Improving Your Security with Service to Service and Token Exchange Patterns

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

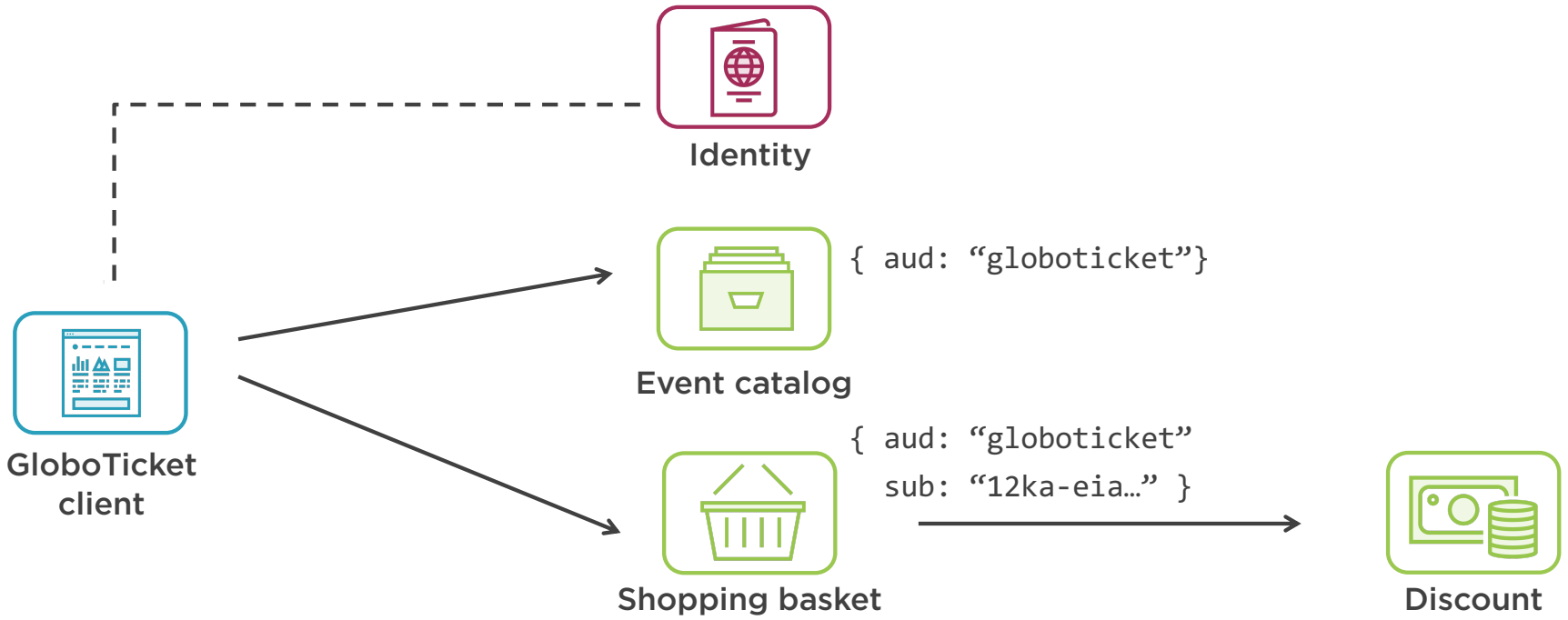
Coming Up

The problems with "one token to rule them all"

