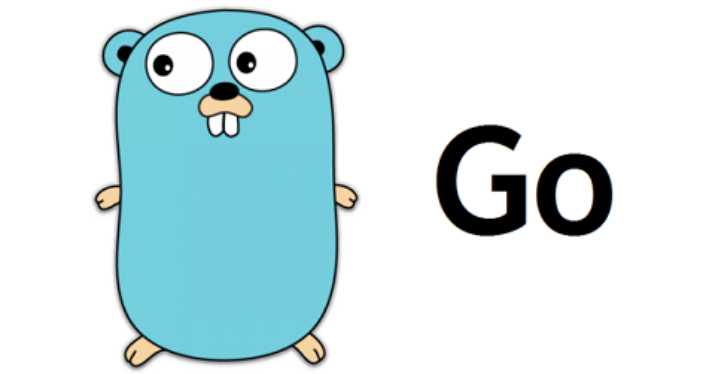
Overview

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 down 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 Casbin is used in. The developer/ administrator has the complete control over the layout, execution and conditions for authorization which is 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:

	
Casbin	jCasbin
production-ready	production-ready

PyCasbin	Casbin.NET
production-ready	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. But 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 which 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 its 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 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. Multiple built-in operators to support the rule matching. For example, `keyMatch` can map a resource key `/foo/bar` to the pattern `/foo*`.

What Casbin does NOT do:

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

It's more convenient for the project to manage their list 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. This will 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
cargo add tokio // make sure you activate its `macros` feature
```

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 report Error: Your user does not have write permissions in /usr/local/lib/luarocks/

rocks -- you may want to run as a privileged user or use your local tree with --local. you can add --local behind your command like this to fix:

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

New a Casbin enforcer

Casbin uses configuration files to set the access control model.

It has two configuration files, `model.conf` and `policy.csv`. Among them, `model.conf` stores our access model, and `policy.csv` stores our specific user permission configuration. The use of Casbin is very refined. Basically, we just need one main structure: **enforcer**. When constructing this structure, `model.conf` and `policy.csv` will be loaded.

In another word, to new a Casbin enforcer, you must provide a Model and an Adapter.

Casbin has a FileAdapter, see Adapter for more information.

- Use the Model file and default FileAdapter:

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

Rust Lua

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

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

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

```node
import { newEnforcer } from 'casbin';

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

```php
require_once './vendor/autoload.php';

use Casbin\Enforcer;

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

```python
import casbin

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

```dotnet
using NetCasbin;

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

```cpp
#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)/casbin")
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 SQLLite 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

```go
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"}})
```

```java
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")
```

```
Roles 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)**. So switching or upgrading the authorization mechanism for a project is just 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, Matchers) describing the relationship between resources and users.

Request

Define 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}`

It actually defines the parameter name and order which we should provide for access control matching function.

Policy

Define the model of the access strategy. In fact, it defines 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, then the result field in the policy file will not be read, and the matching policy result will be allowed by default.

Matcher

Matching rules of 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, that is, if they can be found in the policy, then the policy result (`p.eft`) is returned. The result of the strategy will be saved in `p.eft`.

Effect

It can be understood as a model in which a logical combination judgment is performed again on the matching results of Matchers.

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

This sentence 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. ACL's model CONF is:

```
# 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 ACL model is like:

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

It means:

- alice can read data1
- bob can write data2

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

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

Furthermore, if you are using ABAC, you can try operator `in` like the following in 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 **SHOULD** make sure that the length of the array is **MORE** than **1**, otherwise there will cause it to 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 midleware
- Gin Tutorial Chapter 10: Learn Casbin basic models in 30 minutes
- Gin Tutorial Chapter 11: Coding, API and custom function in Casbin
- 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)
- Role-based access control for HTTP based on Casbin (in Chinese)
- Learning iris + Casbin from scratch

VMware Harbor

- Casbin: Golang access control framework (in Chinese)
- Access control in Harbor (in Chinese)

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

Laravel

- Laravel authorization: authorization library supporting ACL, RBAC, ABAC and other models
- 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

Model

Supported Models

Supported models of Casbin

Syntax for Models

Syntax for Models

Effector

Effector is the interface for Casbin effectors

Function

Using the built-in functions or specifying your own function

RBAC

Casbin RBAC usage

RBAC with Pattern

RBAC with Pattern

RBAC with Domains

RBAC with domains usage

Casbin RBAC vs. RBAC96

Difference between Casbin RBAC and RBAC96

ABAC

ABAC based on Casbin

Priority Model

Priority Model

Super Admin

Super Admin is the administrator of the whole system, we can use it in models like RBAC, ABAC and RBAC with domains etc

Supported Models

1. ACL (Access Control List)
2. ACL with superuser
3. **ACL without users**: especially useful for systems that don't have authentication or user log-ins.
4. **ACL without resources**: some scenarios may target for a type of resources instead of an individual resource by using permissions like `write-article`, `read-log`. It doesn't control the 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 role sets 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. RESTful: supports paths like `/res/*`, `/res/:id` and HTTP methods like `GET`, `POST`, `PUT`, `DELETE`.
10. **Deny-override**: both allow and deny authorizations are supported, deny overrides the allow.
11. **Priority**: the policy rules can be prioritized like 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

Model	Model file	Policy file
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
ABAC	abac_model.conf	N/A
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

Model	Model file	Policy file
Subject-Priority	subject_priority_model.conf	subject_priority_policyl.csv

Syntax for Models

- A model CONF should have at least four sections: `[request_definition], [policy_definition], [policy_effect], [matchers]`.
- If a model uses RBAC, it should also add the `[role_definition]` section.
- A model CONF can contain comments. The comments start with `#`, and `#` will comment the rest of the line.

Request definition

`[request_definition]` is the definition for the access request. It defines the arguments in `e.Enforce(...)` function.

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

`sub, obj, act` represents the classic triple: accessing entity (Subject), accessed resource (Object) and the access method (Action). However, you can customize your own request form, like `sub, act` if you don't need to specify an particular resource, or `sub, sub2, obj, act` if you somehow have two accessing entities.

Policy definition

`[policy_definition]` is the definition for the 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`, e.g., `p`, `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)
```

> TIP
>
> The elements in a policy rule are always regarded as `string`. If you have any question about this, please see the discussion at: https://github.com/casbin/casbin/issues/113

Policy effect

`[policy_effect]` is the definition for the policy effect. It defines 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 if there's any matched policy rule of `allow`, the final effect is `allow` (aka allow-override). `p.eft` is the effect for a policy, it can be `allow` or `deny`. It's optional and the default value is `allow`. So as we didn't specify it above, it uses the default value.

Another example for policy effect is:

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

It means if there's no matched policy rules of `deny`, the final effect is `allow` (aka deny-override). `some` means: if there exists one matched policy rule. `any` means: all matched policy rules (not used here). The policy effect can even be connected with logic expressions:

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

It means at least one matched policy rule of `allow`, and there is no matched policy rule of `deny`. So in this way, both the allow and deny authorizations are supported, and the deny overrides.

> NOTE
>
> Although we designed the syntax of policy effect as above, the current

> implementations only use hard-coded policy effect, as we found there's no much need for that sort of flexibility. So for now, you must use one of the built-in policy effects instead of customizing your own one.

The supported built-in policy effects are:

Policy effect	Meaning	Example
some(where (p.eft == allow))	allow-override	ACL, RBAC, etc.
!some(where (p.eft == deny))	deny-override	Deny-override
some(where (p.eft == allow)) && !some(where (p.eft == deny))	allow-and-deny	Allow-and-deny
priority(p.eft) || deny	priority	Priority
subjectPriority(p.eft)	priority base on role	Subject-Priority

Matchers

`[matchers]` is the definition for policy matchers. The matchers are expressions. It defines how the policy rules are evaluated against the request.

```
[matchers]
```

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

You can use arithmetic like `+, -, *, /` and logical operators like `&&, ||, !` in matchers.

Orders of expressions in matchers

The order of expressions can greatly affect performance. Look at the following example for details:

```
const rbac_models = `
[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
`

func TestManyRoles(t *testing.T) {

    m, _ := model.NewModelFromString(rbac_models)
    e, _ := NewEnforcer(m, false)

    roles := []string{"admin", "manager", "developer", "tester"}
```

The enforce time may be very 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 can 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
    rbac_api_test.go:599: RESPONSE abu
/projects/1        GET :  true IN: 786.635µs
```

Multiple sections type

If you need multiple policy definitions or multiple matcher, you can use like `p2`, `m2`. In fact, all of the above four sections can use multiple types and the syntax is `r`+number, such as `r2`, `e2`. By default these four sections should correspond one to one. Such as your `r2` will only use matcher `m2` to match policies `p2`.

You can pass in `EnforceContext` as the first parameter of `enforce` method to specify the types, the `EnforceContext` is like this

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;
    }
}
```

Example usage, see model and policy, the request is as follows

Go Node.js Java

```
// Pass in a suffix as 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 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.seteType("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 Grammer

You could also use `in`, the only operator with a text name. This operator checks the right-hand side array to see if it contains a value that is equal to the left-side value. Equality is determined by the use of the == operator, and this library doesn't check types between the values. Any two values, when cast to interface{}, and can still be checked for equality with == 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. Besides all the model syntax provided here, those expression evaluators may provide extra functionality, which may be not supported by another language or implementation. Use it at your own risk.

The expression evaluators used by each Casbin implementation are:

Implementation	Language	Expression evaluator
Casbin	Golang	https://github.com/Knetic/govaluate
jCasbin	Java	https://github.com/killme2008/aviator
Node-Casbin	Node.js	https://github.com/donmccurdy/expression-eval
PHP-Casbin	PHP	https://github.com/symfony/expression-

Implementation	Language	Expression evaluator
		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 performance issue about Casbin, it's probably caused by the low efficiency of the expression evaluator. You can both send issue to Casbin or the expression evaluator directly for advice to speed up. See Benchmarks section for details.

Effector

Effect is the result of a policy rule. And the `Effector` is the interface for Casbin effectors.

`MergeEffects()`

MergeEffects merges 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 being merged by this function(Initialized as `Indeterminate`).
- `explainIndex` is the index of `eft` which is `Allow` or `Deny`.(Initialized as `-1`)
- `err` is used to check if the effect is supported.
- `expr` is the policy effects stored as `string`
- `effects` is the array of the Effect which can be `Allow`, `Indeterminate` or `Deny`

- `matches` is the array showing that if the result is matching the policy.
- `policyIndex` is the index of policy in the model.
- `policyLength` is the length of the policy.

The code above illustrates how can we pass the parameters to the `MergeEffects` function and the function will process the effects and matches based on the `expr`

To deploy an Effector, we can do this:

Go

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

The basic idea of the `MergeEffects` indicates that if the `expr` can match the results which means that the `p_eft` is `allow`, then we can merge all effects at last. And if there are no deny rules are matched, then we allow.

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

Function

Functions in matchers

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

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

It returns a boolean indicating whether `url` matches `pattern`.

