check permissions	n	Please wrap it with <code>'''</code>
<code>-ex, --enforceEx</code>	Check permissions and get which policy it is	n	Please wrap it with <code>'''</code>
<code>-AF, --addFuntion</code>	Add custom funtion (casbin-java-cli only)	n	Please wrap it with <code>'''</code> and separate each line with <code>\ </code>
<code>-ap, --addPolicy</code>	Add a policy rule to the policy file (casbin-java-cli only)	n	Please wrap it with <code>'''</code>

options	description	must	remark
-rp, --removePolicy	Remove a policy rule from the policy file (casbin-java-cli only)	n	Please wrap it with <code>'''</code>

Get started

- Check whether Alice has read permission on data1

```
./casbin enforce -m "examples/rbac_model.conf" -p "examples/rbac_policy.csv" "alice" "data1" "read"
```

```
{"allow":true,"explain":null}
```

```
./casbin enforce -m "[request_definition]\nr = sub, obj,
act\n[policy_definition]\nnp = sub, obj,
act\n[role_definition]\ng = _, _\n[policy_effect]\nne =
some(where (p.eft == allow))\n[matchers]\nm = g(r.sub,
p.sub) && r.obj == p.obj && r.act == p.act" -p "p, alice,
data1, read\np, bob, data2, write\np, data2_admin, data2,
read\np, data2_admin, data2, write\nng, alice, data2_admin"
"alice" "data1" "read"
```

```
{"allow":true,"explain":null}
```

- Check whether Alice has write permission for data2. If so, display the effective policy.

```
./casbin enforceEx -m "examples/rbac_model.conf" -p  
"examples/rbac_policy.csv" "alice" "data2" "write"
```

```
{"allow":true,"explain":["data2_admin","data2","write"]}
```

- Add a policy to the policy file (casbin-java-cli only)

```
./casbin addPolicy -m "examples/rbac_model.conf" -p  
"examples/rbac_policy.csv" "alice" "data2" "write"
```

```
{"allow":true,"explain":null}
```

- Delete a policy from the policy file (casbin-java-cli only)

```
./casbin removePolicy -m "examples/rbac_model.conf" -p  
"examples/rbac_policy.csv" "alice" "data2" "write"
```

```
{"allow":true,"explain":null}
```

Log & Error Handling

Logging

Casbin uses the built-in `log` to print logs to the console by default, like:

```
2017/07/15 19:43:56 [Request: alice, data1, read ---> true]
```

Logging is not enabled by default. You can toggle it via `Enforcer.EnableLog()` or the last parameter of `NewEnforcer()`.

 NOTE

For Golang: We already support logging the model, enforce request, role, and policy in Golang. You can define your own log for logging Casbin.

 NOTE

For Python: PyCasbin leverages the default Python logging mechanism. PyCasbin makes a call to `logging.getLogger()` to set the logger. No special logging configuration is needed other than initializing the logger in the parent application. If no logging is initialized within the parent application, you will not see any log messages from PyCasbin. At the same time, when you enable logs in PyCasbin, you can specify the logging configuration through the parameter `logging_config`. If no configuration is specified, it will use the [default log configuration](#). For other PyCasbin extensions, you can refer to the [Django logging docs](#) if you are a Django user. For other Python users, you should refer to the [Python logging docs](#) to configure the logger.

Use different loggers for different enforcers

Every enforcer can have its own logger to log information, and it can be changed at runtime.

And you can use a proper logger via the last parameter of `NewEnforcer()`. If you are using this way to initialize your enforcer, you don't need to use the `enabled` parameter because the priority of the `enabled` field in the logger is higher.

```
// Set a default logger as enforcer e1's logger.  
// This operation can also be seen as changing the logger of e1  
at runtime.  
e1.SetLogger(&Log.DefaultLogger{})  
  
// Set another logger as enforcer e2's logger.  
e2.SetLogger(&YouOwnLogger)  
  
// Set your logger when initializing enforcer e3.  
e3, _ := casbin.NewEnforcer("examples/rbac_model.conf", a,  
logger)
```

Supported loggers

We provide some loggers to help you log information.