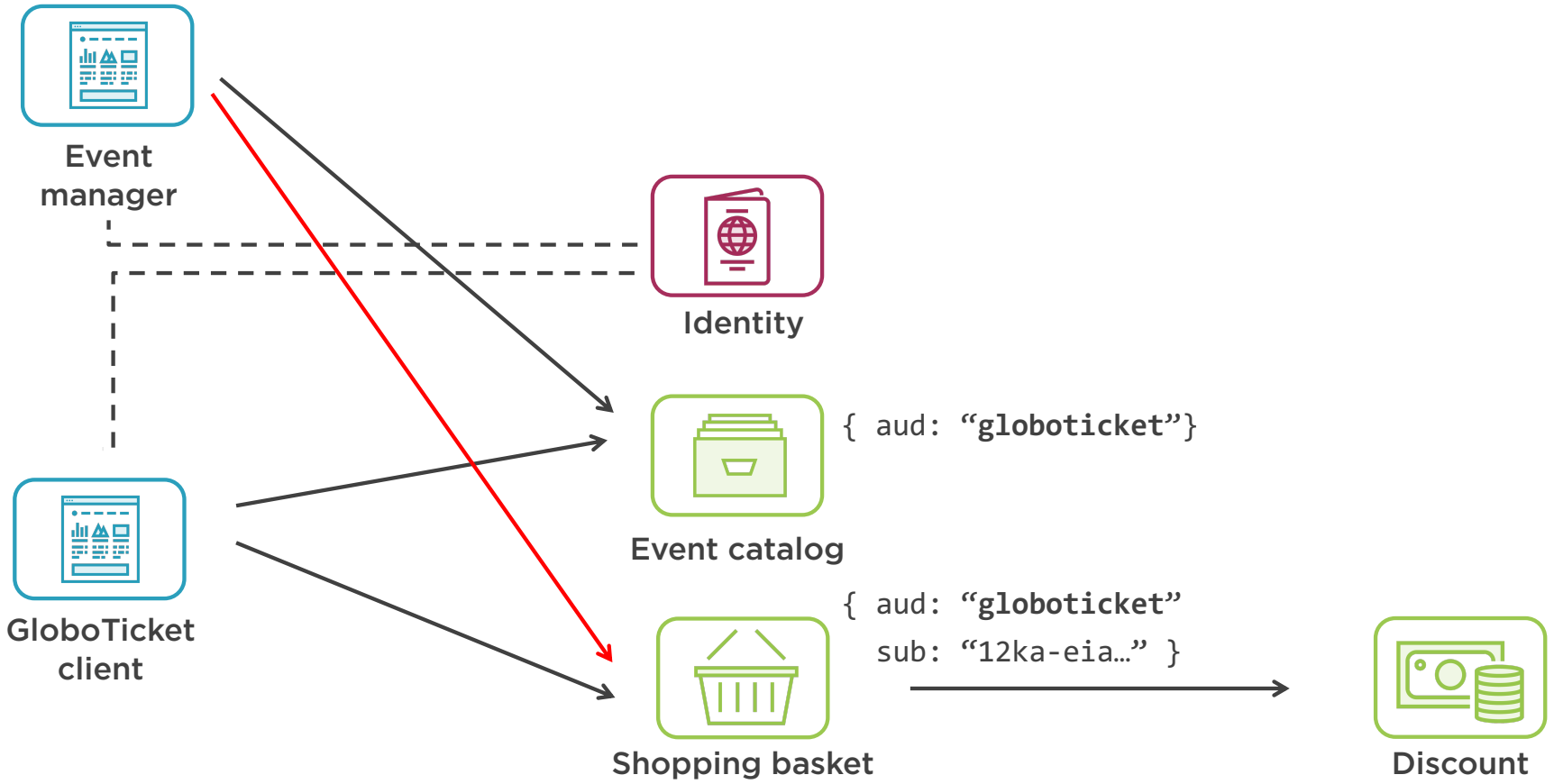
Securing downstream service to service communication on behalf of the user

- Token exchange flow

The Problems with “One Token to Rule Them All”