The supported built-in functions are:

Function	url	pattern	Example
keyMatch	a URL path like `/alice_data/resource1`	a URL path or a `*` pattern like `/alice_data/*`	keymatch_model.conf/keymatch_policy.csv
keyMatch2	a URL path like `/alice_data/resource1`	a URL path or a `:` pattern like `/alice_data/:resource`	keymatch2_model.conf/keymatch2_policy.csv
keyMatch3	a URL path like `/alice_data/resource1`	a URL path or a `{}` pattern like `/alice_data/{resource}`	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L171-L196
keyMatch4	a URL path like `/alice_data/123/book/123`	a URL path or a `{}` pattern like `/alice_data/{id}/book/{id}`	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L208-L222
regexMatch	any string	a regular expression pattern	keymatch_model.conf/keymatch_policy.csv
ipMatch	an IP address like `192.168.2.123`	an IP address or a CIDR like `192.168.2.0/24`	ipmatch_model.conf/ipmatch_policy.csv
globMatch	a path-like path like `/alice_data/resource1`	a glob pattern like `/alice_data/*`	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"`, `KeyGet3("/resource1_admin/action", "/{res}_admin/*", "res")` will return `"resource1"`. As for `KetGet`, which takes two parameters, `KeyGet("/resource1/action", "/*)` will return `"resource1/action"`.

Function	url	pattern	key_name	example
keyGet	a URL path like `/proj/resource1`	a URL path or a `*` pattern like `/proj/*`	\	keyget_model.conf/keymatch_policy.csv
keyGet2	a URL path like `/proj/resource1`	a URL path or `:` pattern like `/prooj/:resource`	key name specified in the pattern	keyget2_model.conf/keymatch2_policy.csv
keyGet3	a URL path like `/proj/res3_admin/`	a URL path or `{}` pattern like `/proj/{resource}_admin/*`	key name specified in the pattern	https://github.com/casbin/casbin/blob/7bd496f94f5a2739a392d333a9aaaa10ae397673/util/builtin_operators_test.go#L209-L247

See details for 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 return 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)

    return (bool)(KeyMatch(name1, name2)), nil
}
```

At last, 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

`[role_definition]` is the definition for the RBAC role inheritance relations. Casbin supports multiple instances of RBAC systems, e.g., 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.

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 a RBAC system, and `g2` is another RBAC system. `_, _` means there are two parties inside an inheritance relation. As a common case, you usually use `g` alone if you only need roles on users. You can also use `g` and `g2` when you need roles (or groups) on 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 `alice` inherits/is a member of role `data2_admin`. `alice` here 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 below:

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

It means `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 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 a 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`, `B` has role `C`, then `A` has role `C`. This transitivity is infinite for now.

> TOKEN NAME CONVENTION
>
> Conventionally subject token name in policy definition is `sub` and placed in the beginning. Now Golang Casbin supports customized token name & place. If the subject token name is `sub`, the subject token can be placed at an arbitrary place and no extra action needs. 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 for sub
[policy_definition]
p =  obj, act, subject
```

```
e.SetFieldIndex("p", constant.SubjectIndex, 2) // index
start 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, meaning if `alice` has `role1`, `role1` has `role2`, then `alice` will also have `role2` and inherit its permissions.

Here is a concept called hierarchy level. So the hierarchy level for this example is 2. For the built-in role manager in Casbin, you can specify the max hierarchy level. The default value is 10. It means 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 role from user in its RBAC. They are all treated as strings. If you only use single-level RBAC (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 just list all `u` and all `r` respectively in all `g, u, r` rules.

But if you are using 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 user and role with same name. You can add a prefix to role like `role::admin` before passing it to Casbin. So 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 directly assigning them in a policy rule, we call such type of assignment as `implicit`. To query such implicit relations, you need to use these 2 APIs: `GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()` instead of `GetRolesForUser()` and `GetPermissionsForUser()`. For more details, please see this GitHub issue.

Use pattern matching in RBAC

See RBAC with Pattern

Role manager

See 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

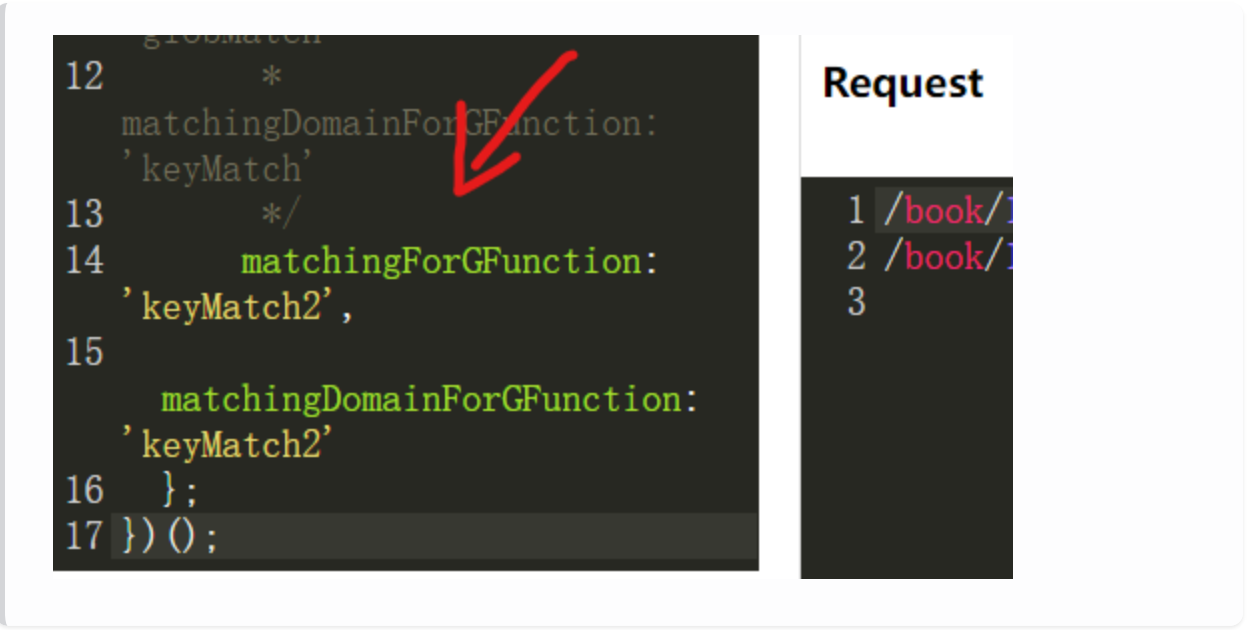
```
e.AddNamedDomainMatchingFunc("g","KeyMatch2",util.KeyMatch2)
```

- use all pattern

  just combined use of two APIs

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

> NOTE
>
> If you use the online editor, it specifies the pattern matching function in the lower left corner.



Use pattern matching in RBAC

Sometimes, you want some subjects, object or domains/tenants with the specific pattern to be automatically granted to 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 matcher. Then we will use policy like:

```
p, alice, book_group, read
g, /book/1, book_group
g, /book/2, book_group
```

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 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 to 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 work. 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 the access to two domains.

By using pattern matching, especially in the scenarios which is more complicated and there are a lot of domains or objects we need to take into consideration, we can implement the `policy_definition` more elegant and effective.

RBAC with Domains

Role definition with domains tenants

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

The role definition with domains/tenants should be something like:

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

The 3rd `_` means the name of domain/tenant, 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
```

It means `admin` role in `tenant1` can read `data1`. And `alice` has `admin` role in `tenant1`, and has `user` role in `tenant2`. So 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 below:

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

Please see the rbac_with_domains_model.conf for examples.

> **TOKEN NAME CONVENTION**
>
> Note: Conventionally domain token name in policy definition is `dom` and placed as the second token(`sub, dom, obj, act`). Now Golang Casbin supports customized token name & place. If the domain token name is `dom`, the domain token can be placed at an arbitrary place and no extra action needs. 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
> start from 0
> users := e.GetAllUsersByDomain("domain1") // without
> SetFieldIndex, it will raise an error
> ```

Casbin RBAC vs. RBAC96

Casbin RBAC and RBAC96

In this document, we'll compare Casbin RBAC with RBAC96.

Casbin RBAC supports almost all the features of RBAC96, and added new features above that.

RBAC version	Support Level	Description
RBAC0	fully supported	RBAC0 is the basic version of RBAC96. It clarified the relationship between Users, Roles and Permissions.
RBAC1	fully supported	RBAC1 added role hierarchies on RBAC0, meaning if `alice` has `role1`, `role1` has `role2`, then `alice` will also have `role2` and inherit its permissions.
RBAC2	mutually exclusive handling is supported (like this), but quantitative limits are not	RBAC2 added constrains on RBAC0. So RBAC2 can handle mutually exclusions found in policies.

RBAC version	Support Level	Description
RBAC3	mutually exclusive handling is supported (like this), but quantitative limits are not	RBAC3 is a combination of RBAC1 and RBAC2. RBAC3 supports role hierarchies and constrains in RBAC1 and RBAC2.

Difference between Casbin RBAC and RBAC96

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

   In Casbin, both the User and the Role are treated as strings. If you wrote a policy file like this:

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

   and call method `GetAllSubjects()` like this (`e` is an instance of Casbin Enforcer):

   ```
   e.GetAllSubjects()
   ```

   then you will get the return value below:

```
[admin alice]
```

Because in Casbin, Subjects included Users and Roles.

However, if you call method `GetAllRoles()` like this:

```
e.GetAllRoles()
```

then you will get the return value below:

```
[admin]
```

And now you know there is a distinction between Users and Roles in Casbin, but is not as sharp as in RBAC96. Of course you can add some prefix to your policies like `user::alice`, `role::admin` to clarify their relationships.

2. Casbin RBAC provides more permissions than RBAC96

   Only 7 permissions are defined in RBAC96: read, write, append, execute, credit, debit, inquiry.

   However, in Casbin, we treat permissions as strings. This way, you can create some permissions suit you better.

3. Casbin RBAC supports domains

   In Casbin, you can do authorizations by domains. This feature made your Access Control Model more flexible.

ABAC

What is the ABAC model actually?

ABAC is `Attribute-Based Access Control`, meaning you can use the attributes (properties) of the subject, object or action instead of themselves (the string) to control the access. You may already hear of a complicated ABAC access control language named XACML. Compared to XACML, Casbin's ABAC is very simple: in ABAC, you can use structs (or class instances based on the programming language) instead of string for model elements.

Use the official ABAC example for 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
```

We use `r.obj.Owner` instead of `r.obj` in the matcher. The `r.obj` passed in the `Enforce()` function will be a struct or class instance instead of 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
}
```

How to use ABAC?

Simply speaking, 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 the argument in 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 comma in policy which conflicts with csv's separator and we need to escape it. Casbin parses policy file through csv library, you could surround statement with quotation marks. For example, `"keyMatch("bob", r.sub.Role)"` will not be split.

Scaling the model for complex and large number of ABAC rules.

The above instance of ABAC implementation is at its core very simple, but oftentimes the authorization system needs a very complex and large number of ABAC rules. To fit this necessity the above implementation will increase the verbosity of the model to a large extent. So, it's wise to add the rules in the policy instead of in the model. This is done by introducing a `eval()` functional construct. Below is the example instance to manage such ABAC models.

This is the definition of the `CONF` file used for defining the ABAC model.

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

[policy_definition]
p = sub_rule, obj, act

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

[matchers]
m = eval(p.sub_rule) && r.obj == p.obj && r.act == p.act
```

Here, `p.sub_rule` is of type struct or class(user-defined type) which consists of 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 which is 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
```

Priority Model

Casbin support load policies with priority.

Load Policy with Priority Implicitly

It's quite simple, the order determines the priority, policy appeared earlier has higher priority.

model.conf:

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

Load Policy with Priority Explicitly

> Also see: casbin#550

The smaller priority value will have a higher priority. If there's a non-numerical character in priority, it will be in the last, rather than throw an error.