[Go](#) [PHP](#)

Logger	Author	Description
Default	Casbin	The default logger using golang log.

Logger	Author	Description
logger (built-in)		
Zap logger	Casbin	Using zap , provide json encoded log and you can customize more with your own zap-logger.
Logger	Author	Description
psr3-bridge logger	Casbin	Provides a PSR-3 compliant bridge.

How to write a logger

Your logger should implement the [Logger](#) interface.

Method	Type	Description
EnableLog()	mandatory	Control whether to print the message.
IsEnabled()	mandatory	Show the current logger's enabled status.
LogModel()	mandatory	Log info related to the model.
LogEnforce()	mandatory	Log info related to enforcing.
LogRole()	mandatory	Log info related to the role.
LogPolicy()	mandatory	Log info related to the policy.

You can pass your custom `logger` to `Enforcer.SetLogger()`.

Here is an example of how to customize a logger for Golang:

```
import (
    "fmt"
    "log"
    "strings"
)

// DefaultLogger is the implementation for a Logger using golang log.
type DefaultLogger struct {
    enabled bool
}

func (l *DefaultLogger) EnableLog(enable bool) {
    l.enabled = enable
}

func (l *DefaultLogger) IsEnabled() bool {
    return l.enabled
}

func (l *DefaultLogger) LogModel(model [[[string]]) {
    if !l.enabled {
        return
    }
    var str strings.Builder
    str.WriteString("Model: ")
    for _, v := range model {
        str.WriteString(fmt.Sprintf("%v\n", v))
    }

    log.Println(str.String())
}
```

Error handling

Errors or panics may occur when you use Casbin for reasons like:

1. Invalid syntax in the model file (.conf).
2. Invalid syntax in the policy file (.csv).
3. Custom errors from storage adapters, e.g., MySQL fails to connect.
4. Casbin's bug.

There are five main functions you may need to be aware of for errors or panics:

Function	Behavior on error
<code>NewEnforcer()</code>	Returns an error
<code>LoadModel()</code>	Returns an error
<code>LoadPolicy()</code>	Returns an error
<code>SavePolicy()</code>	Returns an error
<code>Enforce()</code>	Returns an error

 NOTE

`NewEnforcer()` calls `LoadModel()` and `LoadPolicy()` internally. So you don't have to call the latter two when using `NewEnforcer()`.

Enable and disable

The enforcer can be disabled via the `Enforcer.EnableEnforce()` function.

When it's disabled, `Enforcer.Enforce()` will always return `true`. Other operations like adding or removing policies are not affected. Here's an example:

```
e := casbin.NewEnforcer("examples/basic_model.conf", "examples/basic_policy.csv")

// Will return false.
// By default, the enforcer is enabled.
e.Enforce("non-authorized-user", "data1", "read")

// Disable the enforcer at runtime.
e.EnableEnforce(false)

// Will return true for any request.
e.Enforce("non-authorized-user", "data1", "read")

// Enable the enforcer again.
e.EnableEnforce(true)

// Will return false.
e.Enforce("non-authorized-user", "data1", "read")
```


Frontend Usage

[Casbin.js](#) is a Casbin addon that facilitates your access-control management in the frontend application.

Installation

```
npm install casbin.js  
npm install casbin
```

or

```
yarn add casbin.js
```

Frontend Middlewares

Middleware	Type	Author	Description
react-authz	React	Casbin	React wrapper for Casbin.js
rbac-react	React	@daobeng	Role Based Access Control in React using HOCs, CASL and Casbin.js
vue-authz	Vue	Casbin	Vue wrapper for Casbin.js

Middleware	Type	Author	Description
angular-authz	Angular	Casbin	Angular wrapper for Casbin.js

Quick Start

You can use the `manual` mode in your frontend application and set the permissions whenever you wish.

```
const casbinjs = require("casbin.js");
// Set the user's permission:
// He/She can read `data1` and `data2` objects and can write
// `data1` object
const permission = {
  "read": ["data1", "data2"],
  "write": ["data1"]
}

// Run casbin.js in manual mode, which requires you to set the
// permission manually.
const authorizer = new casbinjs.Authorizer("manual");
```