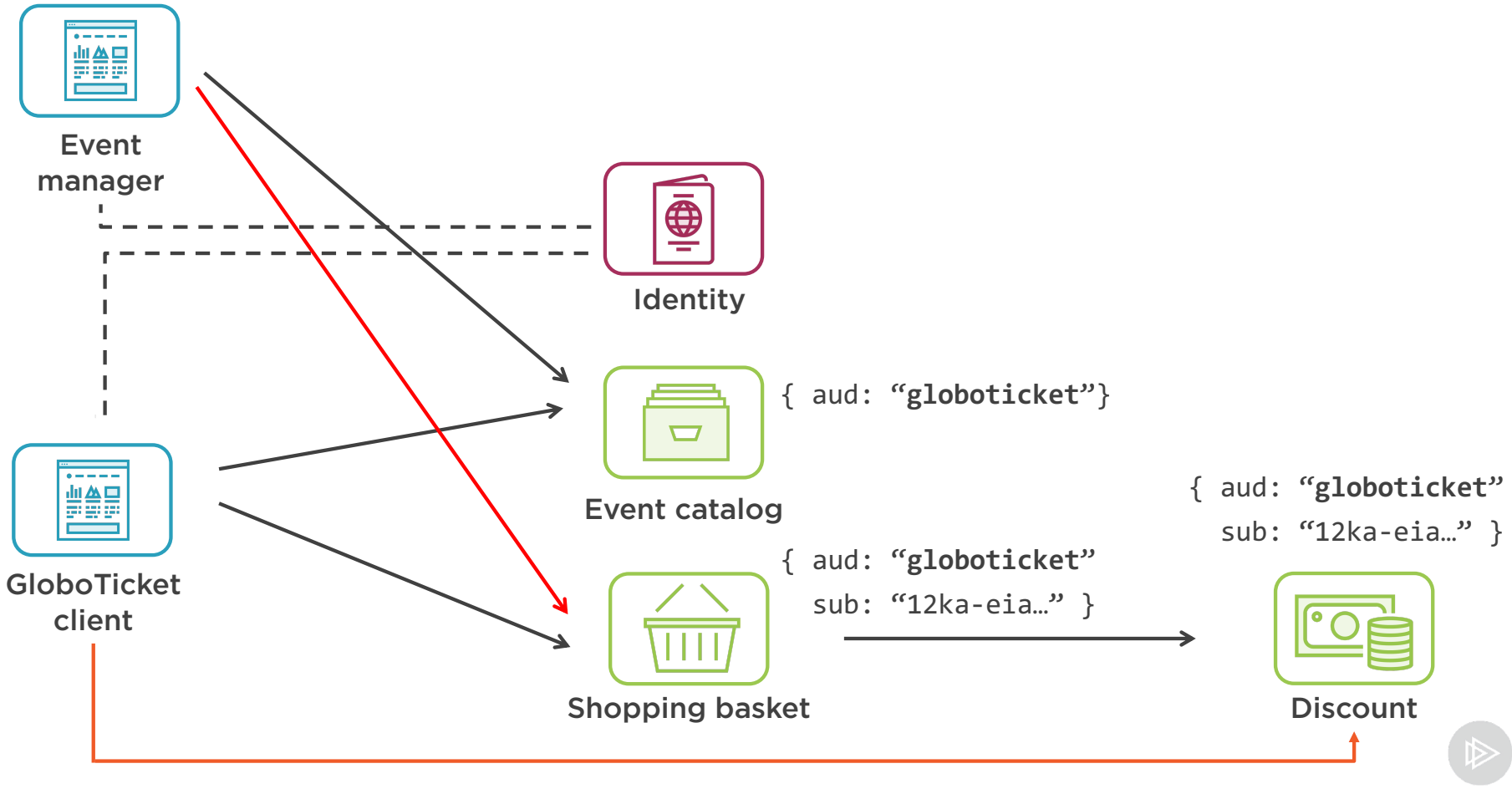
The Problems with “One Token to Rule Them All”