> **TOKEN NAME CONVENTION**
>
> The priority token name in policy definition is "priority" conventionally. A customized one requires invoking `e.SetFieldIndex()` and reload policies (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
failed 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`
```

Now, explicit priority only support `AddPolicy` & `AddPolicies`, if `UpdatePolicy` been called, you shouldn't change the priority attribute.

model.conf:

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

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 // for `p, 1, alice, data1, write,
allow` has highest priority
bob, data2, read --> false
bob, data2, write --> true // for bob has role of
`data2_allow_group` which has 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 one 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(the role corresponding) appeared earlier has higher priority. more details also see casbin#833

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

The role hierarchy like this:

```
role: root
 └─ role: admin
    ├─ role editor
    │  └─ user: jane
    │
    └─ role: subscriber
       └─ user: alice
```

The priority automatically like this:

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

Super Admin

Super Admin is the administrator of the whole system, we can use it in models like RBAC, ABAC and RBAC with domains etc. 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"
```

It illustrates that as for the defined `request_definition`, `policy_definition`, `policy_effect` and `matchers`, Casbin judges if the request can match the policy, or most importantly, if the `sub` is root. Once the judgment is correct, the authorization then is allowed, and the user has permission to do everything.
Just like the root of Linux systems, after the users are authorized as root, then we have the permissions to access all the files and settings. So if we want `sub` to have the full access to the whole system, we can let it become the super admin, then the `sub` has the permission to do everything.

Storage

Model Storage

Model storage

Policy Storage

Policy Storage

Policy Subset Loading

Loading filtered policy

Model Storage

Unlike the policy, the model can be loaded only, it cannot be saved. Because we think the model is not a dynamic component and should not be modified at run-time, so we don't implement an API to save the model into a storage.

However, the good news is, 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's easy to understand for beginners and convenient for sharing when you ask Casbin team for help.

The content of the `.CONF` file examples/rbac_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
```

Then you can load the model file as:

```
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 `.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 files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")

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

Load model from string

Or you can just load the entire model text from a multi-line string. The good point for this way 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 files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")
```

Policy Storage

In Casbin, the policy storage is implemented as an adapter.

Load policy from .CSV file

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

The content of the `.CSV` file examples/rbac_policy.csv:

```
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 it in double quotes, for example:

```
p, alice, "data1,data2", read   --correct
p, alice, data1,data2, read     --incorrect ("data1,data2"
should be a whole)
```

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 (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				

The meaning of each column

- `id`: Only exists in the database as the primary key. Not 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 adapter. If it is not enough for you, please submit an issue to the corresponding adapter repository.

Adapter Details

More details about the use of adapter api and database table structure design,

Please go to: /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 in storage based on a given filter. This allows for efficient policy enforcement in large, multi-tenant environments when 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)
```

There's another method support subset loading feature: `LoadIncrementalFilteredPolicy`. `LoadIncrementalFilteredPolicy` is similar to LoadFilteredPolicy, but it does not clear previous loaded policy, only append.

Extensions

Enforcers

Enforcer is the main structure in Casbin, It acts as an interface for users to make operations on policy rules and models

Adapters

Supported adapters and usage

Watchers

Keeping consistence between multiple Casbin enforcer instances

Dispatchers

Dispatcher 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

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

Supported enforcers

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

Go

Enforcer	Author	Description
Enforcer	Casbin	`Enforcer` is the basic structure for users to interact with Casbin policies and models. You can find more details of `Enforcer`'s API at here.
CachedEnforcer	Casbin	`CachedEnforcer` is based on `Enforcer`. It supports to cache the evaluation result of a request in memory by a map and clear caches in a specified expire time. Moreover, it is guaranted to be thread-safe by a Read-Write lock. You can use `EnableCache` to enable to cache evaluation results (default is enabled).

Enforcer	Author	Description
		`CachedEnforcer`'s other API is the same as `Enforcer`'s.
DistributedEnforcer	Casbin	`DistributedEnforcer` supports multiple instances in distributed clusters. It wraps `SyncedEnforcer` for dispatcher. You can find more details about dispather at here.
SyncedEnforcer	Casbin	`SyncedEnforer` is based on `Enforcer` and provides synchronized access. It is thread-safe.

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

Adapter	Type	Author	AutoSave	Description
Adapter				Server are supported by GORM
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 `master` branch and Oracle is supported in `oracle` 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

Adapter	Type	Author	AutoSave	Description
				GoFrame ORM
GoFrame V2 Adapter	ORM	@hailaz	✅	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Filtered PostgreSQL Adapter	SQL	Casbin	✅	For PostgreSQL
Filtered pgx Adapter	SQL	@pckhoi	✅	PostgreSQL is supported by pgx
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
ArangoDB Adapter	NoSQL	@adamwasila	✅	For ArangoDB

Adapter	Type	Author	AutoSave	Description
Amazon S3 Adapter	Cloud	Soluto	❌	For Minio and Amazon S3
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 (go-redis)	KV store	@mlsen	✅	For Redis
Etcd Adapter	KV store	@sebastianliu	❌	For etcd
BoltDB Adapter	KV store	@speza	✅	For Bolt

Adapter	Type	Author	AutoSave	Description
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 Adapter	JDBC	Casbin	✅	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server are supported by JDBC
Hibernate Adapter	ORM	Casbin	✅	Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL are supported by Hibernate
MyBatis Adapter	ORM	Casbin	✅	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server (the same as JDBC) are supported by MyBatis 3

Adapter	Type	Author	AutoSave	Description
Hutool Adapter	ORM	@mapleafgo	✅	MySQL, Oracle, PostgreSQL, SQLite are supported by Hutool
MongoDB Adapter	NoSQL	Casbin	✅	MongoDB is supported by mongodb-driver-sync
DynamoDB Adapter	NoSQL	Casbin	❌	For Amazon DynamoDB
Redis Adapter	KV store	Casbin	✅	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Filtered File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files with policy subset loading support
String Adapter (built-in)	String	@calebfaruki	❌	For String
Basic Adapter	Native ORM	Casbin	✅	pg, mysql, mysql2, sqlite3, oracledb, mssql are supported by the adapter itself
Sequelize Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by Sequelize
TypeORM	ORM	Casbin	✅	MySQL, PostgreSQL,

Adapter	Type	Author	AutoSave	Description
Adapter				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	@sarneeh and 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 Adapter	SQL	@touchifyapp	✅	PostgreSQL adapter with advanced policy subset loading support and improved performances built with node-postgres.
Mongoose Adapter	NoSQL	elastic.io and Casbin	✅	MongoDB is supported by Mongoose
Mongoose Adapter	NoSQL	minhducck	✅	MongoDB is supported by Mongoose

Adapter	Type	Author	AutoSave	Description
(No-Transaction)				
Node MongoDB Native Adapter	NoSQL	@juicycleff	✅	For Node MongoDB Native
DynamoDB Adapter	NoSQL	@fospitia	✅	For Amazon DynamoDB
Couchbase Adapter	NoSQL	@MarkMYoung	✅	For Couchbase
Redis Adapter	KV store	Casbin	❌	For Redis
Redis Adapter	KV store	@NandaKishorJeripothula	❌	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	ORM	Casbin	✅	Powerful PHP database abstraction layer (DBAL) with many features for database schema introspection and management.

Adapter	Type	Author	AutoSave	Description
(Recommend)				
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

Adapter	Type	Author	AutoSave	Description
				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, Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
Async Databases Adapter	ORM	sampingantech	✅	PostgreSQL, MySQL, SQLite,

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

Adapter	Type	Author	AutoSave	Description
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 `remove_filtered_policy()`)	For Couchbase
DynamoDB Adapter	NoSQL	@abqadeer	✅	For DynamoDB
Pymongo Adapter	NoSQL	Casbin	❌	MongoDB is supported by Pymongo
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

Adapter	Type	Author	AutoSave	Description
				supported by Entity Framework 6
EFCore Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, DB2, etc. are supported by Entity Framework Core
EFCore Adapter (.NET Core 5)	ORM	@g4dvali	✅	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 Cosmos DB Adapter	Cloud	@sagarkhandelwal	✅	For Microsoft Azure Cosmos DB

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

Adapter	Type	Author	AutoSave	Description
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
DynamodDB Adapter	NoSQL	@fospitia	✅	For Amazon DynamoDB
MongoDB Adapter	MongoDB	@wangjun861205	✅	For MongoDB
JSON Adapter	String	Casbin	✅	For JSON
YAML Adapter	String	Casbin	✅	For YAML

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	File	Casbin	❌	For .CSV (Comma-Separated Values) files

Adapter	Type	Author	AutoSave	Description
(built-in)				
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 (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) 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

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

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

```php
use Casbin\Enforcer;

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

```rust
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

```go
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)
```

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

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

```rust
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

```go
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)
```

```rust
// 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");
```

```php
// 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
    'driver'     => 'pdo_mysql',
    'host' => '127.0.0.1',
    'dbname' => 'test',
    'user' => 'root',
    'password' => '',
    'port' => '3306',
];

$a = DatabaseAdapter::newAdapter($config);
$e = new Enforcer('examples/basic_model.conf', $a);
```

Use your own storage adapter

You can use your own adapter like below:

```go
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.LoadPolicy()
```

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 (
    "github.com/casbin/casbin"
    "github.com/casbin/xorm-adapter"
    _ "github.com/go-sql-driver/mysql"
)

// By default, the AutoSave option is enabled for an enforcer.
a := xormadapter.NewAdapter("mysql",
"mysql_username:mysql_password@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basic_model.conf", a)

// Disable the AutoSave option.
e.EnableAutoSave(false)

// Because AutoSave is disabled, the policy change only affects the policy in
Casbin enforcer,
// it doesn't affect the policy in the storage.
e.AddPolicy(...)
e.RemovePolicy(...)