Now we have an authorizer, `authorizer`. We can get permission rules from it by using the `authorizer.can()` and `authorizer.cannot()` APIs. The return values of these 2 APIs are JavaScript Promises ([details here](#)), so we should use the `then()` method of the return value like this:

```
result = authorizer.can("write", "data1");
result.then((success, failed) => {
```

The `cannot()` API is used in the same way:

```
result = authorizer.cannot("read", "data2");
result.then((success, failed) => {
    if (success) {
        console.log("you cannot read data2");
    } else {
        console.log("you can read data2");
    }
});
// output: you can read data2
```

In the code above, the `success` variable in the parameters means the request gets the result without throwing an error and doesn't mean that the permission rule is `true`. The `failed` variable is also unrelated to the permission rules. It only makes sense when something goes wrong in the process of the request.

You can refer to our [React example](#) to see a practical usage of Casbin.js.

Permission Object

Casbin.js will accept a JSON object to manipulate the corresponding permission of a visitor. For example:

```
{
    "read": ["data1", "data2"],
    "write": ["data1"]
}
```

The permission object above shows that the visitor can `read` the `data1` and `data2` objects, while they can only `write` the `data1` objects.

Advanced Usage

Casbin.js provides a perfect solution for integrating your frontend access-control management with your backend Casbin service.

Use the `auto` mode and specify your endpoint when initializing the Casbin.js `Authorizer`, it will automatically sync the permission and manipulate the frontend status.

```
const casbinjs = require('casbin.js');

// Set your backend Casbin service URL
const authorizer = new casbinjs.Authorizer(
    'auto', // mode
    {endpoint: 'http://your_endpoint/api/casbin'}
);

// Set your visitor.
// Casbin.js will automatically sync the permission with your
// backend Casbin service.
authorizer.setUser("Tom");

// Evaluate the permission
result = authorizer.can("read", "data1");
result.then((success, failed) => {
    if (success) {
        // Some frontend procedure ...
    }
});
```

Correspondingly, you need to expose an interface (e.g. a RestAPI) to generate the permission object and pass it to the frontend. In your API controller, call

`CasbinJs GetUserPermission` to construct the permission object. Here is an example in Beego:

 NOTE

Your endpoint server should return something like

```
{  
    "other": "other",  
    "data": "what you get from  
    `CasbinJsGetPermissionForUser`"  
}
```

```
// Router  
beego.Router("api/casbin", &controllers.APIController{},  
"GET:GetFrontendPermission")  
  
// Controller  
func (c *APIController) GetFrontendPermission() {  
    // Get the visitor from the GET parameters. (The key is  
    "casbin_subject")  
    visitor := c.Input().Get("casbin_subject")  
    // `e` is an initialized instance of Casbin Enforcer  
    c.Data["perm"] = casbin.CasbinJsGetPermissionForUser(e,  
    visitor)  
    // Pass the data to the frontend.  
    c.ServeJSON()  
}
```

 NOTE

Currently, the `CasbinJsGetPermissionForUser` API is only supported in Go Casbin and Node-Casbin. If you want this API to be supported in other

languages, please [raise an issue](#) or leave a comment below.

API List

setPermission(permission: string)

Set the permission object. Always used in `manual` mode.

setUser(user: string)

Set the visitor identity and update the permission. Always used in `auto` mode.

can(action: string, object: string)

Check if the user can perform `action` on `object`.

cannot(action: string, object: string)

Check if the user cannot perform `action` on `object`.

canAll(action: string, objects: Array<object>)

Check if the user can perform `action` on all objects in `objects`.

```
canAny(action: string, objects:  
Array<object>)
```

Check if the user can perform `action` on any one of the `objects`.

Why Casbin.js

People may wonder about the difference between Node-Casbin and Casbin.js. In a word, Node-Casbin is the core of Casbin implemented in the NodeJS environment, and it's normally used as an access-controlling management toolkit at the server ends. Casbin.js is a frontend library that helps you use Casbin to authorize your webpage users at the client side.