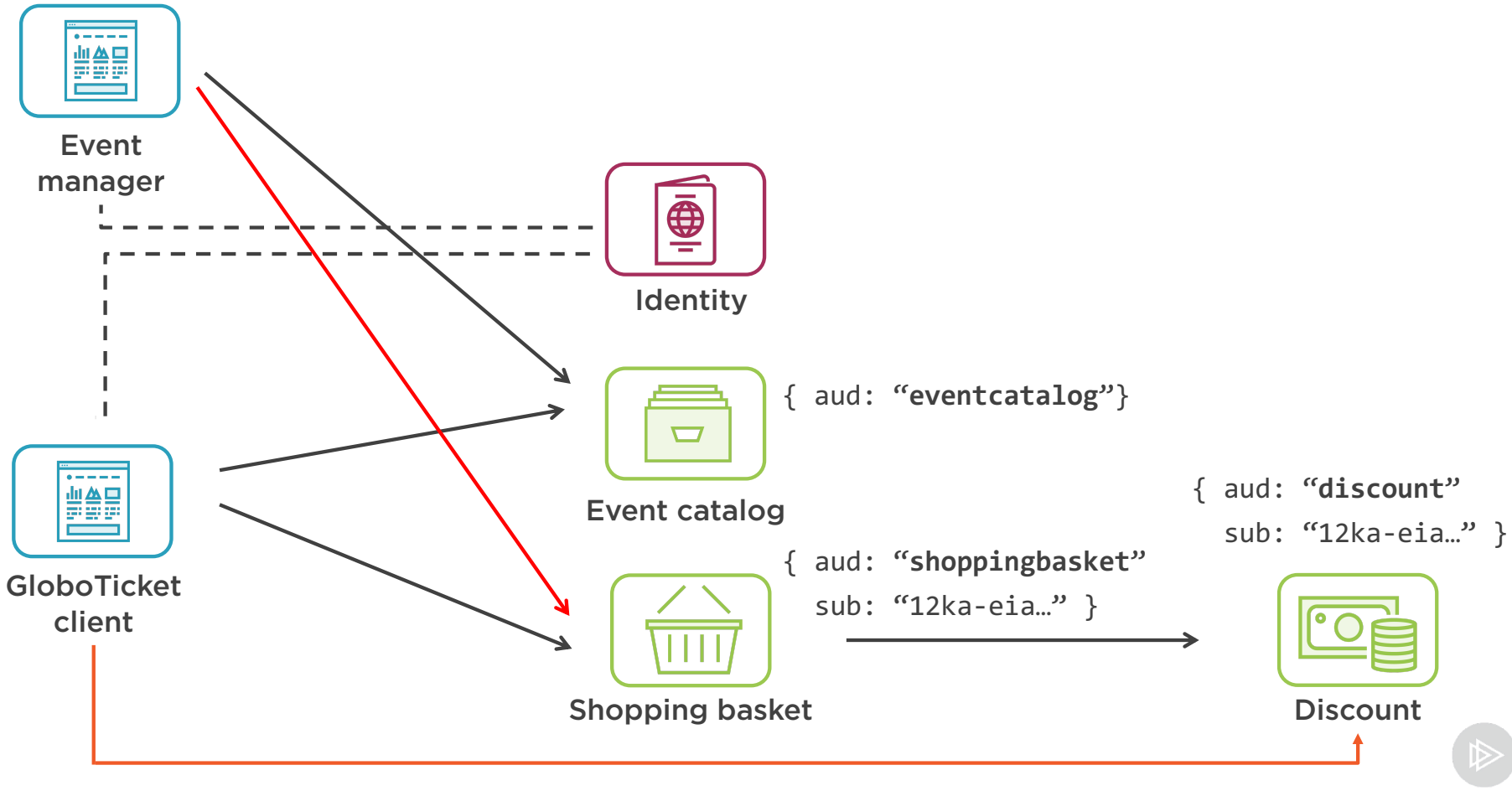
Principle of least privilege

The principle of least privilege is the idea that at any user, program, or process should have only the bare minimum privileges necessary to perform its function

The Problems with "One Token to Rule Them All"



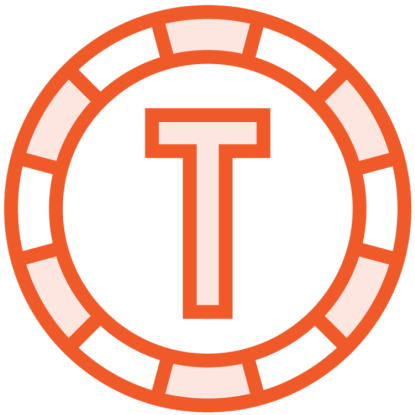
The Problems with "One Token to Rule Them All"



