SCALED ACCESS

WHITEPAPER

The Developer's Guide to Relationship-based Access Control

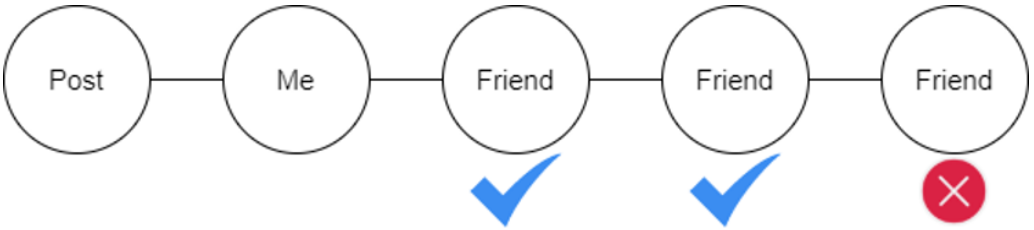
By Peter Morlion

If you've never heard of ReBAC (relationship-based access control), that's fine. It's not too difficult and we'll walk you through it. Chances are, you're already using this model in your current applications! Allow us to tell you why ReBAC is such an interesting model for access control and how you can start implementing it.

What is ReBAC?

Relationship-based access control is a model where access decisions are based on the relationships a subject has. When the subject (often a user, but possibly also a device or application) wants to access a resource, our system will either allow or deny this access based on the specific relationships the subject has.

Probably the most well-known examples of relationship-based access control are social networks. In Facebook, for example, you can allow access to your posts and photos to friends of friends. We can easily draw this in a diagram:



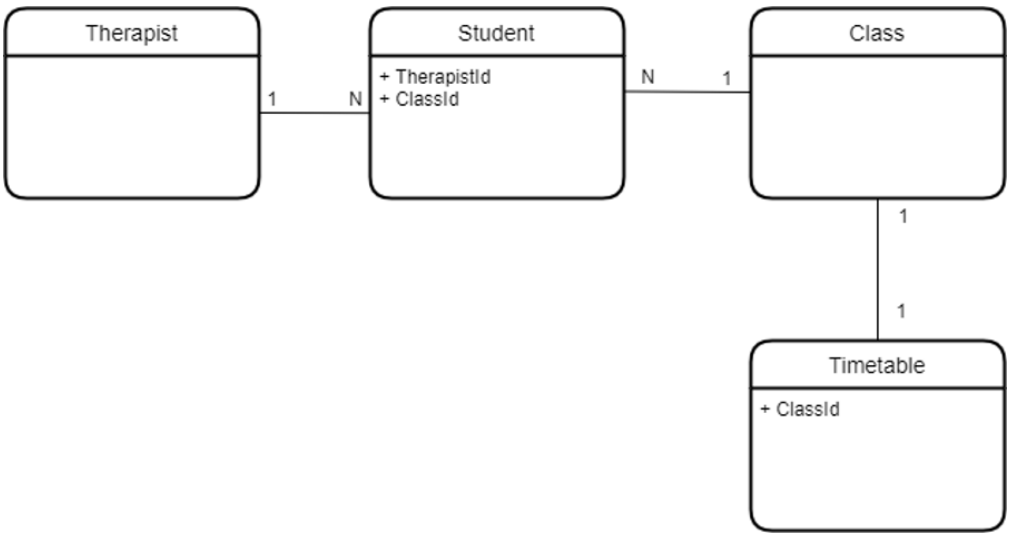
My friends can see my posts. Friends of my friends can see my posts. But friends of those friends cannot because they have the wrong relationship path to my post (one too many steps).

So, in ReBAC, we don't allow access because someone has a certain role (e.g., a user in the "Human Resources" group). We allow access because they have certain relationships with other entities in our system.

ReBAC is often explained in academic literature in reference to social networks because, by definition, they contain a network of relationships. But ReBAC isn't applicable to social networks alone. In fact, you probably already have a network of relationships in your database.

ReBAC Already in Your System

Many applications already work with a database that contains a network of entities that have relationships with each other. We can clearly see this in relational databases. For example, take a look at this database schema:



The therapist has a “is_therapist_of” relationship with a student, who has a “is_member_of” relationship with the class, which in turn has a relationship with the timetable. In classic authorization models, we would give a therapist access to the timetable by adding them to a certain group. That is, we could give them the “timetable_viewer” role.

