ACCESS CONTROL II

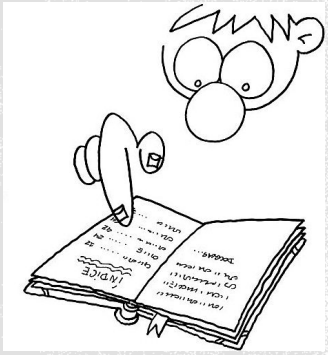
Introduction to Computer and Network Security

Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

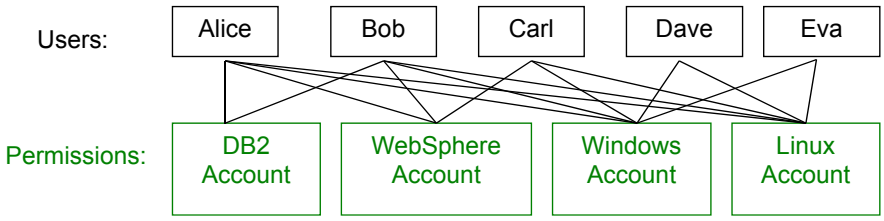
CONTENTS



- Attribute Based AC (ABAC)
  - Overview
  - Reference architecture
- XACML
  - Language for specifying ABAC policies
  - Request/response protocol
  - Reference architecture
- OAuth 2.0
  - Key ideas
  - More details
    - Authentication Code Flow
  - Relationship to capabilities
  - Hints to OpenID Connect

DAC & MAC PROBLEMS

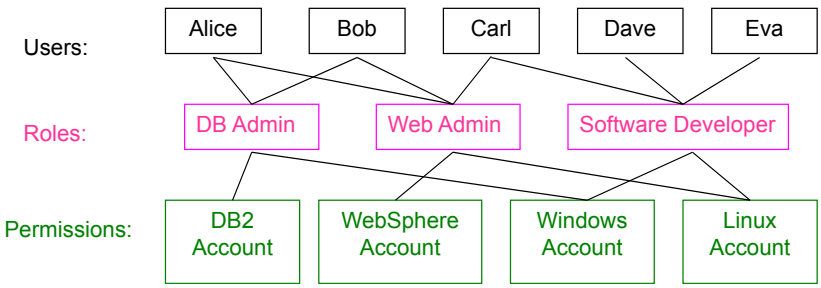
- Complexity of security administration
  - For large number of subjects & objects, the **number of authorizations can become very large**
  - For dynamic user population, the number of **grant & revoke operations** to be performed can become **very difficult to manage**



2

RBAC ADVANTAGES

- Organizations operate based on roles
  - Roles add a useful level of indirection
- RBAC assigns permissions to roles in the organization, rather than directly to users
- With roles, there are fewer relationships to manage
  - Possibly from O(m*n) to O(m+n), where m = num. of users and n = num. of permissions



3

RBAC DISADVANTAGES

- **Roles may not be enough for easily expressing authorization conditions**
  - What about conditions depending on
    - Additional attributes in the profile of a subject?
    - Additional attributes of a resource? E.g., meta-data of files
    - time and location? More in general environment attributes?
- To meet these requirements, RBAC has been extended in several deployments
- **What about mixing different patterns of authorization conditions?**
  - It is possible that a mixture of MLS and RBAC is needed in some situations; their combination is not obvious

ABAC

Attribute Based Access Control

ATTRIBUTE BASED ACCESS CONTROL (ABAC)

- Define authorizations that express conditions on properties of both the resource and the subject
  - Each resource has an attribute (e.g., the subject that created it)
  - A single rule states ownership privileges for the creators

- Strengths
  - Flexibility and expressive power
  - Possibility to combine different patterns of authorization conditions in a natural way
  - Possibility to consider authorization conditions depending on environment attributes

6

ABAC

- ABAC control access based on 3 different attribute types
  - user attributes
  - attributes associated with the resource to be accessed
  - environmental conditions
    - Any available attribute can be used by itself or in combination with another to define the right authorization condition for controlling access to a resource

- Example
  - Allowing only users who are type=employees and have department=HR to access the HR/Payroll system and only during business hours within the same time zone as the company

7

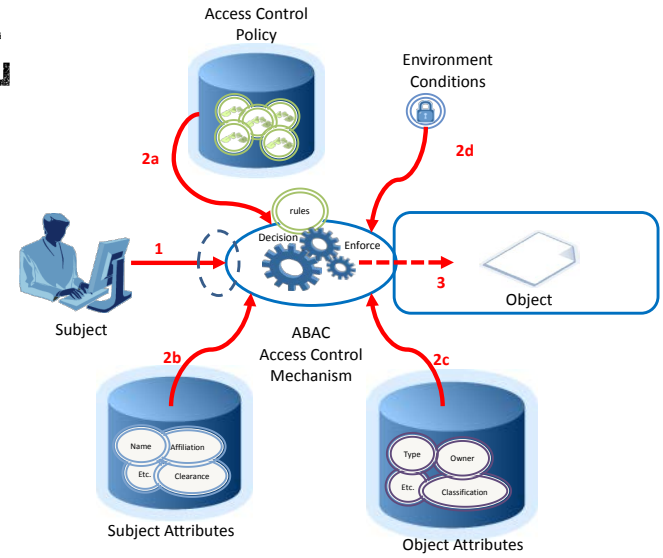
COMPARING ABAC AND RBAC

- **RBAC is for coarse-grain AC and ABAC is for fine-grain AC**

- Examples
  - RBAC
    - Giving all teachers access to Google
  - ABAC
    - Giving teachers access to Google if they are at School X and teach Grade Y

8

THE ABAC MODEL



1. Subject requests access to object
2. Access Control Mechanism evaluates a) Rules, b) Subject Attributes, c) Object Attributes, and d) Environment Conditions to compute a decision
3. Subject is given access to object if authorized

9

THE ABAC MODEL (CONT'D)

- Subjects are associated with attributes
- Objects are associated with attributes
- Environment conditions are associated with attributes
- Authorization is expressed as conditions on these attributes

10

ABAC POLICY

- A **policy** is a set of rules that govern allowable behavior within an organization, based on the privileges of subjects and how resources or objects are to be protected under which environment conditions
- Typically written from the perspective of the object that needs protection and the privileges available to subjects
- Privileges represent the authorized behavior of a subject and are defined by an authority and embodied in a policy

11

ABAC POLICY: EXAMPLE

- MPEG adult movies can only be downloaded by users whose age is greater than 18
- **Authorization does not refer to specific user**
  - Applies to all users whose age is greater than 18 years
- **Authorization does not refer to specific resource**
  - MPEG movies have an attribute that denotes their type
    - In this case it is adult movies

12

SUBJECT ATTRIBUTES

- A subject is an active entity that causes information to flow among objects or changes the system state
- **Attributes define the identity and characteristics of the subject**, e.g.,
  - Name
  - Organization
  - Job title
  - Role
  - …

13

OBJECT/RESOURCE ATTRIBUTES

- An object (or resource) is a passive information system-related entity containing or receiving information

- **Objects have attributes that can be leveraged to make access control decisions**, e.g.,
  - Title
  - Author
  - Date of creation
  - Size
  - Categories of content
  - …

14

ENVIRONMENT ATTRIBUTES

- **Describe the operational, technical, and even situational environment or context in which the information access occurs**, e.g.,
  - Current date
  - Current virus/hacker activities
  - Network security level
  - Any feature *not associated with a resource or subject*

- These attributes have been largely ignored in most access control policies

15

ABAC AND PREVIOUS AC MODELS (1)

- ABAC can mimic all previous AC models
- To see how this is possible, it is sufficient to observe that it is sufficient to introduce appropriate attributes for subjects and resources (and possibly appropriate function or predicates) so as to be able to express the conditions for granting or denying access of the corresponding AC model
- Example: **ACLs**
  - Introduce
    - an attribute *sid* for subject identifiers
    - an attribute *rid* for resource identifiers
  - for a resource identifier *rid1* consider its ACL: rid1-> [sid1 -> {r,w}, sid2 ->{w}, …]
  - add the following rules:
    - r.rid = rid1 & s.sid = id1 & (a.id = r | a.id = w)
    - r.rid = rid1 & s.sid = id2 & a.id = w
    - …

identifier of a resource

identifiers of subjects

- & denotes logical AND
- | denotes logical OR
- *e.a* (dot notation) allows for referencing attribute *a* of entity *e*

ABAC AND PREVIOUS AC MODELS (2)