Demo

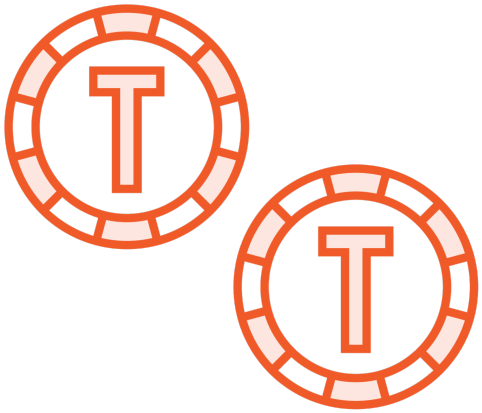
Tightening access with one audience per microservice

Comparing Security Scenarios



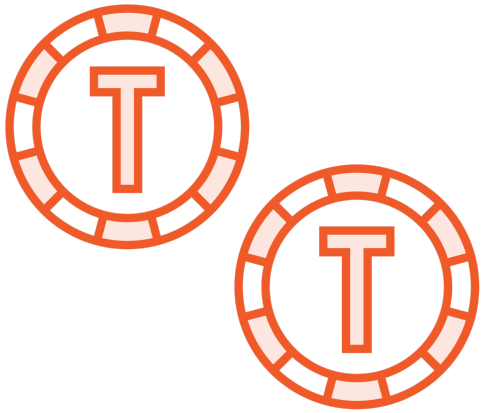
One clientid and clientsecret

```
{ aud: [“eventcatalog, shoppingbasket”], sub: “12ka-eia…” }
```



One clientid and clientsecret

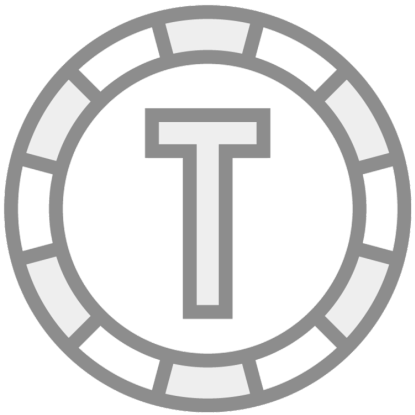
```
{ aud: [“shoppingbasket”], sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```



Two clientids and clientsecrets

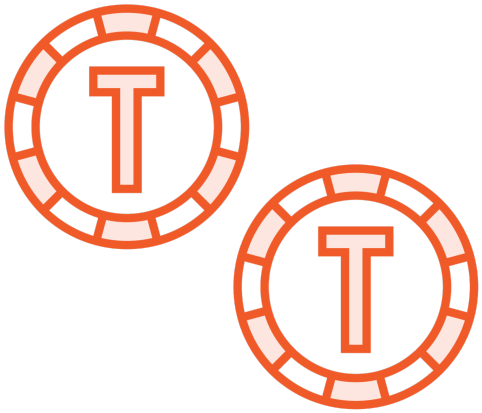
```
{ aud: [“shoppingbasket”], sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```

Comparing Security Scenarios



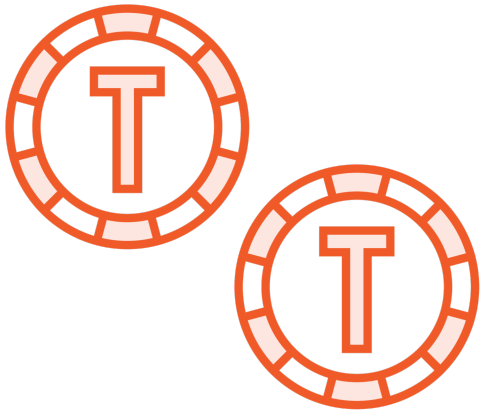
One clientid and clientsecret

```
{ aud: [“eventcatalog,
shoppingbasket”],
sub: “12ka-eia…” }
```