This access may not be granular enough though. What if we only want to give the therapist access to the class timetable of a student for which they are the therapist? If we give them the “timetable_viewer” role, they could have access to all timetables.

We could easily run a query to see if the therapist has the correct relationship. In SQL, this could look something like this:

```
SELECT t.Id, tt.Id
FROM Therapist t
INNER JOIN Student s ON s.TherapistId = t.Id
INNER JOIN Class c ON s.ClassId = c.Id
INNER JOIN Timetable tt ON tt.ClassId = c.Id
WHERE t.Id = @TherapistId
```

This would give you a list of timetables that the given therapist has access to, and the reason they have access is because they are the therapist of certain students in classes for those timetables. This is what we call a policy: a way of expressing who has access to content in your application.

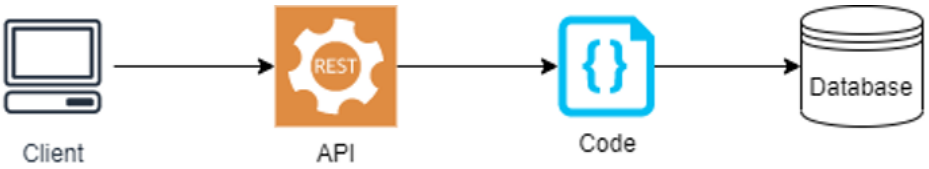
Your code may already contain several such policies, but this doesn’t scale easily. For every change to the policies, the developers need to make changes in the code. You could build a system where you can easily make changes, even with a

good UI. But you wouldn't write your own database, so why would you invest time and money into an authorization system when you should be focused on your business logic and on the areas where your company or organization makes a difference?

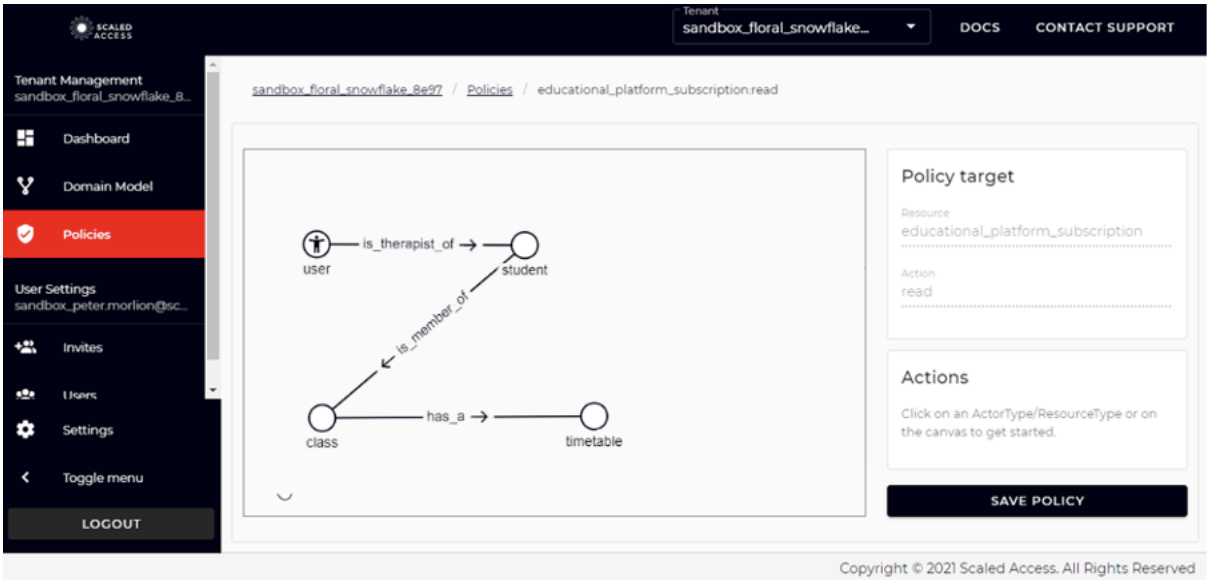
Another downside of this approach, and why it doesn't scale, is that we're tightly coupling our policies to our service implementations. So, let's look at how we can put our policies in a separate context, using Scaled Access.