- Example: **RBAC**
- Introduce
  - an attribute *role* for subjects
  - an attribute *permission* for resources
- For (u,r) in UA and (r,p) in PA
- add the following rule:
  - s.sid = u & s.role = r & r.permission = p & r.rid = get-resource(p) & a.id = get-action(p)
- We assume that a permission p is a pair (rid, aid) where rid is an identifier of a resource and aid is an identifier of an action. Notice the additional functions that are assumed to be such that
  - get-resource((rid,aid)) = rid
  - get-action((rid,aid)) = aid

17

18

XACML

eXtensible Access Control Markup Language is an OASIS standard

XACML: MORE THAN A LANGUAGE FOR ABAC POLICIES

- XACML = e**X**tensible **A**ccess **C**ontrol **M**arkup **L**anguage is an OASIS standard
- Developed for collaborative environments
  - Data sharing across different organizational domains
- XACML is extensible and is an XML encoded language
- Can specify access control policies, access control requests, and access control decisions and contains **more than policy specification language …**

19

XACML COMPONENTS

1. **XACML policy language**
   - Specify access control rules
   - Algorithms for combining policies
2. **XACML request/response protocol**
   - Used to query a decision engine that evaluates user access requests against policies
3. **XACML reference architecture**
   - For deployment of software modules to house policies and attributes and compute and enforce access control decisions

20

VOCABULARY

- **Resource**
  - Data or system component needing protection
- **Subject**
  - An actor who requests access to specific resources
- **Action**
  - An operation on a resource
- **Environment**
  - Properties not belonging to resources, subjects, or actions that are important for the authorization decision
- **Attributes**
  - Characteristics of the resource, subject, action, or the environment
- **Target**
  - Defines conditions that determine whether policy applies to the request

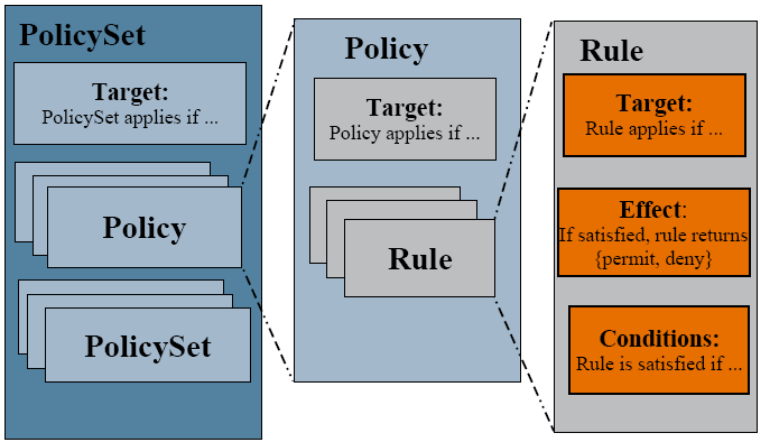
21

XACML REQUESTS

- XACML access request consists of attributes of subject, resource, action, and environment
- XACML attributes are name-value pairs
  - Role = "Doctor", ObjectAttr = "Medical Record"
- Attributes are stored in a **Policy Information Point (PIP)** and retrieved at the time of decision making

22

XACML POLICY STRUCTURE



23

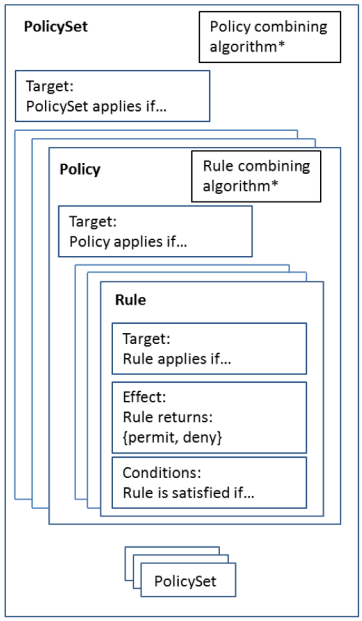
XACML POLICIES

The account that follows on XACML is simplified in many respects to avoid technicalities and convey the main ideas

- XACML policies are structured as PolicySets
- PolicySets consist of Policies and may include other PolicySets
- Policies are composed of Rules
- Target defines a Boolean condition
  - If true, the request gets evaluated by a PDP
  - If false, the decision is Not Applicable
- Target minimizes the PolicySets, Policy, and Rules that must be examined

24

XACML POLICIES



(*) The following rules exist for Policy Combining and Rule Combining algorithms:

- Permit-overrides
- Deny-overrides
- First-applicable
- Only-one-applicable

25

XACML RULES

- Rules have a set of Boolean conditions
- Rules evaluate to **true** or **false** or **indeterminate**
- Policy can have multiple rules
- Rules can be combined by **rule combining algorithms**
- There are 12 rule combining algorithms available but we will only see 4…

26

RULE COMBINING ALGORITHMS

- Four commonly used rule combining algorithm
  - Deny overrides
    - AND operation on Permit
  - Permit overrides
    - OR operation on Permit
  - First applicable
    - Result is the result of the first decision
  - Only one applicable
    - If more than one decision applies, then the result is Indeterminate

- Notice that rule combining algorithms can be used to solve rule and policy conflicts
- Example
  - rule 1 grants access to a set S of subjects to a given resource r
  - rule 2 denies access to the same set S of subjects to the same resource r
  - DenyOverrides(rule 1, rule 2) specifies that in case of conflict, we want to be cautious and deny access possibly causing a bit of problems wrt business continuity
  - PermitOverrieds(rule 1, rule 2) specifies that in case of conflict, we want to guarantee business continuity and possibly be less secure

OBLIGATIONS

- XACML includes the concept of obligations
- **Obligation describes what must be carried out before or after an access request is approved and denied**
- If Alice is denied access to Document X, email her manager that Alice tried to access document X

28

XACML RULE EXAMPLE

- Anyone with the **developer role** can do anything to any resource
- Notes
  - **Rules** can be separately evaluated, but they **cannot live on their own**: they must be part of a Policy
  - *Rules are the smallest unit of reuse in XACML*
  - *Policies are the smallest unit of evaluation*

```
01. <Rule RuleId="rul-0001" Effect="Permit">
02.   <Description>
03.     Some optional text that explains the purpose of the rule
04.   </Description>
05.   <Target>
06.     <Subjects>
07.       <Subject>
08.         <SubjectMatch MatchId=
09.             "urn:oasis:names:tc:xacml:2.0:function:string-equal">
10.           <AttributeValue DataType=
11.               "http://www.w3.org/2001/XMLSchema#string">
12.             developer
13.           </AttributeValue>
14.           <SubjectAttributeDesignator>
15.             role
16.           </SubjectAttributeDesignator>
17.         </SubjectMatch>
18.       </Subject>
19.     </Subjects>
20.   </Target>
21. </Rule>
```

In short, this is equivalent to the following condition
subject.role = developer

29

XACML POLICY EXAMPLE

- Previous rule wrapped in a policy
- Notes
  - The **RuleCombiningAlgId** attribute on the Policy identifies the algorithm that combines Effects from multiple Rules into a single result

```
<Policy PolicyId="pol-0001" RuleCombiningAlgId=
    "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>
    Some optional text that explains the purpose of the policy
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId=
            "urn:oasis:names:tc:xacml:2.0:function:string-equal">
          <AttributeValue DataType=
              "http://www.w3.org/2001/XMLSchema#string">
            developer
          </AttributeValue>
          <SubjectAttributeDesignator>
            role
          </SubjectAttributeDesignator>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <Rule RuleId="rul-0001" Effect="Permit"/>
</Policy>
```

30

XACML POLICYSET EXAMPLE

- Previous policy wrapped in a policy set
- Notes
  - A **Policy Set** contains a Target, a Policy-Combining Algorithm, a set of Policies, and some Obligations
  - The **Policy-Combining Algorithm** specifies the procedure by which the results of evaluating the component Policies are combined
  - A Policy Set can reuse not just Policies, but also entire Policy Sets

```
<PolicySet PolicySetId="pls-0001" PolicyCombiningAlgId=
    "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
  <Description>
    Some optional text that explains the purpose of the policy set
  </Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId=
            "urn:oasis:names:tc:xacml:2.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            developer
          </AttributeValue>
          <SubjectAttributeDesignator>
            role
          </SubjectAttributeDesignator>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>
  <PolicyIdReference>
    pol-0001
  </PolicyIdReference>
  <Policy PolicyId="pol-0001" RuleCombiningAlgId=
      "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
    <Target/>
    <Rule RuleId="rul-0001" Effect="Permit"/>
  </Policy>
</PolicySet>
```