Normally, it is not proper to directly build up a Casbin service and do the authorization/enforcement tasks at a web frontend application due to the following problems:

1. When someone turns on the client, the enforcer will be initialized, and it will pull all the policies from the backend persistent layers. A high concurrency could bring tough pressure on the databases and cost a lot of network throughput.
2. Loading all policies to the client side could bring security risks.
3. It is difficult to separate the client and server as well as facilitate agile development.

We need a tool that eases the process of using Casbin at the frontend. Actually, the core of Casbin.js is the manipulation of the current user's permission at the client side. As you mentioned, Casbin.js does a fetch from a specified endpoint. This procedure will sync the permission of the user with the backend Casbin service. After having the permission data, developers can use Casbin.js interfaces

to manage the behavior of the user at the frontend side.

Casbin.js avoids the two problems mentioned above: Casbin service will no longer be pulled up repeatedly, and the size of passing messages between the client and the server is reduced. We also avoid storing all the policies at the frontend. The user can only access their own permission, but has no knowledge about the access-control model and other users' permissions. Besides, Casbin.js can also efficiently decouple the client and the server in authorization management.

Editor



Online Editor

Writing Casbin model and policy in a web browser



IDE Plugins

Casbin IDE plugins

Online Editor

You can also use the [online editor](#) to write your Casbin model and policy in your web browser. It provides functionality such as "syntax highlighting" and "code completion", just like an IDE for a programming language.

Editor Features

The online editor provides several powerful features to enhance your Casbin development experience:

AI-Powered Assistance

The editor includes AI assistance to help you write better Casbin models and policies.

The screenshot shows the Casbin AI Assistant interface. At the top, there are tabs for "Policy" (selected), "Logs", "Metrics", and "AI Assistant". The "AI Assistant" tab has a red box around it, and a large red arrow points from the "Ask AI" button in the top right towards the "Policy" tab area.

Policy

Node-Casbin (NodeJs) 5.37.0

AI Assistant

Ask AI

Provider Mode

Auto read:

Feedback

Policy Content:

```
1 p, alice, data1, read
2 p, bob, data2, write
```

Enforcement Result

Why this result

```
1 true Reason: ["alice", "data1"]
```

Ask AI

Done in 30.20ms

English

2025/9/16 09:49:13

The model content provided relates to Access Control Lists (ACL) using Node-Casbin in a Node.js environment. The request and policy definitions describe how access decisions are

Type message here

Chat with AI

2025/9/16 09:49:13

m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
Policy: Node-Casbin (NodeJs)
5.37.0
p, alice, data1, read
p, bob, data2, write
Request: alice, data1, read
Enforcement Result: Why this result
true Reason:
"alice", "data1", "read"

Engine Selection

Choose from multiple Casbin engines including Go, Java, Node.js, Python, Rust, and .NET implementations.

A screenshot of a web-based Casbin interface. At the top, there's a navigation bar with tabs for "Policy" (selected), "Upload" (blue icon), and "Edit" (red icon). The main title is "Node-Casbin (NodeJs) 5.37.0". On the left, there's a code editor window showing two lines of policy definitions:

```
1 p, alice,
2 p, bob, da
```

A large red arrow points from the text "Switch between different Casbin implementations to test your models across various platforms." towards this code editor area.

The right side of the interface is a sidebar with a red border containing a list of supported engines:

- Node-Casbin (NodeJs) 5.37.0**
- jCasbin (Java) v1.79.0 | (CLI v1.17.0)**
- Casbin (Go) v2.116.0 | (CLI v1.12.0)**
- Casbin-rs (Rust) v2.10.1 | (CLI v1.3.0)**
- PyCasbin (Python) v1.17.0 | (CLI v1.0.0)**
- Casbin.NET (C#) v1.4.0.0 | (CLI v1.0.0)**

On the far right, there's a vertical sidebar with a blue gradient and the word "Feedback".

Multiple Engine Support

Switch between different Casbin implementations to test your models across various platforms.