// Enable the AutoSave option.
e.EnableAutoSave(true)

// Because AutoSave is enabled, the policy change not only affects the policy
in Casbin enforcer,
// but also affects the policy in the storage.
e.AddPolicy(...)
```

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

Watchers

We support to use distributed messaging systems like etcd to keep consistence between multiple Casbin enforcer instances. So our users can concurrently use multiple Casbin enforcers to handle large number of permission checking requests.

Similar to policy storage adapters, we don't put 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 as below. Any 3rd-party contribution on a new watcher is welcomed, please inform us and I will put it in this list:)

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

Watcher	Type	Author	Description
ZooKeeper Watcher	Messaging system	Grepsr	Watcher for Apache ZooKeeper
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
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
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	Matteo	Watcher for PostgreSQL

Watcher	Type	Author	Description
		Collina	

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
PostgreSQL Watcher	Database	Casbin	Watcher for PostgreSQL

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

API	Description
	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()
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

Dispatcher provide a way to synchronize incremental changes of policy. It should be based on consistency algorithms such as raft to ensure the consistency of all enforcer instances. Through dispatcher users can easily establish distributed clusters.

The dispatcher's method is divided into two parts. The first 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, config changes etc.

> NOTE
>
> we hope dispatcher just ensure the consistency of Casbin enforcer at runtime. So if the policy is inconsistent when initialization, the dispatcher will not work properly. Users need to ensure that the state of all instances is consistent before using dispatcher.

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

Go

Dispatcher	Type	Author	Description
Hashicorp Raft Dispatcher	raft	Casbin	Dispatcher based on hashicorp/raft
KDKYG/casbin-dispatcher	raft	@KDKYG	Dispatcher based on hashicorp/raft

DistributedEnforcer

DistributedEnforcer wraps SyncedEnforcer for 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 the 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 light-weight, we don't put role manager code in the main library (except the default role manager). A complete list of Casbin role managers is provided as below. Any 3rd-party contribution on a new role manager is welcomed, please inform us and I will put it in this list:)

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in Casbin policy
Session Role Manager	EDOMO Systems	Supports role hierarchy stored in 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. 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 Casbin policy

For developers: all role managers must implement the RoleManager interface. 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 Casbin policy
Session Role Manager	Casbin	Supports role hierarchy stored in Casbin policy, with time-range-based sessions

For developers: all role managers must implement the RoleManager interface. 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 Casbin policy

For developers: all role managers must implement the RoleManager interface. 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 Casbin policy

For developers: all role managers must implement the RoleManager interface. Default Role Manager can be used as a reference implementation.

API

See API section for details.

Middlewares

Web frameworks

Go | Java | Node.js | PHP | Python | .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-boilerplate or gofiber-casbin

Name	Description
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: authz
DotWeb	Simple and easy go web micro framework, via plugin: authz
Tango	Micro & pluggable web framework for Go, via plugin: authz

Name	Description
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 jcasbin-springboot-plugin 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
Vert.x	A tool-kit for building reactive applications on the JVM, via plugin: vertx-auth-jcasbin
JFinal	A simple, light, rapid, independent and extensible Java WEB + ORM framework, via plugin: jfinal-authz
Nutz	Web framework (MVC/IOC/AOP/DAO/JSON) for all Java developers, via plugin: nutz-authz

Name	Description
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 shanbe-api 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
Egg	Born to build better enterprise frameworks and apps with Node.js & Koa, via plugin: egg-authz or egg-zrole

Name	Description
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

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
ThinkPHP 6.0	The ThinkPHP 6.0 framework, via plugin: think-authz
Symfony	The Symfony PHP framework, via plugin: symfony-permission or symfony-casbin

Name	Description
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-authorization
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

Name	Description
	APIs with Python 3.6+ based on standard Python type hints, via plugin: fastapi-authz or Fastapi-app
OpenStack	The most widely deployed open source cloud software in the world, via plugin: openstack-patron

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 or rocket-casbin-demo

Name	Description
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-

Name	Description
	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 another word, Casbin should be placed between 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 as below. Any 3rd-party contribution on a new GraphQL middleware is welcomed, please inform us and we will put it in 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 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 friendly API for RBAC with domains. This API is a subset of Management API. The RBAC users could use this API to simplify the code

RoleManager API

RoleManager provides interface to define the operations for managing roles. Adding matching function to rolemanager allows using wildcards in role name and domain

Data Permissions

Solutions for data permissions

API Overview

This overview only shows you how to use Casbin APIs and doesn't explain how Casbin is installed and 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 at the Enforce APIs of Casbin. We will load a RBAC model from `model.conf`, and load policies from `policy.csv`. You can learn the Model syntax here, and we won't talk about 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]
```

`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. Function `casbin.NewEnforcer()` will return an enforcer. It will recognize its 2 parameters as file paths, and load the files from there. Errors occurred in the process are stored in `err`. This code used the default adapter to load model and policies. And of course you can get the same result by using a third-party adapter.

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

EnforceEx API

Sometimes you may wonder which policy allowed the request, so we 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]
```

function `EnforceEx()` will return the exact policy string in the return value `reason`. In this example, `amber` is a role of `admin`, so policy `p, admin, data1, read` made this request `true`. The out put of this code is in the comment.

Casbin prepared a lot of APIs like this. Those APIs added some extra functions on the basic one. They are:

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

  With a matcher.

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

  A combination of `EnforceWithMatcher()` and `EnforceEx()`.

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

  Do a list job, returns an array.

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

Management API

Get API

These APIs are used to get exact objects in policies. This time we loaded an enforcer like the last example and get something from it.

Please read 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)
```

Same as the last example, the first 4 lines loaded some necessary information from local files. We won't talk about that here anymore.

Code `allSubjects := enforcer.GetAllSubjects()` got all the subjects in the policy file and returned them as an array. Then we printed that array.

Normally, 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 show up in the current named policy.

Similarly, we prepared `GetAll` functions for `Objects, Actions, Roles`. The only thing you need to do is to change the word `Subject` in the function name to what you want if you want to access those functions.

Besides, we have more getters for policies. The call method and return value are similar to those above.

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

Add, Delete, Update API

Casbin prepared a lot of APIs for policies. These APIs allow you to add, delete or edit policies dynamically at runtime.

This code shows you how to add, remove and update your policies, and told you

how to confirm that 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, then use HasPolicy() to confirm that
enforcer.AddPolicy("added_user", "data1", "read")
hasPolicy := enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // true, we added that policy
successfully

// remove a policy, then use HasPolicy() to confirm that
enforcer.RemovePolicy("alice", "data1", "read")
hasPolicy = enforcer.HasPolicy("alice", "data1", "read")
fmt.Println(hasPolicy) // false, we deleted that policy
successfully

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

Using these four kinds of APIs can edit your policies. Like these, we prepared same kinds of APIs to `FilteredPolicy, NamedPolicy, FilteredNamedPolicy, GroupingPolicy, NamedGroupingPolicy, FilteredGroupingPolicy, FilteredNamedGroupingPolicy`. To use them, you only need to replace word `Policy` in the function name to the words above.

Besides, if you change parameters to arrays, you can batch edit you policies.

For example, to functions like this:

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

if we change `Policy` to `Policies`, and edit the parameter to:

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

then we can batch edit these policies.

Same operations also useful to `GroupingPolicy, NamedGroupingPolicy`.

AddEx API

Casbin provides AddEx series 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)
(bool, error)
SelfAddPoliciesEx(sec string, ptype string, rules [][]string)
(bool, error)
```

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

For example: Compare `AddPolicies` and `AddPoliciesEx`

You can copy the code below into the test under casbin to run and observe.

```
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
    ok, _ = e.AddPoliciesEx([][]string{{"user1", "data1",
"read"}, {"user2", "data2", "read"}})
    fmt.Println(e.GetPolicy())
    // {"user2", "data2", "read"} is added successfully
    // because AddPoliciesEx automatically filters the existing
{"user1", "data1", "read"}
```

RBAC API

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

Here we only show you how to use RBAC APIs of Casbin and won't talk about RBAC itself. You can get more details here.

We use this code to load 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, use a instance of 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()` returned an array that contained all roles contained amber. In this example, amber has only one role admin, so array `roles` is `[admin]`. And similarly, you can use `GetUsersForRole()` to get users belongs to the 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 have a lot of APIs like this. Their call methods and return values have the same style as the above APIs. You can find these APIs in 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

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

```node
const e = await newEnforcer('examples/rbac_model.conf', 'examples/
rbac_policy.csv')
```

```php
$e = new Enforcer('examples/rbac_model.conf', 'examples/
rbac_policy.csv');
```

```python
e = casbin.Enforcer("examples/rbac_model.conf", "examples/
rbac_policy.csv")
```

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

```rust
let mut e = Enforce::new("examples/rbac_model.conf", "examples/
rbac_policy.csv").await?;
```

```java
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

```go
ok, err := e.Enforce(request)
```

```node
const ok = await e.enforce(request);
```

```php
$ok = $e->enforcer($request);
```

```python
ok = e.enforcer(request)
```

```java
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

```go
ok, err := e.EnforceWithMatcher(matcher, request)
```

```php
$ok = $e->enforceWithMatcher($matcher, $request);
```

```python
ok = e.enforce_with_matcher(matcher, request)
```

```java
boolean ok = e.enforceWithMatcher(matcher, request);
```

`EnforceEx()`

EnforceEx explain enforcement by informing matched rules.

For example:

Go Node.js PHP Python

```go
ok, reason, err := e.EnforceEx(request)
```

```javascript
const ok = await e.enforceEx(request);
```

```php
list($ok, $reason) = $e->enforceEx($request);
```

```python
ok, reason = e.enforce_ex(request)
```

`EnforceExWithMatcher()`

EnforceExWithMatcher use a custom matcher and explain enforcement by informing matched rules.

For example:

Go

```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:

Go Node.js PHP Python .NET Rust Java

```
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:

Go Node.js PHP Python .NET Rust Java

```
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)
```

```php
$removed = $e->removePolicy("alice", "data1", "read");
```

```python
removed = e.remove_policy("alice", "data1", "read")
```

```dotnet
var removed = e.RemovePolicy("alice", "data1", "read");
or
var removed = await e.RemovePolicyAsync("alice", "data1", "read");
```

```rust
let removed = e.remove_policy(vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;
```

```java
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

```go
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

```go
removed := e.RemoveFilteredPolicy(0, "alice", "data1", "read")
```

```nodejs
const p = ['alice', 'data1', 'read']
const removed = await e.removeFilteredPolicy(0, ...p)
```

```php
$removed = $e->removeFilteredPolicy(0, "alice", "data1", "read");
```

```python
removed = e.remove_filtered_policy(0, "alice", "data1", "read")
```

```dotnet
var removed = e.RemoveFilteredPolicy("alice", "data1", "read");
or
var removed = await e.RemoveFilteredPolicyAsync("alice", "data1",
"read");
```

```rust
let removed = e.remove_filtered_policy(0, vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;
```

```java
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.RemoveFilteredNamedPolicyAync("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')
```

```php
$has = $e->hasNamedGroupingPolicy("g", "alice", "data2_admin");
```

```python
has = e.has_named_grouping_policy("g", "alice", "data2_admin")
```

```dotnet
var has = e.HasNamedGroupingPolicy("g", "alice", "data2_admin");
```

```rust
let has = e.has_named_grouping_policy("g", vec!["alice".to_owned(),
"data2_admin".to_owned()]);
```

```java
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

```go
added := e.AddGroupingPolicy("group1", "data2_admin")
```

```nodejs
const added = await e.addGroupingPolicy('group1', 'data2_admin')
```

```php
$added = $e->addGroupingPolicy("group1", "data2_admin");
```

```python
added = e.add_grouping_policy("group1", "data2_admin")
```

```dotnet
var added = e.AddGroupingPolicy("group1", "data2_admin");
or
```

```rust
let added = e.add_grouping_policy(vec!["group1".to_owned(),
"data2_admin".to_owned()]).await?;
```

```java
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

```go
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesAdded := e.AddGroupingPolicies(rules)
```

```javascript
const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
  ];

const areRulesAdded = await e.addGroupingPolicies(groupingRules);
```

```python
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:

Go Node.js PHP Python .NET Rust Java

```
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:

Go Node.js Python Rust Java

```
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");
```

```rust
let removed = e.remove_grouping_policy(vec!["alice".to_owned(),
"data2_admin".to_owned()]).await?;
```

```java
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

```go
rules := [][] string {
                []string {"ham", "data4_admin"},
                []string {"jack", "data5_admin"},
        }

areRulesRemoved := e.RemoveGroupingPolicies(rules)
```

```javascript
const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
  ];

const areRulesRemoved = await e.removeGroupingPolicies(groupingRules);
```

```rust
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");
```

```rust
let removed = e.remove_filtered_grouping_policy(0,
vec!["alice".to_owned()]).await?;
```

```java
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

```go
removed := e.RemoveNamedGroupingPolicy("g", "alice")
```

```javascript
const removed = await e.removeNamedGroupingPolicy('g', 'alice')
```

```php
$removed = $e->removeNamedGroupingPolicy("g", "alice");
```

```python
removed = e.remove_named_grouping_policy("g", "alice", "data2_admin")
```