31

XACML REQUEST EXAMPLE

- Subject has an Attribute named role, that our example Rule refers to using the SubjectAttributeDesignator element
- Notes
  - **Request** consists of attributes for the Subject(s), Resource, Action, and Environment
  - **Attributes** are named, so that Rules can refer to them, and strongly typed, so that the PDP can do proper comparisons.
  - **Environment** Attributes allow different access decisions to be made, for example depending on whether the Subject is in a public place or within the secure confines of his office
  - PDP will match the Target of the Policy Sets, Policies, and Rules against the Request to see whether they are applicable

```
<Request>
  <Subject>
    <Attribute AttributeId="role"
        DataType="http://www.w3.org/2001/XMLSchema#string">
      developer
    </Attribute>
  </Subject>
  <Resource>
    <ResourceContent>
      <!--
        In case the Resource is an XML document, it can be inlined
        here, so that Rules can depend on the contents.
      -->
      <example/>
    </ResourceContent>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
      /some/example.xml
    </Attribute>
  </Resource>
  <Action>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
      retrieve.contents
    </Attribute>
  </Action>
  <Environment>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#dateTime"
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime">
      2010-06-14T11:12:03Z
    </Attribute>
  </Environment>
</Request>
```

In short, this is equivalent to the following authorization request
(subject.role = developer, resource.resource-id=/some/example.xml, action.action-id=retrieve.contents, environment.current-dateTime=2010-06-14T11:12:03Z)

32

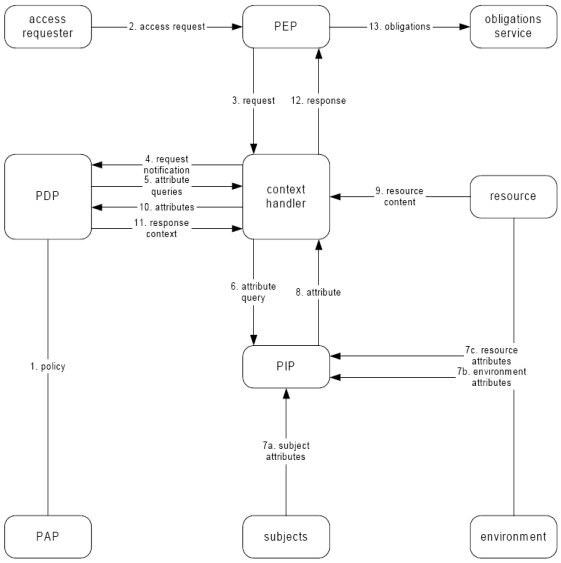
XACML RESPONSE EXAMPLE

- When the PDP has reached a verdict on whether the Subject may perform the Action on the Resource in the given Environment, it creates a Response object
- The Context Handler returns the Response to the PEP, which enforces the decision
- The Result may also include Obligations that the PEP must fulfill
  - Ex: filter out credit card information before returning the Resource content to the Subject
  - Ex: write an audit record
- PEP must either honor all of these Obligations or treat the decision as denied.

```
<Response>
  <Result ResourceId="/some/example.xml">
    <Decision>Permit</Decision>
  </Result>
</Response>
```

33

XACML ARCHITECTURE FOR ENFORCEMENT

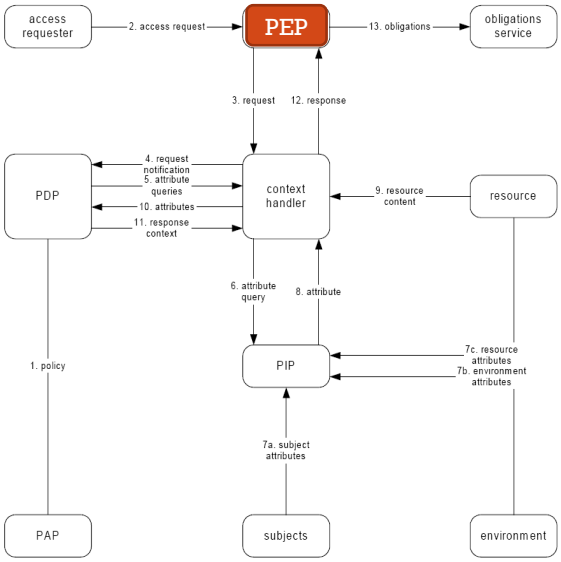


34

XACML ARCHITECTURE FOR ENFORCEMENT

Policy Enforcement Point (PEP)

- Entity protecting the resource(e.g. file system)
- Performs access control by making decision requests and enforcing authorization decisions

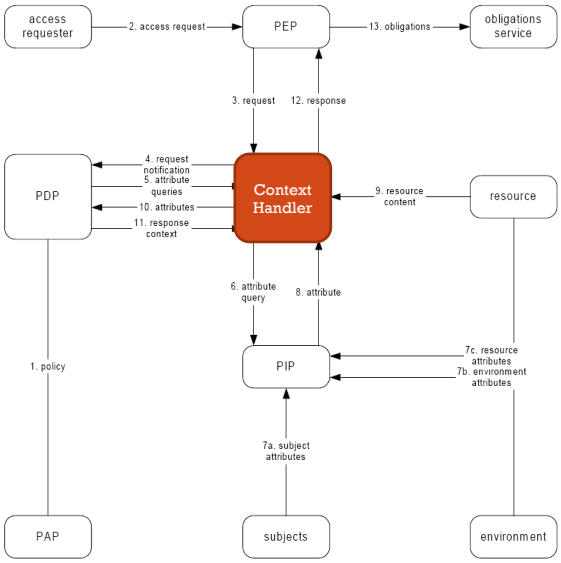


35

XACML ARCHITECTURE FOR ENFORCEMENT

Context Handler

- A Context is the canonical representation of a decision request and an authorization decision.
- Context Handler can be defined to convert the requests in its native format to the XACML canonical form and to convert the Authorization decisions in the XACML canonical form to the native format.

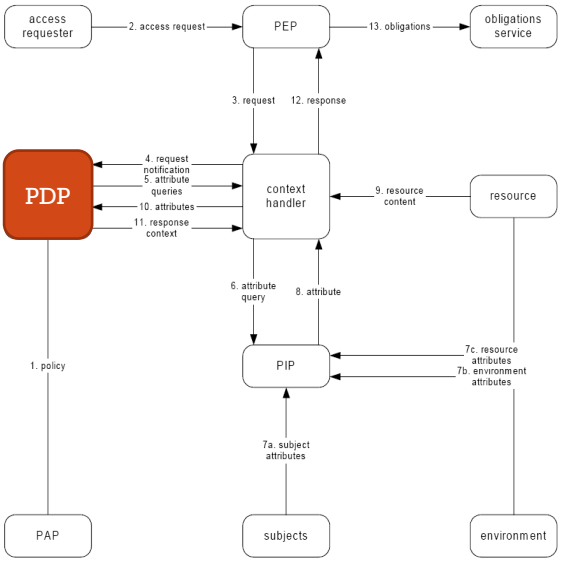


36

XACML ARCHITECTURE FOR ENFORCEMENT

The Policy Decision Point (PDP)

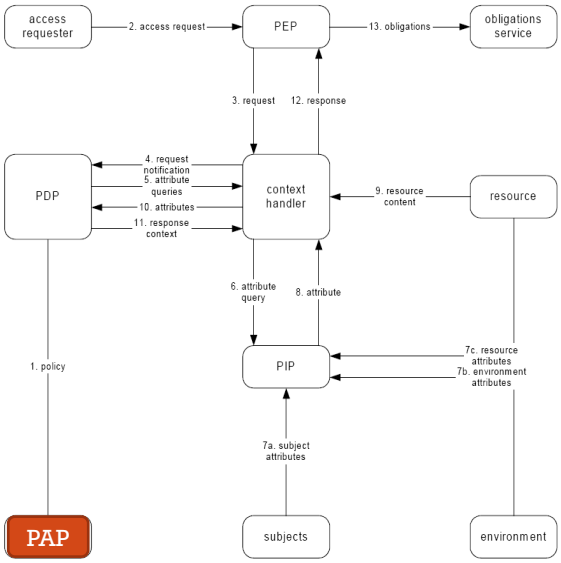
- Receives and examines the request
- Retrieves applicable policies
- Evaluates the applicable policy
- Returns the authorization decision to PEP



37

XACML ARCHITECTURE FOR ENFORCEMENT

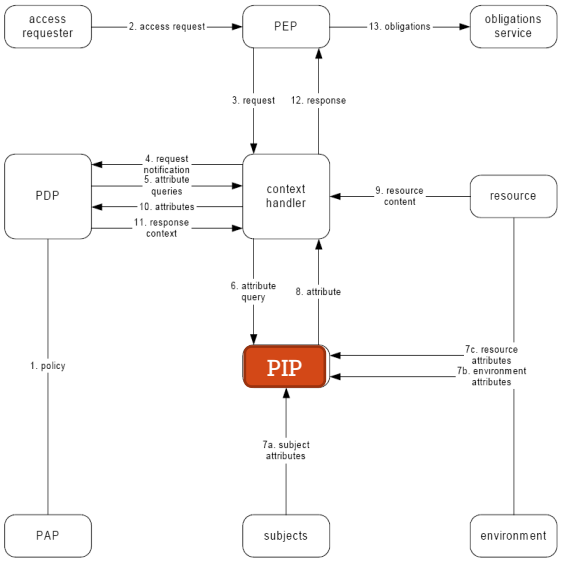
Policy Administration Point (PAP)
creates security policies and stores these policies in the repository.