The screenshot shows a web-based interface for managing Casbin policies. At the top, there's a navigation bar with tabs for "Policy" (selected), "Upload" (blue icon), "Download" (red icon), and "Node-Casbin (NodeJs) 5.37.0" (highlighted in red). Below the navigation is a code editor containing a simple policy:

```
1 p, alice, data1, read
2 p, bob, data2, write
```

To the right of the code editor is a dropdown menu with a red border and a red arrow pointing towards it from the bottom-left. The menu lists several Casbin engines, each with a checked checkbox:

- Node-Casbin (NodeJS) (selected)
- jCasbin (Java)
- Casbin (Go)
- Casbin-rs (Rust)
- PyCasbin (Python)
- Casbin.NET (C#)

On the far right of the interface, there are icons for "Feedback" and "Ask AI".

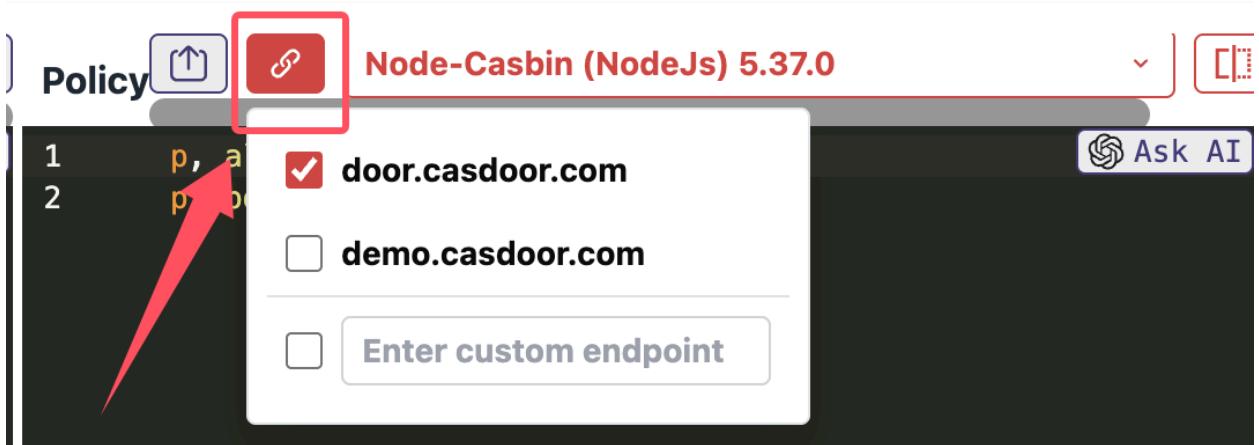
The screenshot shows the "Enforcement Result" section of the Casbin UI. It displays five lines of code, each representing the result of an access check for the same subject and object:

```
1 // node Engine Result
2 true Reason: ["alice", "data1", "read"]
3 // -----
4
5 // java Engine Result
6 true Reason: ["alice", "data1", "read"]
7 // -----
8
9 // go Engine Result
10 true Reason: ["alice", "data1", "read"]
11 // -----
12
13 // rust Engine Result
14 true Reason: ["alice", "data1", "read"]
15 // -----
```

Each line starts with a comment indicating the engine type (node, java, go, rust) followed by the result (true) and the reason (an array of [subject, object, action]). The "Ask AI" button is visible at the top right of this section.