ReBAC with Scaled Access

This is a simplified version of what our application architecture often looks like:

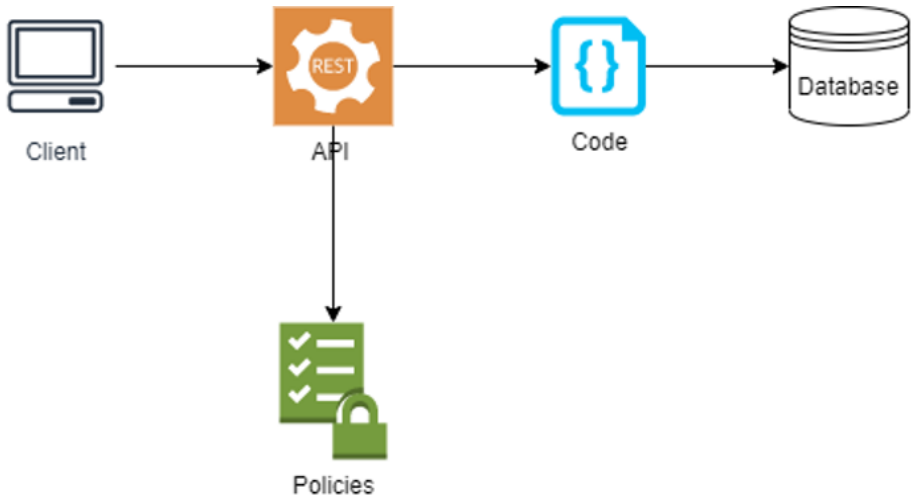


Authorization decisions are now located in our code, based on what's in our database. With Scaled Access, we can design our policies in the user interface, based on the network we've uploaded to the platform.



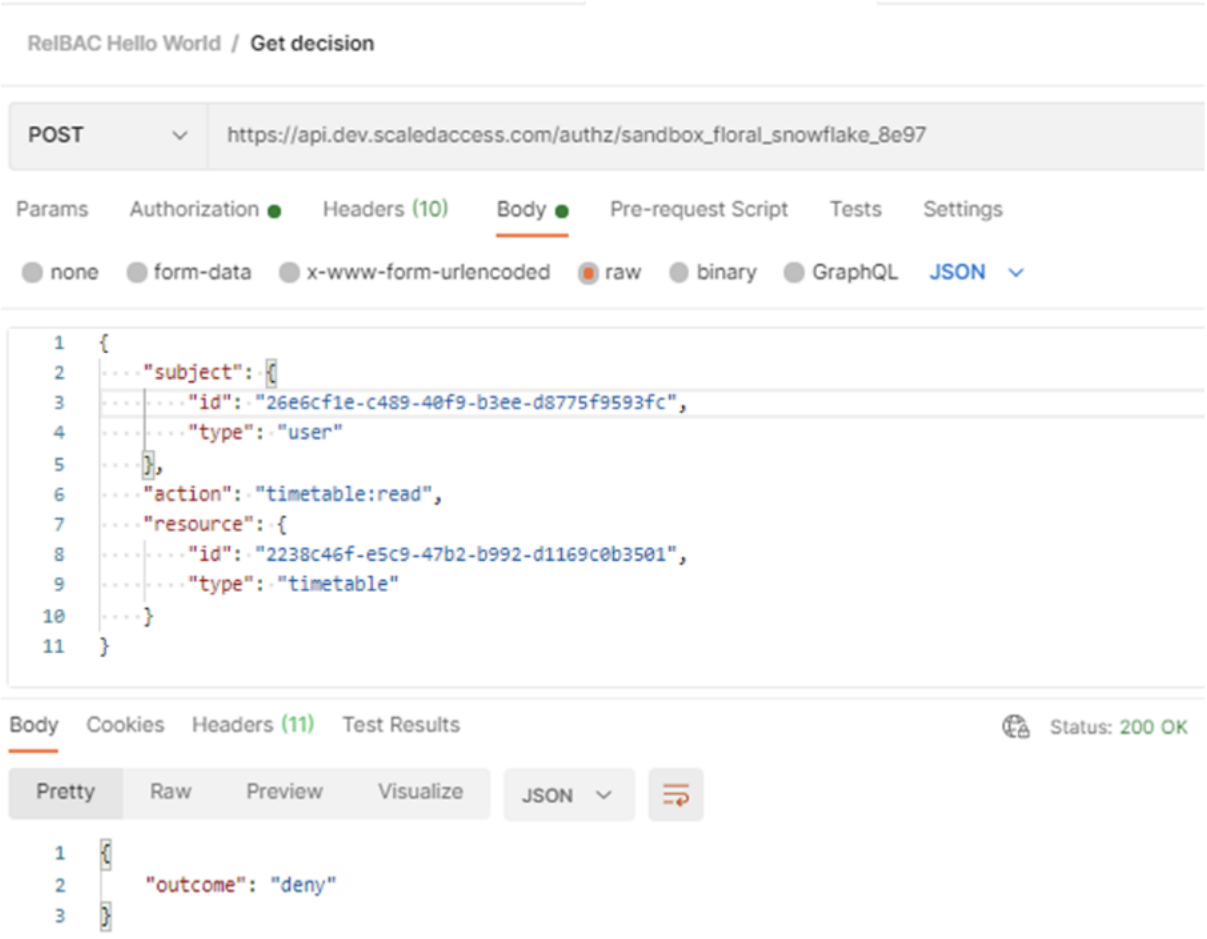
This doesn't imply that we need to replace our existing database with the Scaled Access platform, but the platform does need a way of knowing what the network of entities and relationships looks like. Once we have that data, designing policies is easy.