38

XACML ARCHITECTURE FOR ENFORCEMENT

Policy Information Point (PIP)
serves as the source of attribute values, or the data required for policy evaluation.



39

40

OAUTH 2.0: KEY IDEAS

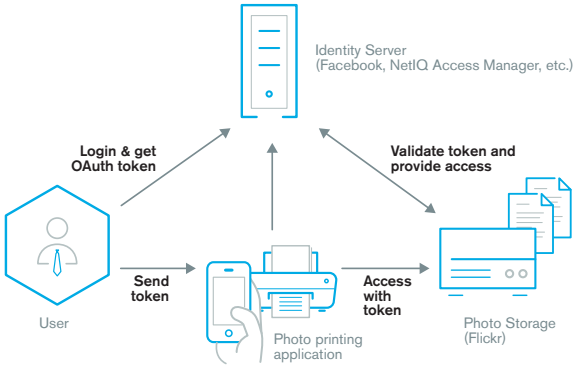
OAUTH 2.0: AN ANALOGY

- Many cars today come with a **valet key**
- It is a special key you give the parking attendant and unlike your regular key, will not allow the car to drive more than a mile or two
- Some valet keys will not open the trunk, while others will block access to your onboard cell phone address book
- Regardless of what restrictions the valet key imposes, the idea is clear:
  - you give someone limited access to your car with a special key, while using your regular key to unlock everything

41

OAUTH 2.0: AN EXAMPLE

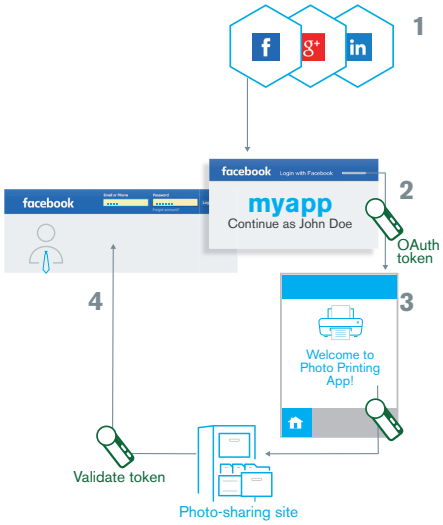
- Consider a photo lab printing your online photos
- Straightforward implementations may request to provide your username and password to the other site
- **When you agree to share your secret credentials, not only do you expose your password to someone else, you also give them full access to do as they wish**
- They can do anything they want – even change your password and lock you out
- This is the problem OAuth solves
  - **It allows you (users) to grant access to your private resources on one site (which is called the Service Provider), to another site (called Consumer/Client)**



42

OAUTH 2.0: THE FLOW

- **Step 1: Authentication** (User logs into social site; not part of OAuth 2.0 protocol)
  - The focus of OAuth 2 is only authorization and leaves authentication to the application
  - OAuth 2 simply requires the user to be authenticated and does not dictate how to do this
  - Extensions to Oauth 2 such as Open ID Connect (OIDC) can be useful for authentication
- **Step 2: User Consent**
  - OAuth 2.0 allows users to decide what can be shared with 3rd-party apps)
  - An OAuth token will be created based on the rights to which the token user consents
- **Step 3: Get OAuth Token**
  - The third-party printing app receives an OAuth bearer token from the social site
  - This token includes details about the access rights of the token bearer
  - The application can then use this token to access the photos on behalf of this user
  - A token is analogous to a valet key
- **Step 4: Access Resource**
  - The printing app can now access the resource server (Flickr) using the token to get the user's data (photos)



43

44 OAUTH 2.0: MORE DETAILS

OAUTH 2.0: DEF FROM RFC6749 (IETF)

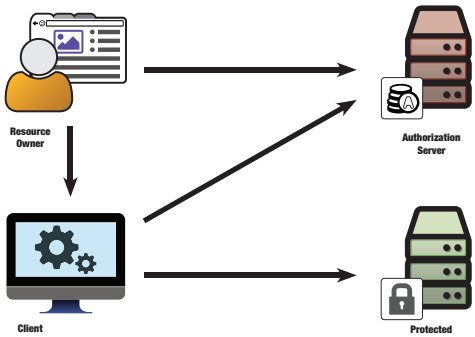
- The OAuth 2.0 authorization framework enables
  - a third-party application to obtain limited access to an HTTP service,
  - either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service,
  - or …

More down-to-earth definition
OAuth 2.0 is a *delegation protocol* that lets users allow applications to access resources on their behalf

45

OAUTH 2.0: INVOLVED ENTITIES

- **Resource owner**
  - Can access to certain resources
  - Can delegate access to resources
  - Is usually a person
- **Protected resource**
  - Service provider protecting resources for their owner
  - Shares resources on owner's request
- **Client** (app)
  - Wish to access protected resources
  - Acts on owner's behalf
- **Authorization server**
  - Generates tokens for the client
  - Authenticate resource owners and clients
  - Manages authorizations

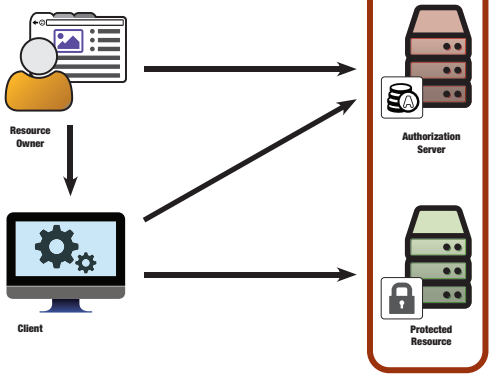


46

OAUTH 2.0: INVOLVED ENTITIES

In many cases, these two "live" on the same server… although this is not always the case…

- **Resource owner**
  - Can access to certain resources
  - Can delegate access to resources
  - Is usually a person
- **Protected resource**
  - Service provider protecting resources for their owner
  - Shares resources on owner's request
- **Client** (app)
  - Wish to access protected resources
  - Acts on owner's behalf
- **Authorization server**
  - Generates tokes for the client
  - Authenticate resource owners and clients & manages authorizations



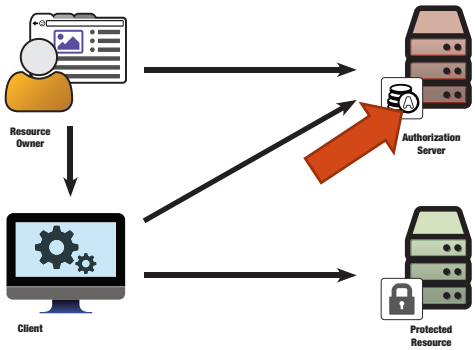
47

OAUTH 2.0: INVOLVED ENTITIES (CONT'D)

OAuth token

- Represent granted delegated authorities
  - From the resource owner to the client for the protected resource
- Issued by authorization server
- Used by client
  - Format is opaque **to clients**
  - Examples
    - 92d42038006dba95d0c501951ac5b5eb
    - 2df029c6-b38d-4083-b8d9-db67c774d13f
- Consumed by protected resource

- Token is not opaque to AS and RS
- How can AS and RS understand what is inside a token?
  - Database lookup
  - Put info in the token (JWT)
  - RS query the AS



48

OAUTH 2.0: REMARKS

- Core protocol defined only for HTTP
  - Relies on TLS for securing messages

OAuth 2.0
TLS
HTTP

- **Not an authentication protocol**
  - Relies on authentication in several places
    - Client authentication to token endpoint
    - Resource owner authentication at authorization endpoint
  - **Authentication protocols can be built using OAuth (OpenID Connect)**
    - More on this protocol later in these slides

OpenID Connect
OAuth 2.0
TLS
HTTP

- Allows a user to delegate to a piece of software but not to another user

49

OAUTH 2.0: REMARKS (CONT'D)