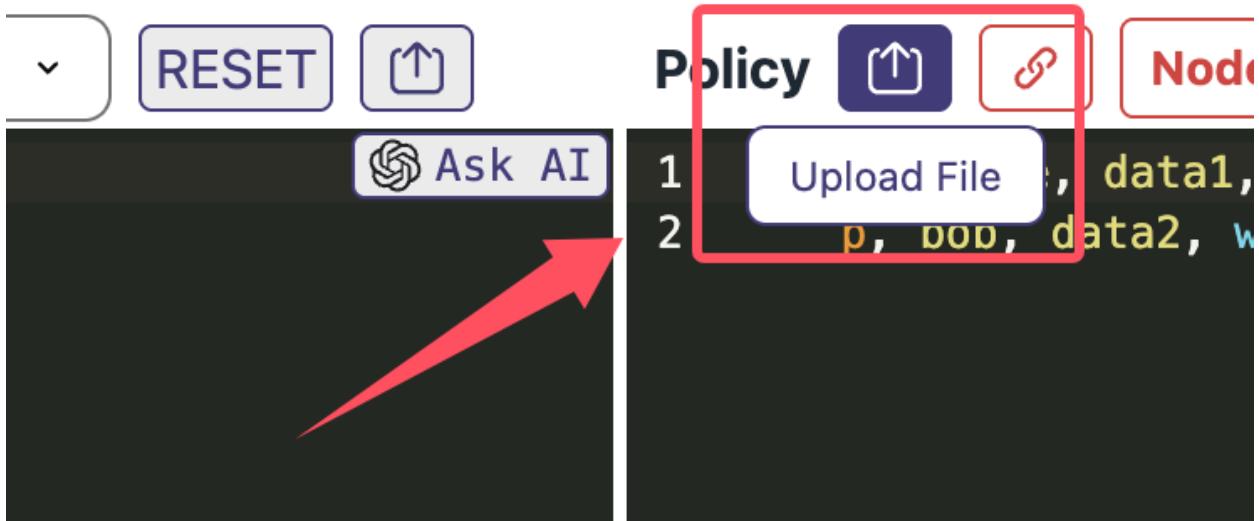
Endpoint Configuration

Configure custom endpoints for testing your Casbin models against different environments.



File Upload

Upload existing model and policy files to quickly get started with the editor.



Use Custom Functions

If you need to use a customized matching function (e.g. "RBAC with Patterns"), you can add it by clicking the "Add Role Matching" button at the bottom left corner of the editor.

matchingForGFunction

X

```
1  (user, role) => {
2      return user.department ===
3          role.department;
4 }
```

Add
Function

Add Role
Matching

Add Domain
Matching



If you want to write the equivalent code, you need to specify the pattern matching function through the relevant API. Refer to [RBAC with Pattern](#) for more information.

NOTE

The editor supports multiple Casbin implementations, including [Node-Casbin \(Node.js\)](#), [JCasbin \(Java\)](#), [Casbin \(Go\)](#), [Casbin-rs \(Rust\)](#), [PyCasbin \(Python\)](#), and [Casbin.NET \(C#\)](#). You can switch between different implementations in the upper right corner to test your model and policy.

Although the editor validates through a remote CLI, due to environment differences, the validation results may differ slightly from the results you get in your local environment. If you encounter any issues, please submit them to the corresponding Casbin implementation repository.

IDE Plugins

We offer plugins for the following IDEs:

JetBrains IDEs

- Download: <https://plugins.jetbrains.com/plugin/14809-casbin>
- Source code: <https://github.com/will7200/casbin-idea-plugin>

VSCode

- Source code: <https://github.com/casbin/casbin-vscode-plugin>

More

Our Adopters

Casbin's Adopters

Contributing

Contributing to Casbin

Privacy Policy

Casbin Website Privacy Policy

Terms of Service

Casbin Terms of Service

Refund Policy

Casbin Website Refund Policy

Our Adopters

Direct Integration

Go Java Node.js Python

Name	Description	Model	Policy
VMware Harbor	VMware's open source trusted cloud native registry project that stores, signs, and scans content.	Code	Beego ORM
Intel RMD	Intel's resource management daemon.	.conf	.csv
VMware Dispatch	A framework for deploying and managing serverless style applications.	Code	Code
Skydive	An open source real-time network topology and protocols analyzer.	Code	.csv
Argo CD	GitOps continuous delivery for Kubernetes.	.conf	.csv
Muxi Cloud	PaaS of Muxi Cloud, an easier way to manage Kubernetes clusters.	.conf	Code
EngineerCMS	A CMS to manage knowledge for engineers.	.conf	SQLite