Now that we have our data and our policies, all we need to do is call Scaled Access when a user needs access to a resource:



How you do this depends on how your application is hosted. In AWS, you can use Custom Authorizers and Azure offers a solution with API Management Policies. What you're looking for, based on your requirements, is a way of adding custom logic to the authorization model of your cloud provider's API solution. If the platform you run your applications and APIs on doesn't offer an obvious way of customizing authorization, don't hesitate to contact us. We can help you.

Conceptually, all you need to add is an HTTP call to our authorization endpoint:



The request above shows we're checking if the user has read access to a specific timetable. The outcome "deny" shows us that they don't.

But Wait! There's More

With Scaled Access, you can easily add relationship-based access control to your applications and APIs. It offers a simple way of designing and applying policies, but we didn't stop there.

Invitations

Scaled Access has a built-in system of invitation, so users can invite others to accept a relationship in the application. In our example, we could have a teacher invite a new therapist to accept an "is_therapist_of" relationship with a student. If the therapist accepts, they will automatically have access to the timetable. You don't need to implement this invitation flow yourself. You just need to call the necessary endpoint to send the invite, and we'll do the rest.

In fact, we dogfood our system to manage our tenants and customers. When a new customer is added, we send them an invitation through the same flow that you use.

Token Enrichment

If you're already using an external Identity Provider like Auth0 or Okta, we provide integration to enrich the access token that your users receive. After things are set up, you'll be able to add a user's relationship to their token. This is what the payload of such a decoded token could look like:

```
{
  "https://scaledaccess.com/relationships": [
    {
      "relationshipType": "is_therapist_of",
      "to": {
        "id": "2484519d-2e4a-4e2b-89e7-f5e7d6955634",
        "type": "student",
        "name": "Elizabeth"
      }
    }
  ],
  "iss": "https://petermorlion-relbac-helloworld.eu.auth0.com/",
  "sub": "auth0|605318e5a9b06e006a7e5092",
  "aud": "https://helloworld.ropc.example.com",
  "iat": 1616667136,
  "exp": 1616753536,
  "azp": "t2OGcArqg6EGvdPW9XkZ6BLDbbCdqv6c",
  "gty": "password"
}
```

The direct relationships are already present in the token, allowing you to make simple authorization decisions without any extra calls.

Interested in token enrichment, but not working with Auth0 or Okta? Let us know, and we'll help you set things up.

Custom Policies

If you're thinking, "great, but my policies won't easily transfer to the designer in your UI," we've got you covered. Behind the scenes, we're using the Rego language. What you design in the UI is translated to Rego, a language to define access policies. Then, when you need a decision from the authorization endpoint, we evaluate this Rego policy with the necessary data as input (e.g., a user and their relationships).

But, if the designer doesn't fit your needs or taste, you can skip it and just upload your custom Rego policy. Check out the docs for more on that.

Scaled Access Makes ReBAC Easy

Relationship-based access control is more than just a model for social networks. Your database likely contains a network of entities and their relationships already. With ReBAC, you can model authorization policies based on these relationships. ReBAC offers more than role-based access control because it allows policies based on multi-step relationships between entities and decisions for specific entities, not entity types (e.g., you have access to this specific timetable, not all timetables).

Additionally, Scaled Access offers you a platform to manage and configure your relationship-based access control, which gives you more time to focus on what's important for your business.