- No authorization processing
  - Tokens can represent scopes and other authorization information
  - Processing of this information is up to the resource server
  - However, several methods (e.g., Jason Web Token) to communicate this information
- No token format
  - Token is opaque to the client
  - **Token needs to be issued by the authorization server and understood by the resource server, but they're free to use whatever format they want**
    - JSON Web Tokens (JWT) provide a useful common format
- Not a single protocol
  - OAuth 2.0 is a *framework* consisting of several flows (more on this in the following slides)

DIGRESSION ON JWT

JWT (1)

- JWT = JSON Web Token
- De facto standard for OAuth 2.0 access token
  - OAuth Standard does not impose any particular format
- JWT consists of 3 components
  - **Header**
    - identifies algorithm used to sign
  - Payload
  - Signature

- JWT defines a container to transport data between interested parties
- Became an IETF standard in May 2015 with the RFC 7519

```
{
 "alg" :  RS256 ,
 "typ" : "JWT"
}

base64url encoded string: eyBhbGciOiBIUzI1NiwgdHlwIDogSldUIH0K
```

52

JWT (2)

- JWT = JSON Web Token
- De facto standard for OAuth 2.0 access token
  - OAuth Standard does not impose any particular format
- JWT consists of 3 components
  - **Header**
    - identifies algorithm used to sign
  - Payload
  - Signature

- RS256 = RSA Signature with SHA-256 asymmetric algorithm using a public/private key pair:
  - IdP has a private key used to generate the signature
  - the consumer of the JWT gets a public key to validate the signature
  - Most IdPs make public keys easily available for consumers to obtain and use through a metadata URL
  - If you don't have control over the client, RS256 is suggested since the consumer only needs to know the public key

```
{
 "alg" :  RS256 ,
 "typ" : "JWT"
}

base64url encoded string: eyBhbGciOiBIUzI1NiwgdHlwIDogSldUIH0K
```

53

JWT (3)

- JWT = JSON Web Token
- De facto standard for OAuth 2.0 access token
  - OAuth Standard does not impose any particular format
- JWT consists of 3 components
  - **Header**
    - identifies algorithm used to sign
  - Payload
  - Signature

```
{
 "alg" : "HS256",
 "typ" : "JWT"
}

base64url encoded string: eyBhbGcgOiBIUzI1NiwgdHlwIDogSldUIH0K
```

- **Base64** is a way to **encode** binary data into an ASCII character set known to pretty much every computer system, in order to transmit the data without loss or modification of the contents itself
- **Base64url** encoding is basically base64 encoding except for the use of non-reserved URL characters (e.g., – is used instead of + and _ is used instead of /) and omit the padding characters

54

JWT (4)

- JWT = JSON Web Token
- De facto standard for OAuth 2.0 access token
  - OAuth Standard does not impose any particular format
- JWT consists of 3 components
  - Header
  - **Payload**
    - information actually used for access control
  - Signature

```
{
 "user_name" : "admin",
}

base64url encoded string: eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg
```

55

JWT (5)

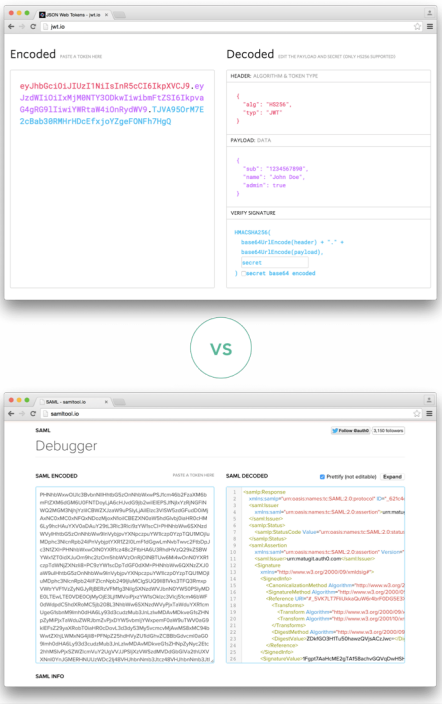
- JWT = JSON Web Token
- De facto standard for OAuth 2.0 access token
  - OAuth Standard does not impose any particular format
- JWT consists of 3 components
  - Header
  - Payload
  - **Signature**
    - used to validate that the token has not been tampered with
    - calculated by concatenating the header with the payload, then signing with the algorithm specified in the header (e.g., RS256)

56

JWT (6)

- JWT = JSON Web Token
- De facto format for OAuth 2.0 access token
  - Standard does not impose any particular format
- JWT consists of 3 components
  - Header
  - Payload
  - Signature
- Complete token obtained by concatenating the 3 components above

57



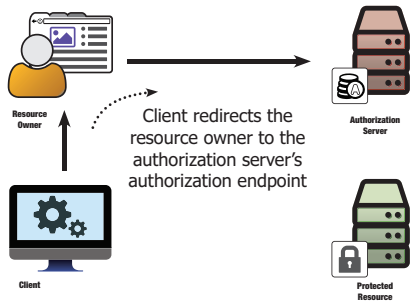
- JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML
  - This makes JWT a good choice to be passed in HTML and HTTP environments
- Both JWT tokens and SAML responses can use a public/private key pair in the form of a X.509 certificate for signing
  - Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON
- JSON parsers are common in most programming languages because they map directly to objects; XML doesn't have a natural document-to-object mapping
- JWT is used at Internet scale. This highlights the ease of client-side processing of the JSON Web token on multiple platforms, especially mobile

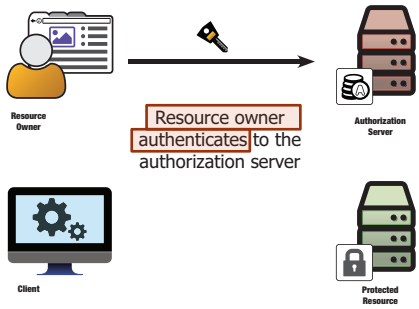
58

59

END OF DIGRESSION ON JWT

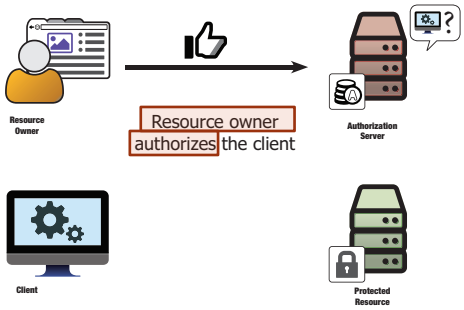
AUTH CODE FLOW: STEP 1-2

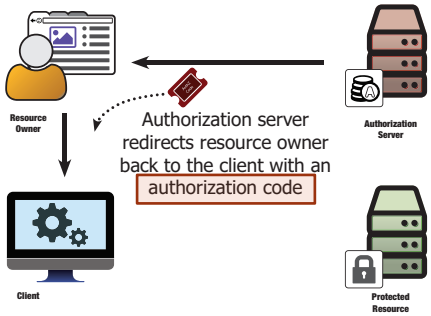




60

AUTH CODE FLOW: STEP 3-4

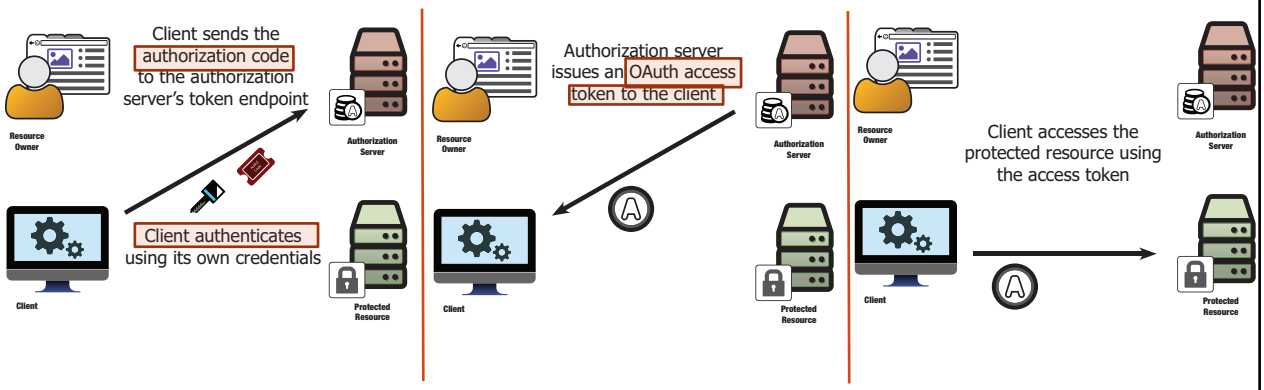




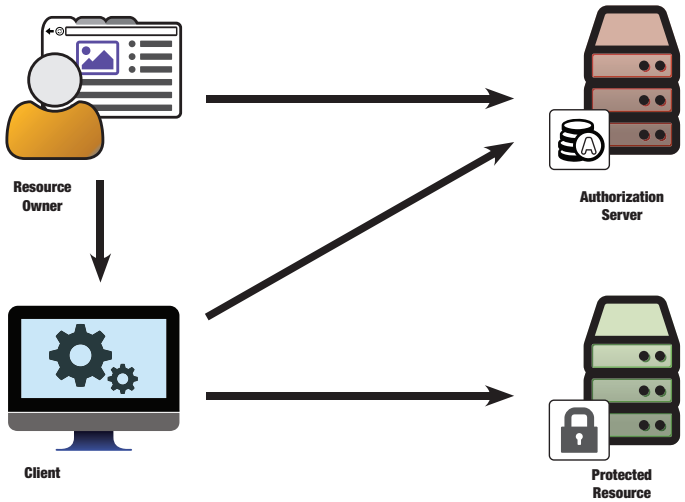
This is not yet the access token that allows the client to access the protected resource… can you guess why?