One clientid and clientsecret

```
{ aud: [“shoppingbasket”],
sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```



Two clientids and clientsecrets

```
{ aud: [“shoppingbasket”],
sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```

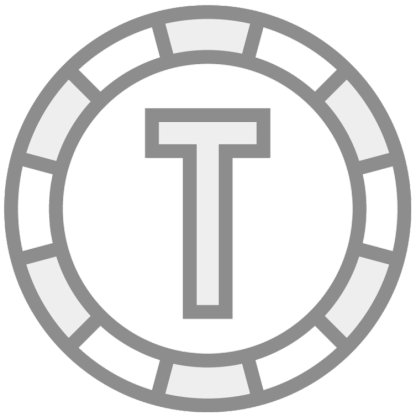
Comparing Security Scenarios

It’s better to have a token with less consent intercepted

But… consider the full system architecture

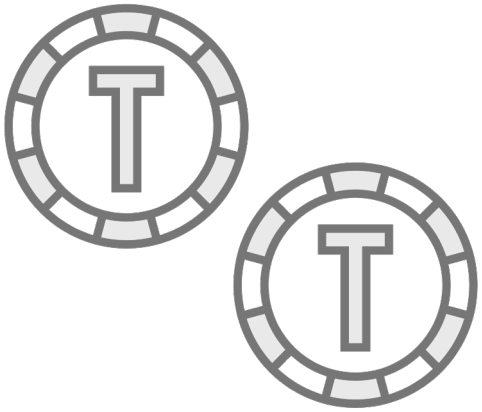
- The advantage may not outweigh the additional cost of ownership

Comparing Security Scenarios



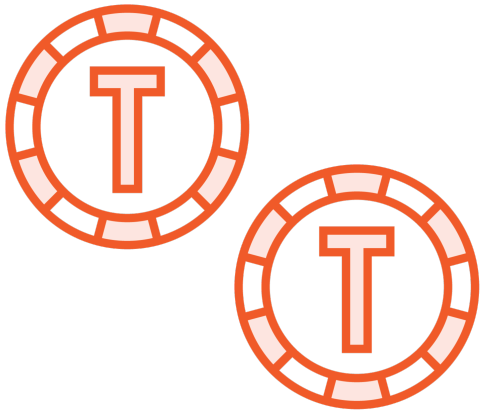
One clientid and clientsecret

```
{ aud: [“eventcatalog, shoppingbasket”],
sub: “12ka-eia…” }
```



One clientid and clientsecret

```
{ aud: [“shoppingbasket”],
sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```



Two clientids and clientsecrets