Name	Description	Model	Policy
Cyber Auth API	A Golang authentication API project.	.conf	.csv
Metadata DB	BB archive metadata database.	.conf	.csv
Qilin API	ProtocolONE's licenses management tool for game content.	Code	.csv
Devtron Labs	Software Delivery Workflow For Kubernetes.	.conf	Xorm
Name	Description	Model	Policy
lighty.io	OpenDaylight's solution for SDN controllers.	README	N/A
Name	Description	Model	Policy
Notadd	A micro-service development architecture based on Nest.js.	.conf	DB adapter
ARC API	A Catalog of Microservices based on Loopback Created by SourceFuse.	Usage	Provider
Name	Description	Model	Policy
dtrace	EduScaled's tracing system.	Commit	N/A

Integration via Plugin

Name	Description	Plugin	Model	Policy
Docker	The world's leading software container platform	casbin-authz-plugin (recommended by Docker)	.conf	.csv
Gobis	Orange 's lightweight API Gateway written in go	casbin	Code	Request

Contributing

Casbin is a powerful authorization library that supports access control models with implementations in many programming languages. If you are proficient in any programming language, you can contribute to the development of Casbin. New contributors are always welcome.

Currently, there are two main types of projects:

- **Algorithms-oriented projects** - These projects involve implementing algorithms in different programming languages. Casbin supports a wide range of languages, including Golang, Java, C++, Elixir, Dart, and Rust, along with their related products.

 The Gopher logo, a blue cartoon gopher character with large white eyes and a small brown nose, standing next to the word "Go".	 The Java logo, which consists of a steaming coffee cup with orange steam rising from it.
Casbin	jCasbin
Production-ready	Production-ready



python™



PyCasbin

Casbin.NET

Production-ready

Production-ready

- Application-oriented projects - These projects are related to applications built on top of Casbin.

Project	Demo	Details	Skill Stacks
Casdoor	Casdoor	Casdoor is a UI-first centralized authentication/Single-Sign-On (SSO) platform based on OAuth 2.0/OIDC.	JavaScript + React and Golang + Beego + SQL
Casnnode	Casbin Forum	Casnnode is a next-generation forum software.	JavaScript + React and Golang + Beego + SQL

Project	Demo	Details	Skill Stacks
Casbin OA	OA system	Casbin-OA is an official manuscript processing, evaluation, and display system for Casbin technical writers.	JavaScript + React and Golang + Beego + MySQL
Casbin Editor	Casbin Editor	Casbin-editor is a web-based editor for Casbin models and policies.	TypeScript + React

Getting Involved

There are many ways to contribute to Casbin. Here are some ideas to get started:

- **Use Casbin and report issues!** When using Casbin, report any issues you encounter to help promote the development of Casbin. Whether it's a bug or a proposal, filing an issue on [GitHub](#) is recommended. However, it would be better to have a discussion first on [Discord](#) or [GitHub Discussions](#) before filing an issue.

Note: When reporting an issue, please use English to describe the details of your problem.
- **Help with documentation!** Contributing to the documentation is a good starting point for your contribution.
- **Help solve issues!** We have prepared a table containing easy tasks suitable for beginners, with different levels of challenges labeled with different tags. You can check the table [here](#).

Pull Requests

Casbin uses GitHub as its development platform, so pull requests are the main way to contribute.

Before opening a pull request, there are a few things you need to know:

- Explain why you are sending the pull request and what it will do for the repository.
- Make sure the pull request does only one thing. If there are multiple changes, please split them into separate pull requests.
- If you are adding new files, please include the Casbin license at the top of the new file(s).

```
// Copyright 2021 The casbin Authors. All Rights Reserved.  
//  
// Licensed under the Apache License, Version 2.0 (the  
"License");  
// you may not use this file except in compliance with the  
License.  
// You may obtain a copy of the License at  
//  
//     http://www.apache.org/licenses/LICENSE-2.0  
//  
// Unless required by applicable law or agreed to in  
writing, software  
// distributed under the License is distributed on an "AS  
IS" BASIS,  
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either  
express or implied.
```

- In projects like [Casdoor](#), [Casnodel](#), and [Casbin OA](#), you may need to set up a demo to show the maintainer how your pull request helps with the development of the project.
- When opening a pull request and committing your contribution, it is recommended to use semantic commits with the following format:
`<type>(<scope>): <subject>`. The `<scope>` is optional. For more detailed usage, please refer to [Conventional Commits](#).