61

AUTH CODE FLOW: STEP 5-7



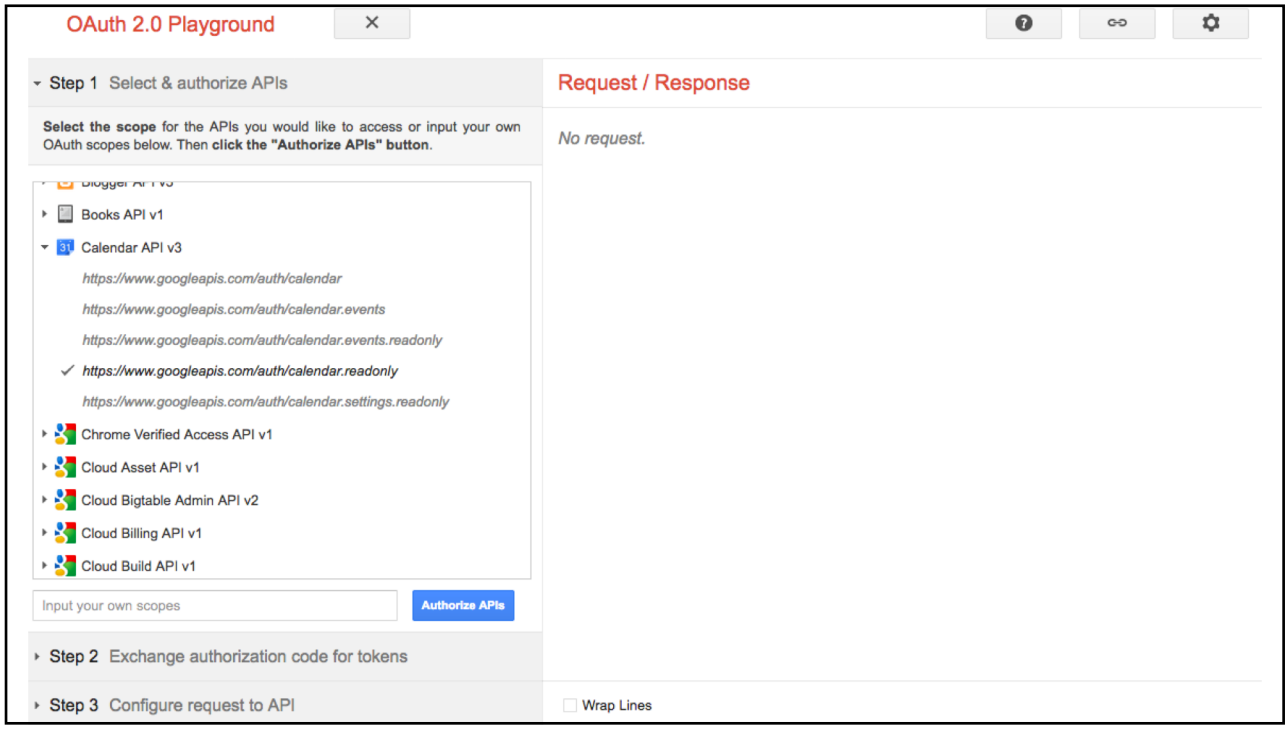
62

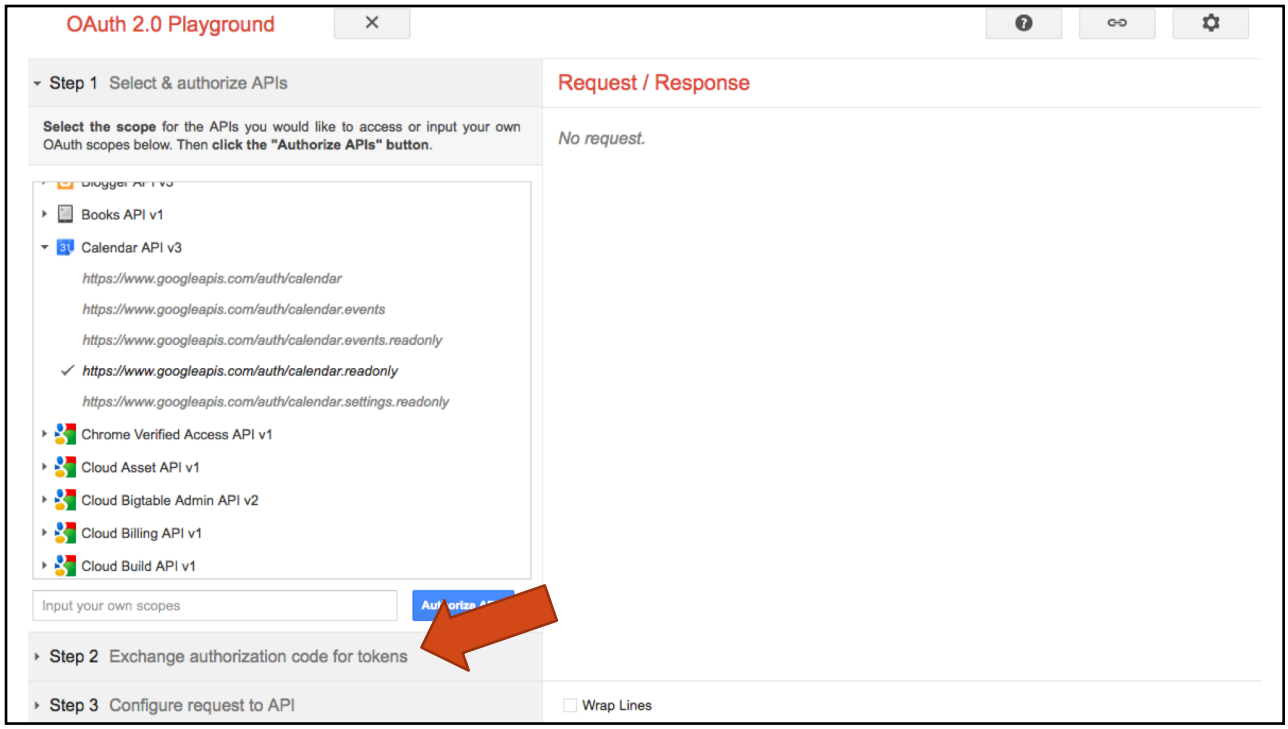


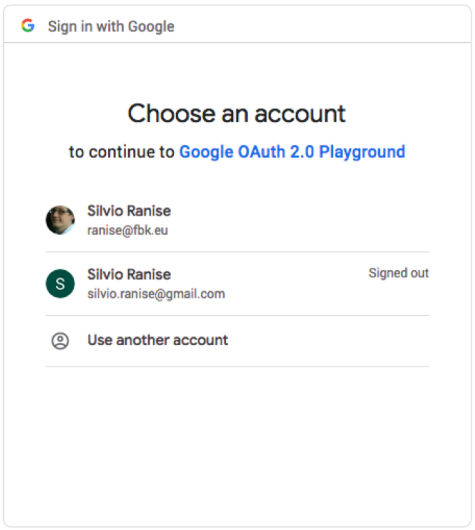
AN EXAMPLE

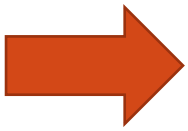
- Resource Owner:
  - Me
- Client:
  - OAuth 2.0 Playground
  - https://developers.google.com/oauthplayground/
- Authorization server:
  - https://accounts.google.com/o/oauth2/v2/auth
- Protected resource:
  - My calendar (ranise@fbk.eu)
  - Accessible through google API
    - https://www.googleapis.com/auth/calendar.readonly