```
{ aud: [“shoppingbasket”],
sub: “12ka-eia…” }

{ aud: [“eventcatalog”] }
```

Comparing Security Scenarios

For most applications the “two token approach” is overkill

- Some highly secure environments might require it

On application security...

You don’t need the most secure approach. You need the best fit for your application.

Authorization with Scopes Inside of a Microservice

The audience value defines whether or not a client application is allowed access to a service

- Audience: eventcatalog

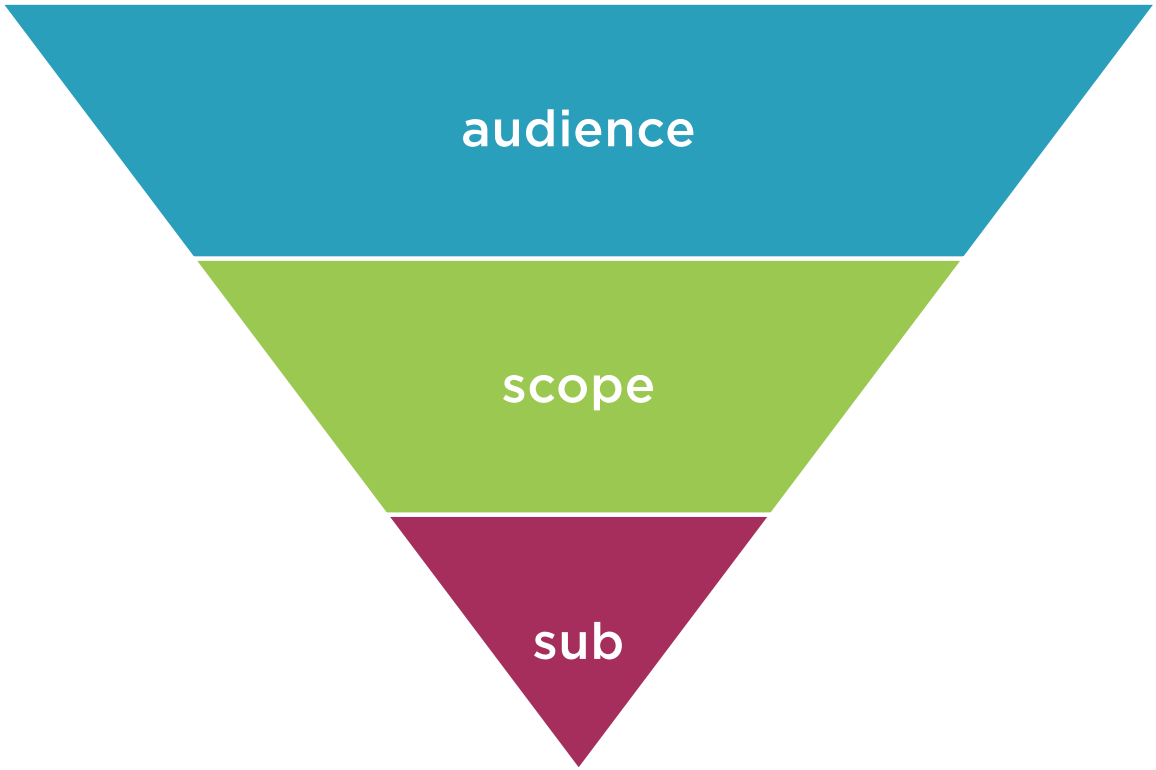
Scopes are used to define what a client application can do *inside* of a service

- Scopes: eventcatalog.read, eventcatalog.write

Audience
service access

Scopes
client-specific rules

Sub
user-specific rules



Demo

Authorization with scopes inside of a microservice

Downstream Service to Service Communication on Behalf of the Client

The shopping basket service should be able to access the discount service

- Token with “discount” audience is required

This makes the shopping basket service a client for accessing the discount service

- If no user is required, use the client credentials flow

Supporting Token Exchange

We need to access the discount service on behalf of the user

- There is no user anymore to interact with our system...

Token exchange standard

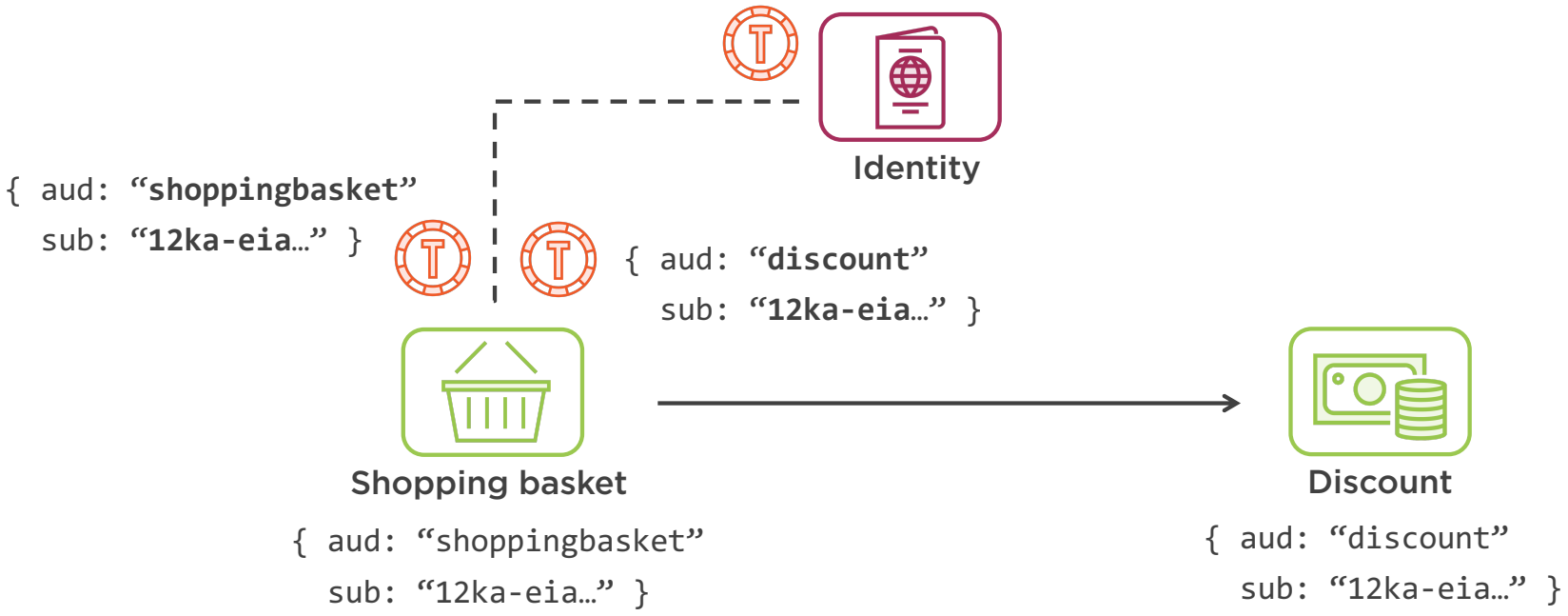
This standard describes how to safely exchange tokens for other tokens, including how to request tokens for employing impersonation and delegation

Supporting Token Exchange

Token exchange standard (RFC8693)

- https://tools.ietf.org/html/rfc8693

Supporting Token Exchange