```dotnet
var removed = e.RemoveNamedGroupingPolicy("g", "alice");
or
var removed = await e.RemoveNamedGroupingPolicyAsync("g", "alice");
```

```rust
let removed = e.remove_named_grouping_policy("g",
vec!["alice".to_owned()]).await?;
```

```java
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

```go
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesRemoved := e.RemoveNamedGroupingPolicies("g", rules)
```

```node.js
const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
  ];

const areRulesRemoved = await e.removeNamedGroupingPolicies('g',
groupingRules);
```

```python
rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]
are_rules_removed = e.remove_named_grouping_policies("g", rules)
```

```rust
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?;
```

```java
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

```go
removed := e.RemoveFilteredNamedGroupingPolicy("g", 0, "alice")
```

```javascript
const removed = await e.removeFilteredNamedGroupingPolicy('g', 0,
'alice')
```

```php
$removed = $e->removeFilteredNamedGroupingPolicy("g", 0, "alice");
```

```python
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)
```

`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
```

```
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" && $key2 ==
"/alice_data/:resource") {
        return true;
    } elseif ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data2/:id/using/:resId") {
        return true;
    } else {
        return false;
    }
}

func customFunctionWrapper(...$args){
    $key1 := $args[0];
    $key2 := $args[1];

    return customFunction($key1, $key2);
}

$e->addFunction("keyMatchCustom", customFunctionWrapper);
```

```
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"
```

```
fn custom_function(key1: STring, 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";
    }
}

FunctionTest.CustomFunc customFunc = new FunctionTest.CustomFunc();
e.addFunction(customFunc.getName(), customFunc);
```

`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")
```

`UpdateGroupingPolicy()`

UpdateGroupingPolicy updates oldRule to newRulein `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 newRulein `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 suport 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

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

```node
const e = await newEnforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv')
```

```php
$e = new Enforcer('examples/rbac_model.conf', 'examples/rbac_policy.csv');
```

```python
e = casbin.Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

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

```rust
let mut e = Enforcer::new("examples/rbac_model.conf", "examples/
```

```
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

```
res := e.GetRolesForUser("alice")
```

```
const res = await e.getRolesForUser('alice')
```

```
$res = $e->getRolesForUser("alice");
```

```
roles = e.get_roles_for_user("alice")
```

```
var res = e.GetRolesForUser("alice");
```

```
let roles = e.get_roles_for_user("alice", None); // No domain
```

```
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

```go
res := e.GetUsersForRole("data1_admin")
```

```javascript
const res = await e.getUsersForRole('data1_admin')
```

```php
$res = $e->getUsersForRole("data1_admin");
```

```python
users = e.get_users_for_role("data1_admin")
```

```dotnet
var res = e.GetUsersForRole("data1_admin");
```

```rust
let users = e.get_users_for_role("data1_admin", None); // No domain
```

```java
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

```go
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",
```

```rust
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

```go
e.DeletePermissionForUser("bob", "read")
```

```javascript
await e.deletePermissionForUser("bob", "read")
```

```php
$e->deletePermissionForUser("bob", "read");
```

```python
e.delete_permission_for_user("bob", "read")
```

```dotnet
var deleted = e.DeletePermissionForUser("bob", "read");
or
var deleted = await e.DeletePermissionForUserAsync("bob",
"read");
```

```rust
let deleted = e.delete_permission_for_user("bob",
vec!["read".to_owned()]).await?;
```

```java
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

```go
e.DeletePermissionsForUser("bob")
```

```javascript
await e.deletePermissionsForUser('bob')
```

```php
$e->deletePermissionsForUser("bob");
```

```python
e.delete_permissions_for_user("bob")
```

```dotnet
var deletedAtLeastOne = e.DeletePermissionsForUser("bob");
or
var deletedAtLeastOne = await
e.DeletePermissionsForUserAsync("bob");
```

```rust
let deleted_at_least_one =
e.delete_permissions_for_user("bob").await?;
```

```java
boolean deletedAtLeastOne = e.deletePermissionForUser("bob");
```

GetPermissionsForUser()

GetPermissionsForUser gets permissions for a user or role.

For example:

Go | Node.js | PHP | Python | .NET | Java

```go
e.GetPermissionsForUser("bob")
```

```javascript
await e.getPermissionsForUser('bob')
```

```php
$e->getPermissionsForUser("bob");
```

```python
e.get_permissions_for_user("bob")
```

```dotnet
var permissions = e.GetPermissionsForUser("bob");
```

```java
List<List<String>> permissions = e.getPermissionsForUser("bob");
```

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");
```

`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 GetImplicitUesrsForRole("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.GetImplicitPermissionsForUser("alice");
```

```
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
```

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")
```

`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.")
```

RBAC with Domains API

A more friendly API for RBAC with domains. 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

```go
e, err := NewEnforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```node
const e = await newEnforcer('examples/
rbac_with_domains_model.conf', 'examples/
rbac_with_domains_policy.csv')
```

```php
$e = new Enforcer('examples/rbac_with_domains_model.conf',
'examples/rbac_with_domains_policy.csv');
```

```python
e = casbin.Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```dotnet
var e = new Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv");
```

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

```java
Enforcer e = new Enforcer("examples/
rbac_with_domains_model.conf", "examples/
rbac_with_domains_policy.csv");
```

GetUsersForRoleInDomain()

GetUsersForRoleInDomain gets the users that has a role inside a domain.

For example:

Go Node.js Python

```go
res := e.GetUsersForRoleInDomain("admin", "domain1")
```

```javascript
const res = e.getUsersForRoleInDomain("admin", "domain1")
```

```python
res = e.get_users_for_role_in_domain("admin", "domain1")
```

GetRolesForUserInDomain()

GetRolesForUserInDomain gets the roles that a user has inside 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()`

GetPermissionsForUserInDomain gets permissions for a user or role inside a domain.

For example:

Go Java

```
res := e.GetPermissionsForUserInDomain("alice", "domain1")
```

```
List<List<String>> res =
e.getPermissionsForUserInDomain("alice", "domain1");
```

`AddRoleForUserInDomain()`

AddRoleForUserInDomain adds a role for a user inside a domain. Returns false if

the user already has the role (aka not affected).

For example:

Go Python Java

```go
ok, err := e.AddRoleForUserInDomain("alice", "admin", "domain1")
```

```python
ok = e.add_role_for_user_in_domain("alice", "admin", "domain1")
```

```java
boolean ok = e.addRoleForUserInDomain("alice", "admin",
"domain1");
```

`DeleteRoleForUserInDomain()`

DeleteRoleForUserInDomain deletes a role for a user inside a domain. Returns false if the user does not have the role (aka not affected).

For example:

Go Java

```go
ok, err := e.DeleteRoleForUserInDomain("alice", "admin",
"domain1")
```

```java
boolean ok = e.deleteRoleForUserInDomain("alice", "admin",
"domain1");
```

`DeleteRolesForUserInDomain()`

DeleteRolesForUserInDomain deletes all roles for a user inside a domain. Returns false if the user does not have any roles (aka not affected).

For example:

Go

```
ok, err := e.DeleteRolesForUserInDomain("alice", "domain1")
```

`GetAllUsersByDomain()`

GetAllUsersByDomain would get all users associated with the domain. Returns empty string array if has no domain defined in model.

For example:

Go

```
res := e.GetAllUsersByDomain("domain1")
```

`DeleteAllUsersByDomain()`

DeleteAllUsersByDomain would delete all users associated with the domain. Returns false if has no domain defined in 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 reversed keyword, just like `for`, `if` in a programming language, we should never put `::` in a domain.

RoleManager API

RoleManager

RoleManager provides interface to define the operations for managing roles. Adding matching function to rolemanager allows using wildcards in role name and domain.

`AddNamedMatchingFunc()`

AddNamedMatchingFunc add MatchingFunc by ptype to RoleManager. MatchingFunc will work when operating 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()`

AddNamedDomainMatchingFunc add MatchingFunc by ptype to RoleManager. `DomainMatchingFunc` is similar to `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()`

GetRoleManager 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()`

GetNamedRoleManager 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()`

SetRoleManager 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()`

SetNamedRoleManager sets the role manager by named ptype.

For example:

Go Python

```
rm := e.SetNamedRoleManager("g2", rm)
```

```
rm = e.set_role_manager("g2", rm)
```

`Clear()`

Clear clears all stored data and resets the role manager to the 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)
```

Data Permissions

We have two solutions for data permissions (filtering). Using implicit assignment APIs. Or just use BatchEnforce() API.

1. Query implicit roles or permissions

When a user inherits a role or permission via RBAC hierarchy instead of directly assigning them in a policy rule, we call such type of assignment as `implicit`. To query such implicit relations, you need to use these 2 APIs: `GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser` instead of `GetRolesForUser()` and `GetPermissionsForUser`. For more details, please see this GitHub issue.

2. Use `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);
```

Advanced usage

Multi-threading

Using Casbin in a multi-threading manner

Benchmarks

The overhead of policy enforcement of 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

If you use Casbin in a multi-threading manner, you can use 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++).

It also supports the `AutoLoad` feature, which means the Casbin enforcer will automatically load the latest policy rules from DB if it has changed. Call `StartAutoLoadPolicy()` to start automatically loading policy periodically and call `StopAutoLoadPolicy()` to stop it.

Benchmarks

Go C++ Lua (JIT)

The overhead of policy enforcement is benchmarked in model_b_test.go. The testbed is:

```
Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core(s),
8 Logical Processor(s)
```

The benchmarking result of `go test -bench=. -benchmem` is as follows (op = an `Enforce()` call, ms = millisecond, KB = kilo bytes):

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

Test case	Rule size	Time overhead (ms/op)	Memory overhead (KB)
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 of Casbin CPP is benchmarked in `tests/benchmarks` directory with the help of Google's benchmarking tool. The testbed for these benchmarks is:

```
Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 4 cores, 4 threads
```

Here is the benchmarking result of executing `casbin_benchmark` target built in `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
Priority	9 rules (2 users, 2 roles)	0.035

Lua Casbin's overhead of policy enforcement is benchmarked in bench.lua. The

testbed was a Ubuntu VM with the CPU:

