Authorization (RBAC)

Cascading RBAC: global, organization, team, and resource voters.

Anonymous · 09/09/2026

Authorization (RBAC)

The bundle provides a **four-level cascading permission system**:

```
Resource-level → Team-level → Organization-level → Global system-level
```

Each level falls back to the next. A global super-admin passes every check without being explicitly named in each organization or resource.

Roles and permissions

Roles are identified by a unique `code` string. There is no `scope` column — the context is defined by the **relation** that uses the role:

Relation	Context	Example codes
`UserRole`	Global system	`system.admin`, `system.auditor`
`OrganizationMembership`	Organization	`org.owner`, `org.admin`, `org.member`
`TeamAccess` (app)	Team within org	`team.lead`, `team.contributor`
`ResourceAccess` (app)	Resource	`project.admin`, `project.read`

Permissions are atomic capabilities attached to roles via the `RolePermission` pivot:

```
Role "org.admin" ──► Permission "org.invite"
                 ──► Permission "org.billing"
                 ──► Permission "org.settings"
```

Using voters in controllers

The bundle registers three voters automatically:

Global (no subject)

```
#[IsGranted('system.manage')]
public function adminDashboard(): Response { ... }
```

Checks: `user → UserRole → Role → RolePermission → Permission.code`.

Organization-scoped (subject = Organization)

```
#[IsGranted('org.invite', subject: 'organization')]
public function inviteMember(Organization $organization): Response { ... }
```

Checks: `user → OrganizationMembership(org) → Role → Permissions`. Falls back to global.

Resource-scoped (subject = ResourceAccessible entity)

```
#[IsGranted('write', subject: 'project')]
public function editProject(Project $project): Response { ... }
```

Checks: `user → ResourceAccess(resource) → Role → Permissions`. Falls back to team-level (if resource implements `TeamAccessibleInterface`), then org-level (if resource implements `OrganizationOwnedInterface`), then global.

Programmatic checks

```
use Derafu\PlatformBundle\Identity\Contract\Service\PermissionCheckerInterface;

class SomeService
{
    public function __construct(
        private readonly PermissionCheckerInterface $checker
    ) {}

    public function doSomething(UserInterface $user, Organization $org): void
    {
        // Global
        if ($this->checker->hasPermission($user, 'system.manage')) { ... }

        // Organization-scoped
        if ($this->checker->hasOrganizationPermission($user, $org, 'org.invite')) {
... }

        // Resource-scoped
        if ($this->checker->hasResourcePermission($user, $project, 'write')) { ... }
    }
}
```

How the cascade works

`hasResourcePermission($user, $resource, 'write')`:

1. If `$resource` implements `ResourceAccessibleInterface`: iterate the resource's direct access entries for this user → check each role's permissions for `'write'`.
2. If not found and `$resource` implements `TeamAccessibleInterface`: iterate team access entries → for each team, check if the user is a member → if yes, check the team's role permissions for `'write'`.
3. If not found and `$resource` implements `OrganizationOwnedInterface`: call `hasOrganizationPermission($user, $resource->getOrganization(), 'write')`.
4. If not found: call `hasPermission($user, 'write')` (global).

Any level returning `true` short-circuits the rest.

API Platform integration

The bundle's voters work with API Platform's `security` attribute out of the box — no extra configuration. API Platform calls Symfony's `AuthorizationCheckerInterface`, which triggers the voters:

```
use ApiPlatform\Metadata\ApiResource;
use ApiPlatform\Metadata\Get;
use ApiPlatform\Metadata\GetCollection;
use ApiPlatform\Metadata\Post;

#[ApiResource(
    operations: [
        new GetCollection(security: "is_granted('ROLE_USER')"),
        new Get(security: "is_granted('read', object)"),
        new Post(security: "is_granted('write', object)"),
    ],
)]
class Project { ... }
```

The `Get` and `Post` operations pass `object` as the subject, so the `ResourcePermissionVoter` fires and resolves the full cascade (direct → teams → org → global). The `GetCollection` operation uses a global check (no subject), so the `PermissionVoter` fires.

For automatic **listing filters** (which rows the user sees), see Multi-tenancy & RLS.

Vote reasons (debugging)

All three voters add human-readable reasons to Symfony's `Vote` object when they deny. Check the Web Profiler's Security panel to see why a specific check failed:

```
User "alice@example.com" does not hold permission "org.invite"
in organization #42.
```

Last updated on 09/09/2026