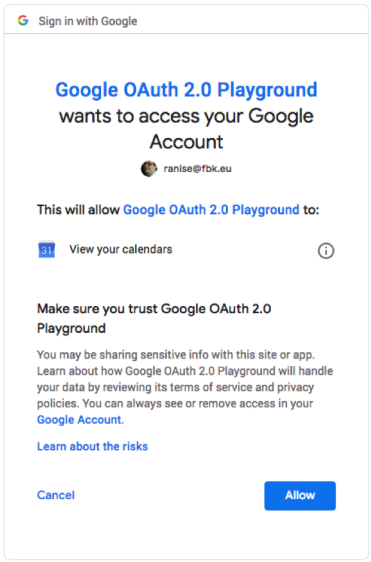
63

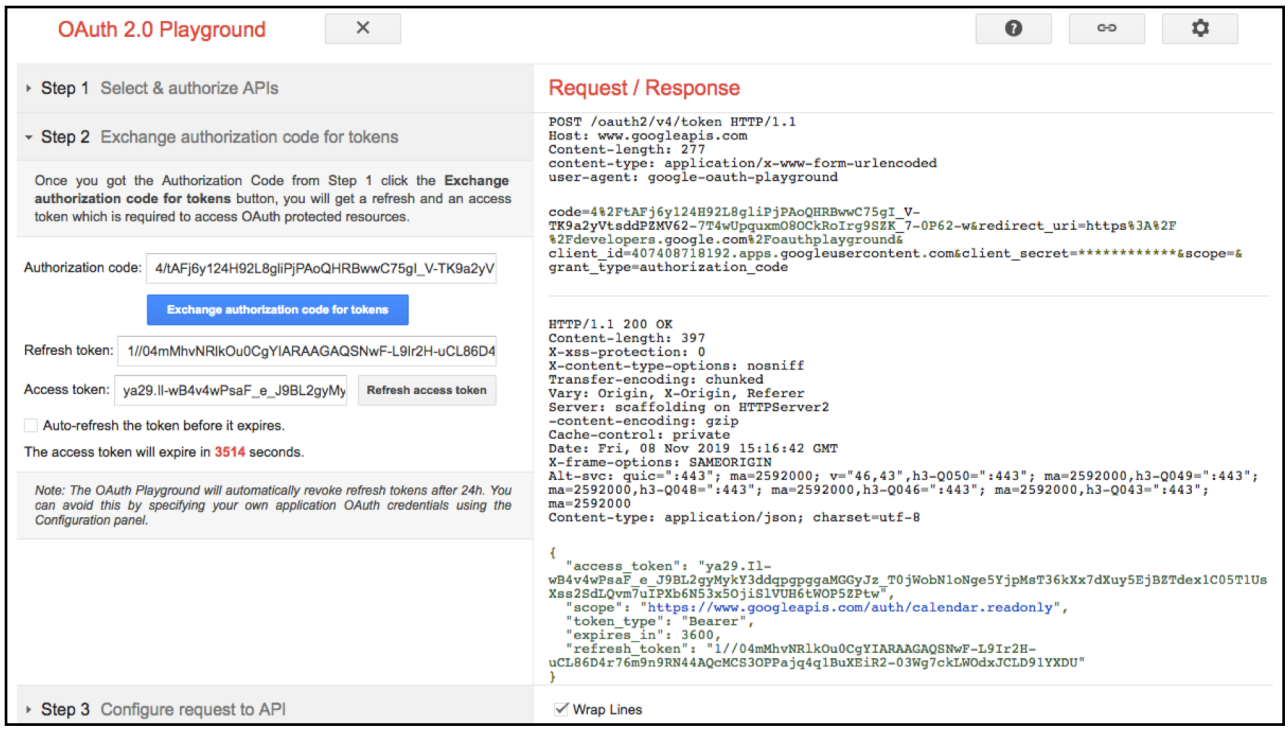

OAuth 2.0 Playground
Step 1 Select & authorize APIs
Select the scope for the APIs you would like to access or input your own OAuth scopes below. Then click the "Authorize APIs" button.
Books API v1
Calendar API v3
https://www.googleapis.com/auth/calendar
https://www.googleapis.com/auth/calendar.events
https://www.googleapis.com/auth/calendar.events.readonly
https://www.googleapis.com/auth/calendar.readonly
https://www.googleapis.com/auth/calendar.settings.readonly
Chrome Verified Access API v1
Cloud Asset API v1
Cloud Bigtable Admin API v2
Cloud Billing API v1
Cloud Build API v1
Input your own scopes
Authorize APIs
Step 2 Exchange authorization code for tokens
Step 3 Configure request to API
Request / Response
No request.
Wrap Lines

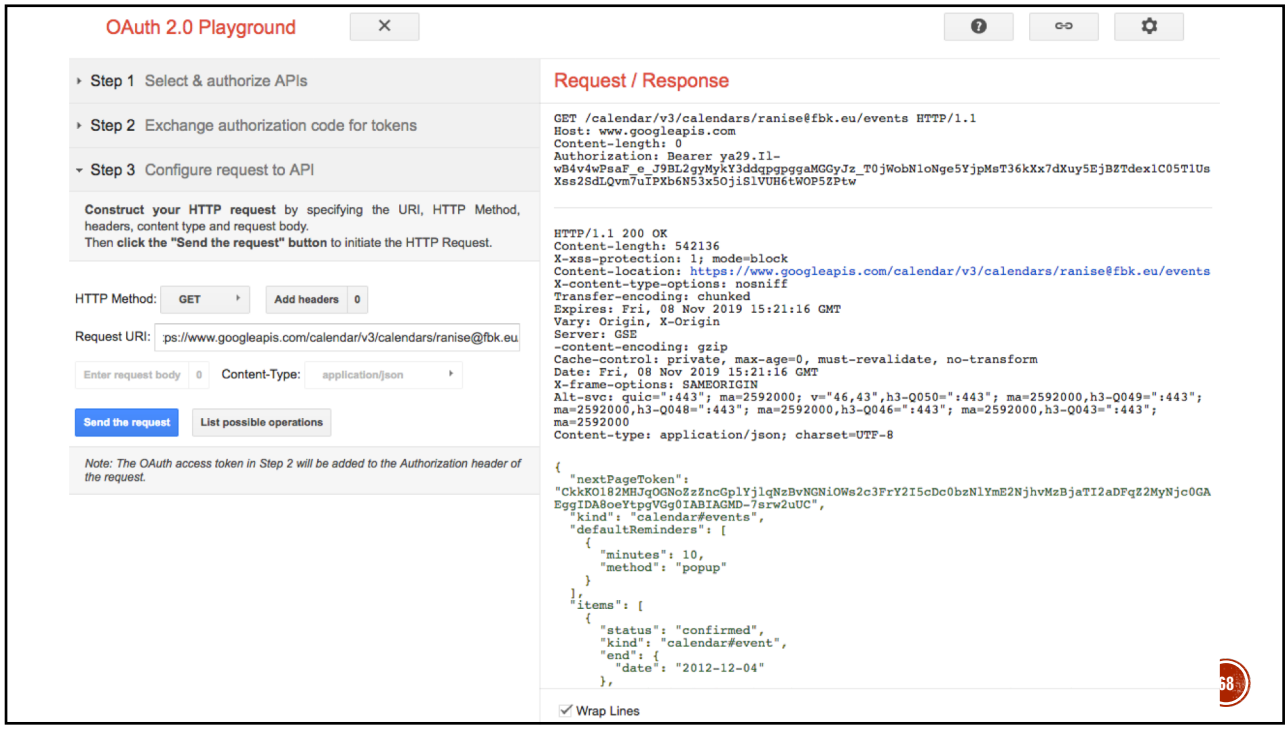

OAuth 2.0 Playground
Step 1 Select & authorize APIs
Select the scope for the APIs you would like to access or input your own OAuth scopes below. Then click the "Authorize APIs" button.
Books API v1
Calendar API v3
https://www.googleapis.com/auth/calendar
https://www.googleapis.com/auth/calendar.events
https://www.googleapis.com/auth/calendar.events.readonly
https://www.googleapis.com/auth/calendar.readonly
https://www.googleapis.com/auth/calendar.settings.readonly
Chrome Verified Access API v1
Cloud Asset API v1
Cloud Bigtable Admin API v2
Cloud Billing API v1
Cloud Build API v1
Input your own scopes
Step 2 Exchange authorization code for tokens
Step 3 Configure request to API
Request / Response
No request.
Wrap Lines