```
AMD Ryzen(TM) 5 4600H CPU @ 3.0GHz, 6 Cores, 12 Threads
```

The benchmarking result of `luajit bench.lua` is as follows (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

Test case	Rule size	Time overhead (ms/op)
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

Performance Optimization

When applied in a production environment with millions of users or permissions, you may encounter performance downgrade in Casbin enforcement, there are usually two causes:

High Volume Traffic

The number of coming requests per second is too large, e.g., 10,000 request/s for a single Casbin instance. In such case, a single Casbin instance is usually not enough to handle all the requests. There are 2 possible solutions:

1. Use multi-threading to enable multiple Casbin instances, so you can fully utilize all the cores in the machine. See details at: Multi-threading.
2. Deploy Casbin instances to a cluster (multiple machines). Use Watcher to guarantee all Casbin instances are consistent. See details at: Watchers.

> NOTE
>
> You can use the above methods both at the same time, e.g., deploy Casbin to a 10-machine cluster. Each machine has 5 threads simultaneously to serve Casbin enforcement requests.

High Number of Policy Rules

Millions of policy rules may be required in a cloud or multi-tenant environment. Each enforcement call or even loading the policy rules at the initial time is very slow. Such cases can usually be mitigated in several ways:

1. Your Casbin model or policy is not well-designed. A well-written model and policy will abstract out the duplicated logic for each user/tenant and reduce the number of rules to a very small level (< 100): e.g., you can share some default rules across all tenants and let users customize their rules later. Customized rules can override the default rules. If you still have question, please send GitHub issue to the Casbin repos.

2. Do sharding to let a Casbin enforcer only load a small set of policy rules, e.g., enforcer_0 only serves for tenant_0 to tenant_99, enforcer_1 only serves for tenant_100 to tenant_199. To load only a subset of all policy rules, see details at: 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 uses O(1) time to query the RBAC tree for role-user relationship and do enforcement. If your g rules don't change often, then the RBAC tree won't need to update. See details at this dicussion: https://github.com/casbin/casbin/issues/681#issuecomment-763801583

> NOTE
>
> You can try the above methods all at the same time.

Authorization of Kubernetes

K8s-authz is a Kubernetes (k8s) RBAC & ABAC authorization middleware based on Casbin. This middleware uses K8s validation admission webhook to check the policies defined by casbin, for every request of the k8s resources. These custom admission controllers perform some kind of validation on the request object that was forwarded by api server and based on a logic, sends back a response to api server that contains information on whether to allow or reject the request. These controllers are registered with Kubernetes using the `ValidatingAdmissionWebhook`.

The K8s API server needs to know when to send the incoming request to our admission controller. For this part, we have defined a validation webhook which would proxy the requests for any type of K8s resource/sub-resource and perform policy verification on it. The user would be allowed to perform the operations on these resources, only if the casbin enforcer authorizes it. The enforcer checks the roles of the user defined in the policies. This middleware would be deployed on the K8s cluster.

Requirements

Before proceeding, make sure to have the following-

- A running k8s Cluster. You can either run the clusters through Docker by enabling it on the Docker Desktop or you can setup the complete K8s ecosytem locally or on your server. You can follow this detailed guide to setup

the k8s cluster locally on Windows or this guide if want to setup for Linux.
- Kubectl CLI This is the guide to setup it on Windows and this guide for Linux.
- OpenSSL

Usage

- Generate the certificates and keys for every user by using openssl and running the following script:-

```
./gen_cert.sh
```

- Build the docker image from the Dockerfile manually by running the following command and then change the build version here and at the deployment file, as per the builds.

```
docker build -t casbin/k8s_authz:0.1 .
```

- Define the casbin policies in the model.conf and policy.csv. You can refer the docs to get to know more about the working of these policies.

- Before deploying, you can change the ports in main.go and also in the validation webhook configuration file depending on your usage.

- Deploy the validation controller and the webhook on k8s cluster by running:-

```
kubectl apply -f deployment.yaml
```

- For a production server, we need to create a k8s `secret` to place the certificates for security purposes.

```
kubectl create secret generic casbin -n default \
  --from-file=key.pem=certs/casbin-key.pem \
  --from-file=cert.pem=certs/casbin-crt.pem
```

- Once, this part is done we need to change the directory of the certs in main.go and then in manifests with that of the `secret`.

Now the server should be running and ready to validate the requests for the operations on the k8s resources.

Kubernetes + Casbin Plugin: K8s-Gatekeeper

1. Overview & Documents for Casbin K8s-Gatekeeper

Casbin K8s-GateKeeper is an Kubernetes admission webhook which 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 writting any piece of code but several lines of declerative configurations of Casbin models and policies, which are part of Casbin ACL(Access Control List) language.

Casbin K8s-GateKeeper is developed and maintained by Casbin community. Repository of this project is here. https://github.com/casbin/k8s-gatekeeper

0.1 A simple example

For example, you dont need to write any code, but using 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. Writting several lines of ACL is far better than writting lots of codes.
- It allows hot update of configurations. You don't need to shut down the whole plugin to modify configurations.
- It is fledxible. Arbitrary rules can be made on any k8s resource which can be explored with kubectl gatekeeper
- It screened the impelementation of k8s admission webhook, which is very complicated. You don't need to really know what K8s admission webhook is, or how to write code for it. What 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 Casbin Community. Feel free to contact us if anything about this plugin confuses you, or if you encounter any problem when trying this.

1.1 How Casbin K8s-gatekeeper works?

K8s-gatekeeper is an admission webhook for k8s, using Casbin to apply arbitrary user-defined access control rules to help prevent any operation on k8s which 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 detail 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 webhoook: '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 doc

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:
```

This request will go through the process of all the middleware shown in the picture, including our K8s-gatekeeper. K8s-gatekeeper can detected all the Casbin enforcers stored in K8s's etcd, which is created and maintained by user(via kubectl or 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 is Casbin enforcer, model or policy, see this document see: Get Started)

For example, for some reason, the administrator want to forbid the apperance of 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 and how to write them in 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 containg model and policy above, the previous admission request will be reject by this enforcer, which means K8s won't create this deployment.

2 Install K8s-gatekeeper

Three methods are provided for installing K8s-gatekeeper: External webhook, Internal webhook and helm.

> NOTE
>
> Note: these methods are only for user to try K8s-gatekeeper, and it is not secure. If you want to use it in productive environment, please make sure you read Chapter 5. Advanced setting and make modifications

> accordingly when necessary before installation

2.1 Internal webhook

2.1.1 Step 1: Build image

Internal webhook means the webhook itself will be implmented as a service inside k8s. Creating a service as well as deployment requires a image of K8s-gatekeeper. You can should build your own image.

Run

```
docker build --target webhook -t k8s-gatekeeper .
```

Then there will be 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 following commands

```
kubectl apply -f config/rbac.yaml
kubectl apply -f config/webhook_deployment.yaml
kubectl apply -f config/webhook_internal.yaml
```

Soon K8s-gatekeeper should be running, and you can use `kubectl get pods` to confirm that.

2.1.3 Step3: Install Crd Resources for K8s-gatekeeper

Run following commands

```
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
```

2.2 External webhook

External webhook means K8s-gatekeeper will be running outside the K8s, and K8s will visit K8s-gatekeeper like visiting a ordinary website. K8s has mandatory requirement that admission webhook must be HTTPS. For the sake of user's experience in trying K8s-gatekeeper, we have provided you a set of certificate as well as private

key (though it is not secure). If you prefer to use your own certificate, please refer to Chapter 5. Advanced setting to make adjustments to the certificate and private key.

The certificate we provide is issued for 'webhook.domain.local', so please modify the host (like /etc/hosts), point webhook.domain.local to the ip address on which K8s-gatekeeper is running.

Then execute

```
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 image

See Chapter 2.1.1

2.3.2 helm install

Run `helm install k8sgatekeeper ./k8sgatekeeper`

3. Try K8s-gatekeeper

3.1 Create Casbin Model and Policy

You have 2 methods to create a model and policy: via kubectl or via go-client we provide.

3.1.1 Create/Update Casbin Model and Policy via kubectl

In K8s-gatekeeper, Casbin model is stored in a kind of CRD Resource called 'CasbinModel'. Its definition is located in `config/auth.casbin.org_casbinmodels.yaml`

There are examples in `example/allowed_repo/model.yaml`. You are supposed to pay attention to the following fields:

- metadata.name: name of the model. This name MUST be same with the name of 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 CasbinPolicy object related to this model) will be ignored.
- spec.modelText: a string which contains the model text of a casbin model.

Casbin Policy is stored in another kind of 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`. You are supposed to pay attention to the following fields:

- metadata.name: name of the policy. This name MUST be same with the name of CasbinModel object related to this policy, so that K8s-gatekeeper can pair them and create an enforcer.
- spec.policyItem: a string which contains the policy text of a casbin model.

After creating your own CasbinModel and CasbinPolicy files, use

```
kubectl apply -f <filename>
```

to put them into effect.

Once a pair of CasbinModel and CasbinPolicy is created, within at most 5 seconds K8s-gatekeeper will be able to detect it.

3.1.2 Create /Updata Casbin Model and Policy via go-client we provide

It has been taken into consideration that there may be situation that it is not convenient to use shell to execute command directly on a node of K8s cluster, for example, when you are building a automatic cloud platform for your corporation. Therefore we have developed a go-client to create 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)
```

parameter externalClient means whether K8s-gatekeeper is running inside the K8s cluster or not.

In model.go we provide various functions to create/delete/modify CasbinModel. You can find out how to use there interfaces in model_test.go.

In policy.go we provide various functions to create/delete/modify CasbiPolicy. You can find out how to use there interfaces in policy_test.go.

3.1.2 Try whether K8s-gatekeeper works

Suppose you have already created exactly the model and policy in example/allowed_repo, now try this

```
kubectl apply -f example/allowed_repo/testcase/reject_1.yaml
```

you are supposed to find that k8s will reject this request, and mentioning that this webhook was the reason why this request is rejected. However, when you tries to apply example/allowed_repo/testcase/approve_2.yaml, it will be accepted.

4. How to write Model and Policy K8s-gatekeeper

First of all, you are supposed to know the basic grammar of Casbin Models and Policies. If you haven't acknowledged it, please read Get Started first. In this chapter we will assume that you have known what are Casbin Models and Policies.

4.1 Request Definition of Model

When K8s-gatekeeper is authorizing a request, the input is always one object: the go object of the Admission Request. Which means the enforcer will always be used like this

```
ok, err := enforcer.Enforce(admission)
```

in which admission is an `AdmissionReview` object defined by K8s's official go api `"k8s.io/api/admission/v1"`. You can see the definition of this struct is this repository https://github.com/kubernetes/api/blob/master/admission/v1/types.go. Or see https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-request-and-response for more information

Therefore for any model used by K8s-gatekeeper, the definitiion of request_definition should always be like this

```
[request_definition]
r =  obj
```

Name 'obj' is not mandatory, as long as the name is consistent with the name used in `[matchers]` part.

4.2 Matchers of Model

You are supposed to use the ABAC feature of Casbin to write down your rule. However, the expression evaluator integrated in Casbin supports neither indexing in maps or arrays(slices), nor the expansion of array. Therefore K8s-gatekeeper provide various 'Casbin functions' as extension to impelement these features. If you still find that your demand cannot be fulfilled by these extensions, it is welcomed to start a issue, or pr directly.

If you don't know what is casbin funtion, see Function for more information.

Here are the extension functions

4.2.1 Externsion functions

4.2.1.1 access

Access is used to solve the problem that Casbin doesn't support indexing in map or array. `example/allowed_repo/model.yaml` is the example 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 equal to `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Image`, in which `r.obj.Request.Object.Object.Spec.Template.Spec.Containers` is obviously a slice.