```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29
tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI…0b3J5In0.PRBg-
jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&scope=discount.fullaccess
```

Token Exchange Syntax

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29
tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI…0b3J5In0.PRBg-
jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&scope=discount.fullaccess
```

Token Exchange Syntax

Grant type: a fixed value (signifying token exchange)

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29
tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI…0b3J5In0.PRBg-
jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&scope=discount.fullaccess
```

Token Exchange Syntax

Subject token: the incoming token

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29
tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI…0b3J5In0.PRBg-
jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&scope=discount.fullaccess
```

Token Exchange Syntax

Subject token type: a fixed value (signifying access token)

```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=eyJhbGciOiJFUzI1NiIsImtpZCI6IjE2In0.eyJhdWQiOiJodHRwczovL2FzLmV4YW1wbGUuY29
tIiwiaXNzIjoiaHR0cHM6Ly9vcmlnaW5hbC1pc3N1ZXI…0b3J5In0.PRBg-
jXn4cJuj1gmYXFiGkZzRuzbXZ_sDxdE98ddW44ufsbWLKd3JJ1VZ
hF64pbTtfjy4VXFVBDaQpKjn5JzAw
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&scope=discount.fullaccess
```

Token Exchange Syntax

Scope: the requested scope(s)

Supporting Token Exchange

Azure AD implements a version of this standard as the "on behalf of" flow

- https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow

Demo

Adding support for the token exchange grant

Demo

Service to service communication on behalf of the user

Further Improvements

Tokens can become very permissive

- Large list of scopes / audiences
- Large list of claims that aren't necessarily required by each service

Further Improvements

For some scenarios that require a high level of security, this might be unwanted

- Tokens can be “split up” into tokens with a smaller attack surface
- The client is not a good place to put this responsibility on

Summary

Problems with "one token to rule them all"

- Doesn't respect principle of least privilege
- Unable to block direct access to downstream services

Require a unique audience per service to avoid these issues

Summary

Audience vs. scope

- Audience value is for (dis)allowing service access in general
- Scopes are used for more granular policies inside of a service
- This is specific to a client, not to a user

Summary

Downstream access on behalf of the user

- Use the token exchange standard