License

By contributing to Casbin, you agree that your contributions will be licensed under the Apache License.

Privacy Policy

Your privacy is important to us. It is Casbin's policy to respect your privacy regarding any information we may collect from you across our [docs website](#), as well as other sites we own and operate.

We only ask for personal information when we truly need it to provide a service to you. We collect it by fair and lawful means, with your knowledge and consent. We also let you know why we are collecting it and how it will be used.

We only retain collected information for as long as necessary to provide you with your requested service. The data we store will be protected within commercially acceptable means to prevent loss and theft, as well as unauthorized access, disclosure, copying, use, or modification.

We do not share any personally identifying information publicly or with third-parties, except when required to by law.

Our website may link to external sites that are not operated by us. Please be aware that we have no control over the content and practices of these sites and cannot accept responsibility or liability for their respective privacy policies.

You are free to refuse our request for your personal information, with the understanding that we may be unable to provide you with some of your desired services.

Your continued use of our website will be regarded as acceptance of our practices regarding privacy and personal information. If you have any questions about how we handle user data and personal information, feel free to contact us.

This policy is effective as of 29th June 2020.

Terms of Service

1. Terms

By accessing the website at <https://casbin.org>, you are agreeing to be bound by these terms of service, all applicable laws and regulations, and agree that you are responsible for compliance with any applicable local laws. If you do not agree with any of these terms, you are prohibited from using or accessing this site. The materials contained in this website are protected by applicable copyright and trademark law.

2. Use License

a. Permission is granted to temporarily download one copy of the materials (information or software) on Casbin's website for personal, non-commercial transitory viewing only. This is the grant of a license, not a transfer of title, and under this license you may not:

- i. modify or copy the materials;
- ii. use the materials for any commercial purpose, or for any public display (commercial or non-commercial);
- iii. attempt to decompile or reverse engineer any software contained on Casbin's website;
- iv. remove any copyright or other proprietary notations from the materials; or
- v. transfer the materials to another person or "mirror" the materials on any other server.

b. This license shall automatically terminate if you violate any of these restrictions and may be terminated by Casbin at any time. Upon terminating your viewing of these materials or upon the termination of this license, you

must destroy any downloaded materials in your possession whether in electronic or printed format.

3. Disclaimer

- a. The materials on Casbin's website are provided on an 'as is' basis. Casbin makes no warranties, expressed or implied, and hereby disclaims and negates all other warranties including, without limitation, implied warranties or conditions of merchantability, fitness for a particular purpose, or non-infringement of intellectual property or other violation of rights.
- b. Further, Casbin does not warrant or make any representations concerning the accuracy, likely results, or reliability of the use of the materials on its website or otherwise relating to such materials or on any sites linked to this site.

4. Limitations

In no event shall Casbin or its suppliers be liable for any damages (including, without limitation, damages for loss of data or profit, or due to business interruption) arising out of the use or inability to use the materials on Casbin's website, even if Casbin or a Casbin authorized representative has been notified orally or in writing of the possibility of such damage. Because some jurisdictions do not allow limitations on implied warranties, or limitations of liability for consequential or incidental damages, these limitations may not apply to you.

5. Accuracy of materials

The materials appearing on Casbin's website could include technical, typographical, or photographic errors. Casbin does not warrant that any of the materials on its website are accurate, complete or current. Casbin may make changes to the materials contained on its website at any time without

notice. However Casbin does not make any commitment to update the materials.

6. Links

Casbin has not reviewed all of the sites linked to its website and is not responsible for the contents of any such linked site. The inclusion of any link does not imply endorsement by Casbin of the site. Use of any such linked website is at the user's own risk.

7. Modifications

Casbin may revise these terms of service for its website at any time without notice. By using this website you are agreeing to be bound by the then current version of these terms of service.

8. Governing Law

These terms and conditions are governed by and construed in accordance with the laws of San Francisco, CA and you irrevocably submit to the exclusive jurisdiction of the courts in that State or location.

Refund Policy

In most cases, payments for Casbin subscriptions are not refundable.

If you have an issue with your account or think there has been an error in billing, please [contact support](#) for assistance.