Access is also able to call simple funtion which has not parameters and one single return value. `example/container_resource_limit/model.yaml` is an example.

```
[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 equal 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 funtion which has not parameters and one single return value.

4.2.1.2 accessWithWildcard

Sometimes it is natural to have demand like this: all elements in an array must have prefix "aaa". However, Casbin doesn't support `for` loop. However with `accessWithWildcard` and the "map/slice expansion" feature, such demand can be easily implemented.

For example, suppose `a.b.c` is an array `[aaa,bbb,ccc,ddd,eee]`, then result of `accessWithWildcard(a,"b","c","*")` will be a slice `[aaa,bbb,ccc,ddd,eee]`. See? with wildcard `*` this slice is expanded.

Similarly, wildcard can be used more than once. For example, 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 Argument

In the expression evaluator of Casbin, when a parameter is an array, it will be automatically expanded as the variable-length argument. Utilizing this feature to support the array/slice/map expansion, we also integrated serveral functions accepting an array/slice as parameter.

- contain(), accept multiple parameters, and returns whether there is an parameter other than the last parameter equals the last parameter
- split(a,b,c...,sep,index) it returns a slice which contains `[splits(a,sep)[index], splits(b,sep)[index], splits(a,sep)[index]...]`
- len() return the length of the variable-length argument
- matchRegex(a,b,c...regex) return whether a,b,c... all of them matches 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)
```

Assume `accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , "*", "Image")` returns `["a:b", "c:d", "e:f", "g:h"]` then because splits supports variable-length argument, and splits operation is applied on each element, and eventualy element whose index is 1 will be selected and return, so `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(): parse an integer to a float. (It is because that any number in comparsion must be converted into float).
- ToString(): convert an object to 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(): return whether the parameter is nil

5. Advanced Settings

5.1 About Certificates

In k8s, it is mandatory that a webhook should use HTTPS. There are two approaches to achieve that:

- Use self-signed certificates(examples in this repo use this method)
- Use a normal certificate

5.1.1 Self-signed certificates

Using a self-signed certificate means that the CA issuing the certificate is not one of the well-known CAs, therefore you must let k8s know this CA.

Current the example in this repo uses a self-made CA, whose private key and certificate is stored in `config/certificate/ca.crt` and `config/certificate/ca.key`. Certificate for the webhook is `config/certificate/server.crt`, issued by the self-made CA. The domains of this certificate is "webhook.domain.local"(for external webhook) and "casbin-webhook-svc.default.svc"(for internal webhook)

Information about CA is passed to k8s via webhook configuration files. Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", whose content is base64 encoded string of the certificate of the CA.

In case that you need to change the certificate/domain (for example, maybe you want to put this webhook into another namespace of k8s while using internal webhook; or maybe you want to change a domain while using external webhook), the following procedures should be taken:

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 webhook server

```
openssl genrsa -des3 -out server.key 2048
openssl rsa -in server.key  -out server.key
```

3. Use the self-generate CA to sign the certificate for webhook

Copy your system's openssl config file for temporary use. You can use `openssl version -a` to find out the location of the config file, ususally called `openssl.cnf`.

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 following lines to the file:

```
[alt_names]
DNS.2=<The domain you want>
```

The 'casbin-webhook-svc.default.svc' should be replaced 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 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
```

3. Replace the 'CABundle' field

Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", whose content is base64 encoded string of the certificate of the CA.

4. If you are using helm, similar changes need to be applied to helm charts.

5.1.2 Legal certificates

If you uses legal certificates, you just don't need all these procedures. Remove "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 of Envoy which performs external RBAC & ABAC authorization through casbin. This middleware uses Envoy's external authorization API through a gRPC server. This proxy would be deployed on any type of envoy-based service meshes like Istio.

Requirements

- Envoy 1.17+
- Istio or any type of service mesh
- grpc dependencies

Dependencies are managed through `go.mod`.

Working of Middleware

- A client would make a http request.
- Envoy proxy would send that request to grpc server.
- The grpc server would then authorize the request based on casbin policies.
- If authorized, the request would be sent through or else, it gets denied.

The grpc server is based on protocol buffer from external_auth.proto from Envoy.

```
// A generic interface for performing authorization check on
incoming
// requests to a networked service.
service Authorization {
  // Performs authorization check based on the attributes
associated with the
  // incoming request, and returns status `OK` or not `OK`.
  rpc Check(v2.CheckRequest) returns (v2.CheckResponse);
}
```

From the above proto file, we have to use `Check()` service in the authorization server.

Usage

- Define the Casbin policies under config files by following this guide.

You can verify/test your policies on online casbin-editor.

- Start the authorizing server by running:-

```
$ go build .
$ ./authz
```

- Load the envoy configuration:-

```
$  envoy -c authz.yaml -l info
```

Once the envoy starts, it will start intercepting requests for the authorization process.

Integrating to Istio

You need to send custom headers, which would contain usernames in the JWT token OF headers for this middleware to work. You can check the official Istio docs to get more info on modifying `Request Headers`.

Management

Admin Portal

Admin portal for Casbin

Casbin Service

Using Casbin as a service

Log & Error Handling

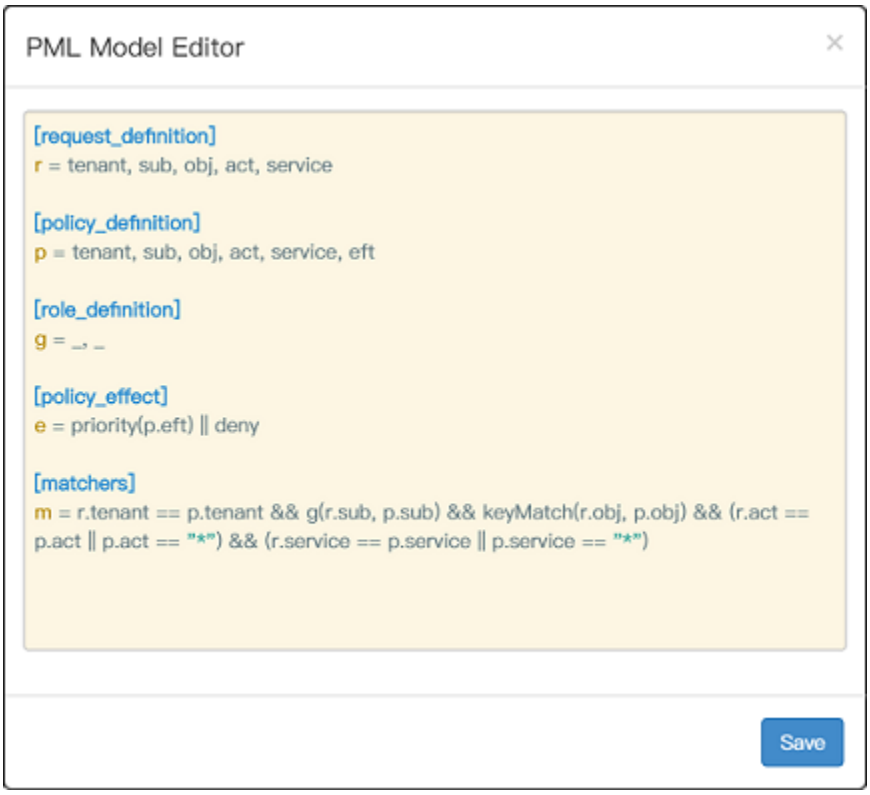
Casbin log & error handling

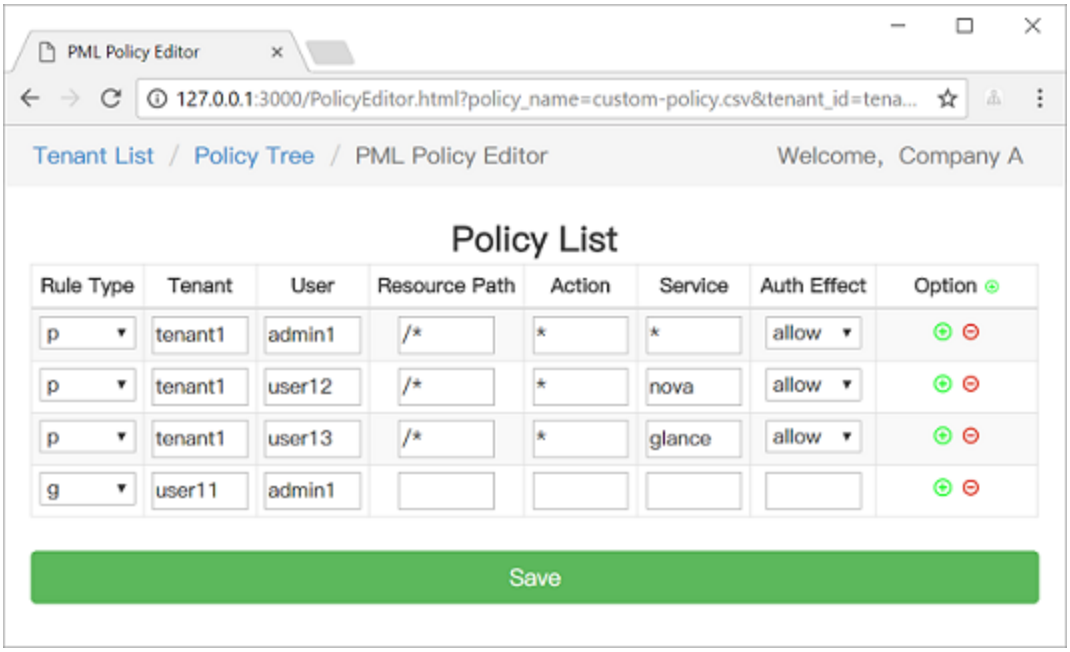
Frontend Usage

Casbin.js is a Casbin addon that facilites your access-control management in the frontend application

Admin Portal

We provide a web-based portal called Casdoor for model management and policy management:





There are also 3rd-party admin portal projects that use Casbin as authorization engine. You can get started to build 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-	@piexlmax	Vue +	Gin	Based on Gin +

Project	Author	Frontend	Backend	Description
admin		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-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 ThinkPHP
video.tinywan.com	@Tinywanner	LayUI	ThinkPHP	RESTful API gateway based on ThinkPHP5 +

Project	Author	Frontend	Backend	Description
				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.
Buttress	The Access Control as a Service solution based on Casbin.
auth-server	Auth Server for proofreading services.

Log & Error Handling

Logging

Casbin uses the built-in `log` to print logs to console by default like:

```
2017/07/15 19:43:56 [Request: alice, data1, read ---> true]
```

The logging is not enabled by default. You can toggle it via `Enforcer.EnableLog()` or the last parameter of `NewEnforcer()`.

> NOTE
>
> We already support logging the model, enforce request, role, policy in Golang. You can define your own log for logging Casbin. If you are using Python, pycasbin leverages the default Python logging mechanism. The pycasbin package 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 intitilized within the parent application you will not see any log messages from pycasbin.

Use different logger for different enforcer

Every enforcer could have its own logger to log info, and it could be changed at run-time.

And you could use a proper logger via the last paramter of `NewEnforcer()`, if you using this way to initialize your enforcer, you needn't use the enabled parameter,

cause the priority of the enabled field in logger is higher.

```
// Set a default logger as enforcer e1's logger.
// This operation could also be seen as changing the logger of
e1 at run-time.
e1.SetLogger(&Log.DefaultLogger{})

// Set another logger as enforcer e2's logger.
e2.SetLogger(&YouOwnLogger)

// Set your logger when initialize 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
Defatule logger (built-in)	Casbin	The default logger using golang log.
Zap logger	Casbin	Using zap, provide json encoded log and you could 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 print the message.
IsEnabled()	mandatory	Show the current logger's enabled status.
LogModel()	mandatory	Log info related to model.
LogEnforce()	mandatory	Log info related to enforce.
LogRole()	mandatory	Log info related to role.
LogPolicy()	mandatory	Log info related to policy.

You can pass your custom `logger` to `Enforcer.SetLogger()`.

Here is an example about how to customize a logger for Golang:

```
import (
    "fmt"
    "log"
    "strings"
```

Error handling

Error or panic may happen when you use Casbin for reasons like:

1. Invalid syntax in model file (.conf).
2. Invalid syntax in policy file (.csv).
3. Custom error from storage adapters, e.g., MySQL fails to connect.
4. Casbin's bug.

There are five main functions you may need to care about for error or panic:

Function	Behavior on error
NewEnforcer()	Return error
LoadModel()	Return error
LoadPolicy()	Return error
SavePolicy()	Return error
Enforce()	Return error

> NOTE
>
> `NewEnforcer()` calls `LoadModel()` and `LoadPolicy()` inside. So you don't have to call the latter two calls when using `NewEnforcer()`.

Enable & 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 policy is 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 run-time.
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 facilites 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

Middleware	Type	Author	Description
vue-authz	Vue	Casbin	Vue wrapper for Casbin.js
angular-authz	Angular	Casbin	Angular wrapper for Casbin.js

Quick Start

You can use `manual` mode in your frontend application, and set the permission 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 got an authorizer `authorizer`. We can get permission rules from it by using the API `authorizer.can()` and `authorizer.cannot()`. 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) => {
    if (success) {
        console.log("you can write data1");
    } else {
        console.log("you cannot write data1");
    }
});
// output: you can write data1
```

and `cannot()` 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, variable `success` in parameters means the request get the result without throwing an error, and doesn't mean that the permission rule is `true`. `failed` 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

Advanced Usage

Casbin.js provides a perfect solution to integrating your frontend access-control

management with your backend Casbin service.

Use `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 `CasbinJsGetUserPermission` 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 fronend.
    c.ServeJSON()
}
```

NOTE

Currently, `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** object 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 the difference between Node-Casbin and Casbin.js. In a word, Node-Casbin is the core of Casbin implemented in NodeJS environment, and it's normally used as an access-controlling management toolkit at the server ends.

Casbin.js is an frontend library that help you use Casbin to authorize your webpage user 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 turn 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 sides could bring secure risks.
3. Difficult for the seperation between client and server as well as the agile development.

We wish a tool that eases the process of using Casbin at the frontend. Actually, the core of Casbin.js is the manipulation of 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 behaviors of the user at the frontend side.

Casbin.js avoid the two problems that mentioned above: Casbin service will no longer be pulled up repeatedly, and the size of passing messages between the client and the server are reduced. We also avoid to store all the policies at the frontend. User can only accessible to his own permission, but have no idea about anything about things like 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 web browser

IDE Plugins

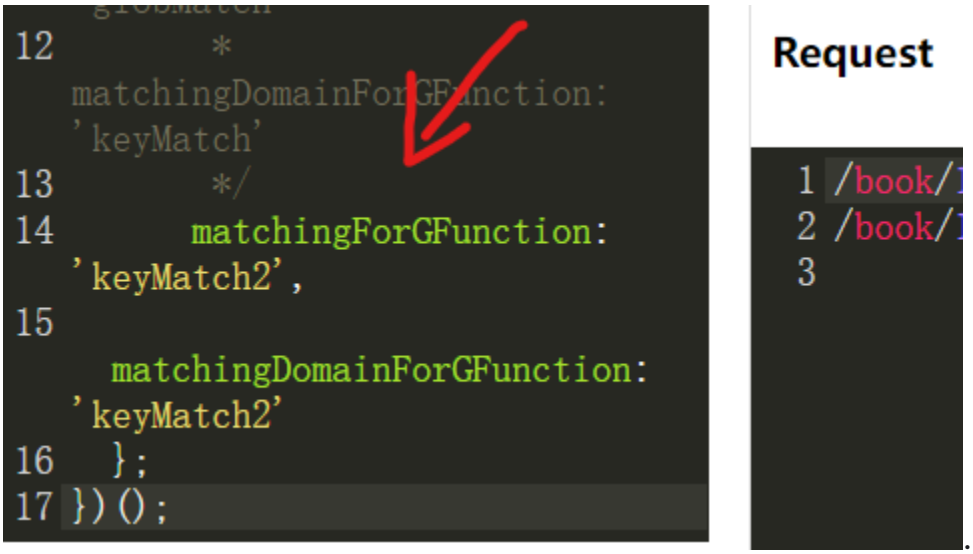
IDE plugins for Casbin

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.

Use Pattern

If you use `RBAC with pattern` or `RBAC with all pattern`, it specifies the pattern matching function in the lower left corner.



If you want to write the equivalent code, you need to specify the pattern matching function through the relevant api. See RBAC with Pattern

> NOTE
>
> The editor is based on node-casbin. Due to the synchronization delay

> between different language of casbin, the authentication result of the `editor` may be different from the authentication result of the casbin you are using. If so, please submit issues to the casbin repository you are using.

IDE Plugins

We have plugins for these IDEs:

JetBrains

- Download: https://plugins.jetbrains.com/plugin/14809-casbin
- Source code: https://github.com/will7200/casbin-idea-plugin

VSCode (WIP)

- Source code: https://github.com/casbin/casbin-vscode-plugin

More

Our Adopters

Casbin' 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
Zenpress	A CMS system written in Golang.	.conf	Gorm
Argo CD	GitOps continuous delivery for Kubernetes.	.conf	.csv
Muxi Cloud	PaaS of Muxi Cloud, an easier way to manage Kubernetes cluster.	.conf	Code

Name	Description	Model	Policy
EngineerCMS	A CMS to manage knowledge for engineers.	.conf	SQLite
Cyber Auth API	A Golang authentication API project.	.conf	.csv
IRIS Community	Website for IRIS Community Activities.	.conf	.csv
Metadata DB	BB archive metadata database.	.conf	.csv
Qilin API	ProtocolONE's licenses managemen 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 controller.	README	N/A

Name	Description	Model	Policy
Notadd	A micro-service development architecture based on Nest.js.	.conf	DB adapter

Name	Description	Model	Policy
dtrace	EduScaled's tracing system.	Commit	N/A

Integration via plugin

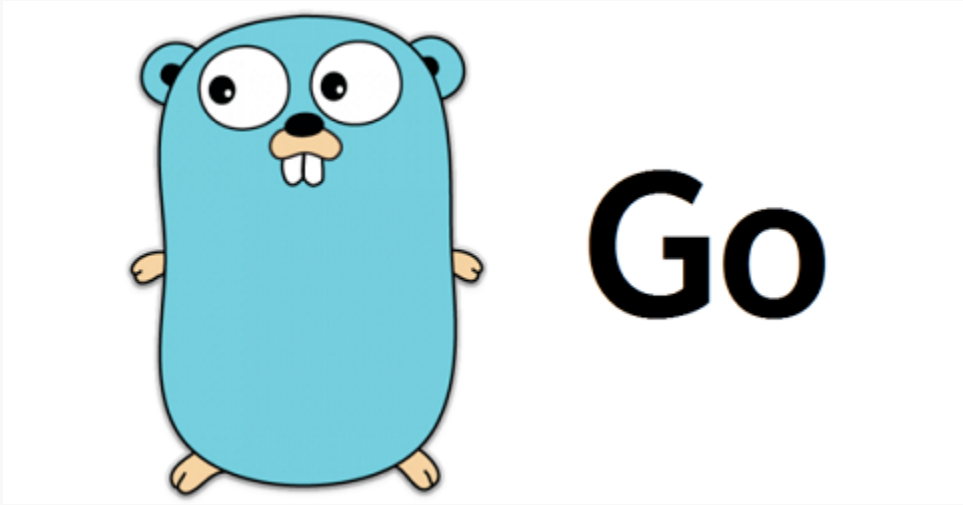
Name	Description	Plugin	Model	Policy
Docker	The world's leading software container platform	casbin-authz-plugin (recommended by Docker)	.conf	.csv
Gobis	Orange's lightweight API Gateway written in go	casbin	Code	Request

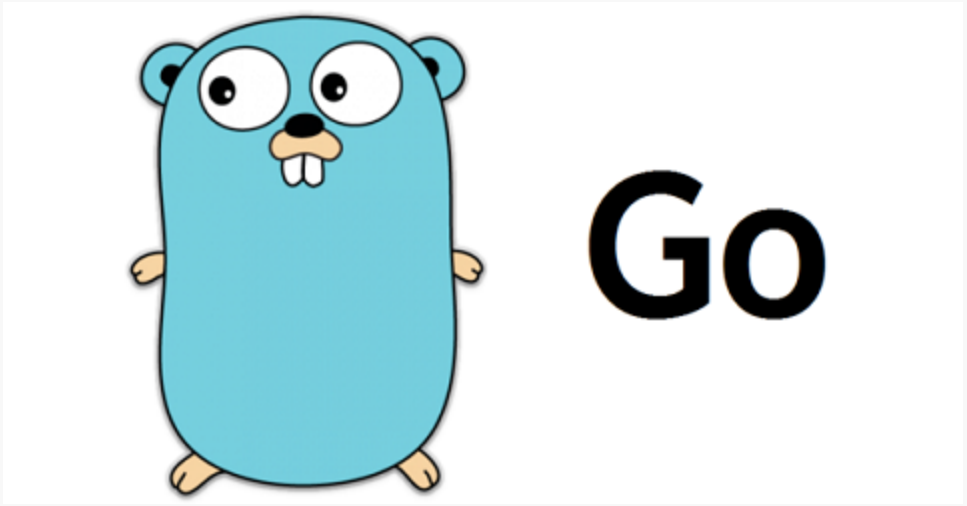
Contributing

Casbin is a powerful authorization library supports access control models with many languages implementations, so long as you are good at one language, you can participate in the development of Casbin. New contributors are always welcomed.

Currently, there are mainly two types of projects.

- **Algorithms oriented**——First kind of projects are algorithms related projects for different language implementations, include most of the mainstream programming languages ranging from Golang, Java, C++ to Elixir, Dart and Rust, and their peripheral products.

	
Casbin	jCasbin



production-ready

production-ready

PyCasbin

production-ready

Casbin.NET

production-ready

- **Application oriented**——Second kind of projects are application related projects.

Project	Demo	Details	Skill stacks
Casdoor	Casdoor	Casdoor is UI-first centralized authentication / Single-Sign-On (SSO) platform based on OAuth 2.0 / OIDC	JavaScript + React and Golang + Beego + SQL
Casnode	Casbin Forum	Casnode is next generation forum software	JavaScript + React and Golang + Beego + SQL
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 Casbin model and policy editor	TypeScript + React

Get 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 issues to promote development of Casbin, no matter bugs or proposal. Before file an issue on

GitHub, it would be better to discuss first on Discord or GitHub Discussions

> Note: When reporting an issue, please use English to describe the details of you problem.

- **Help with docs!** Contributing start from docs are a good choice to start your contribution.

- **Help solve issues!** We prepare a table containing easy tasks suitable for beginner, with different level of challenges labeled with different tags, check the table here.

Pull Requests

Casbin uses GitHub as its developing platform. So the pull requests are the main source of contributions.

There are something you need to know before you open a pull request:

- Explain why you send this PR and what this PR would do to the repo.

- Make sure the PR does only one thing, otherwise please split it.

- If there are newly added files, please include Casbin license to the top of 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.
```

- In Casdoor, Casnode and Casbin OA, your might need to setup demo to show the maintainer your pull request help the development of project.

- When you open PR and commit your contribution, it would be better to use the semantic commits, format: `<type>(<scope>): <subject>`, `<scope>` is optional. For more detailed usage, please see conventional commit

License

By contributing to Casbin, you agree that your contributions will be licensed under its 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, and 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're 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. What data we store, we'll protect within commercially acceptable means to prevent loss and theft, as well as unauthorized access, disclosure, copying, use or modification.

We don't 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 around 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 29 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 aren’t refundable.

If you have an issue with your account, or think there’s been an error in billing, please contact the support for more help.