Sign in with Google
Choose an account
to continue to Google OAuth 2.0 Playground
Silvio Ranise
ranise@fbk.eu
Silvio Ranise
silvio.ranise@gmail.com
Signed out
Use another account



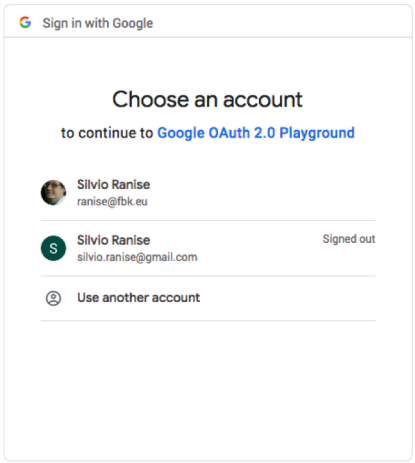

Sign in with Google
Google OAuth 2.0 Playground
wants to access your Google Account
ranise@fbk.eu
This will allow Google OAuth 2.0 Playground to:
View your calendars
Make sure you trust Google OAuth 2.0 Playground
You may be sharing sensitive info with this site or app. Learn about how Google OAuth 2.0 Playground will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.
Learn about the risks
Cancel
Allow


OAuth 2.0 Playground
Step 1 Select & authorize APIs
Step 2 Exchange authorization code for tokens
Once you got the Authorization Code from Step 1 click the Exchange authorization code for tokens button, you will get a refresh and an access token which is required to access OAuth protected resources.
Authorization code: 4/tAFj6y124H92L8gliPjPAoQHRBwwC75gI_V-TK9a2yV
Exchange authorization code for tokens
Refresh token: 1//04mMhvNRlkOu0CgYIARAAGAQSNwF-L9Ir2H-uCL86D4
Access token: ya29.Il-wB4v4wPsaF_e_J9BL2gyMy
Refresh access token
Auto-refresh the token before it expires.
The access token will expire in 3514 seconds.
Note: The OAuth Playground will automatically revoke refresh tokens after 24h. You can avoid this by specifying your own application OAuth credentials using the Configuration panel.
Step 3 Configure request to API
Request / Response
POST /oauth2/v4/token HTTP/1.1
Host: www.googleapis.com
Content-length: 277
content-type: application/x-www-form-urlencoded
user-agent: google-oauth-playground
code=4%2FtAFj6y124H92L8gliPjPAoQHRBwwC75gI_V-TK9a2yVtsddPZMV62-7T4wUpquxmO8OCkRoIrg9SZK_7-0P62-w&redirect_uri=https%3A%2F%2Fdevelopers.google.com%2Foauthplayground&client_id=407408718192.apps.googleusercontent.com&client_secret=***********&scope=&grant_type=authorization_code
HTTP/1.1 200 OK
Content-length: 397
X-xss-protection: 0
X-content-type-options: nosniff
Transfer-encoding: chunked
Vary: Origin, X-Origin, Referer
Server: scaffolding on HTTPServer2
-content-encoding: gzip
Cache-control: private
Date: Fri, 08 Nov 2019 15:16:42 GMT
X-frame-options: SAMEORIGIN
Alt-svc: quic=":443"; ma=2592000; v="46,43",h3-Q050=":443"; ma=2592000,h3-Q049=":443"; ma=2592000,h3-Q048=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000
Content-type: application/json; charset=utf-8
{
"access_token": "ya29.Il-wB4v4wPsaF_e_J9BL2gyMykY3ddqpgpggaMGGyJz_T0jWobN1oNge5YjpMsT36kXx7dXuy5EjBZTdex1C05T1UsXss2SdLQvm7uIPXb6N53x5OjiSlVUH6tWOP5ZPtw",
"scope": "https://www.googleapis.com/auth/calendar.readonly",
"token_type": "Bearer",
"expires_in": 3600,
"refresh_token": "1//04mMhvNRlkOu0CgYIARAAGAQSNwF-L9Ir2H-uCL86D4r76m9n9RN44AQcMCS3OPPajq4q1BuXEiR2-03Wg7ckLWOdxJCLD91YXDU"
}
Wrap Lines



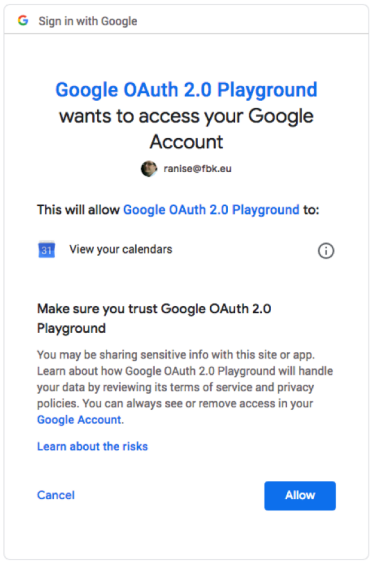
OAUTH 2.0: KEY POINTS (1)

1. Need to log into the Provider's OAuth service when redirected



OAUTH 2.0: KEY POINTS (2)

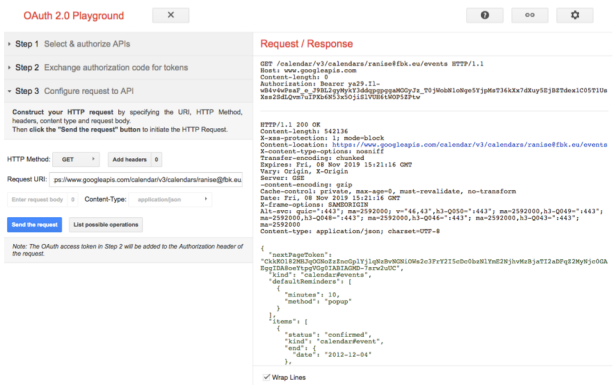
1. Need to log into the Provider's OAuth service when redirected
2. Explicit consent to limited access
   - **Scope**: string that represent what the token can do
     - Client can ask for scopes
     - Resource owner approves scopes
     - Access token is bound to scopes
   - Type of action
     - Read, write, delete
   - Type of resource
     - Photos, metadata, profile
   - Time of access
     - User is offline, limited number of accesses



70

OAUTH 2.0: KEY POINTS (3)

1. Need to log into the Provider's OAuth service when redirected
2. Explicit consent to limited access
3. Protected Resource validates access token (i.e. requested access) when getting it from the Client
   - Need of suitable cryptographic mechanisms to protect integrity and avoid forging!



71

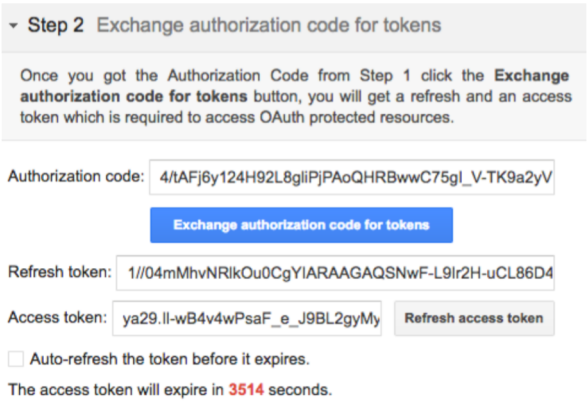
MORE POINTS: REFRESH TOKENS

- When the user is no more there… **access tokens** work after the user leaves
- What does a client do when the access token stops working?
  - Expiration
  - Revocation
- Getting a new token
  - Repeat the process of getting a token
    - Interactive grants: send the resource owner to the authorization endpoint
  - But what if the user's not there anymore?
- **Refresh tokens**
  - Issued alongside the access token
  - Used for getting new access tokens
    - Presented along with client credentials
    - Not good for calling protected resources directly

72

MORE POINTS: AUTHORIZATION CODES

- Why do clients need to obtain authorization codes before access tokens?
- Clients cannot be trusted
  - E.g., Man in the browser
- APIs do not know who is calling them (**bearer token**)
  - Anyone presenting the access token will be granted access!
- To prevent this kind of problems, exchange is done server-to-server and requires both the client_id and client_secret, preventing even the (client) resource owner from obtaining the access token



73

RELATIONSHIP TO CAPABILITIES

- Recall that capabilities collect privileges for a user
  - Contrast these with ACLs
    - They collect users per privilege
  - Recall the access control matrix concept

- OAuth tokens are **similar** to capabilities
  - They can be transferred to other subjects so that permissions are delegated
  - **Permissions are decoupled from the identities of subjects**

74

75

USING OAUTH 2.0 FOR AUTHENTICATION?

It turns out to be a bad idea…

WHY IT IS A BAD IDEA

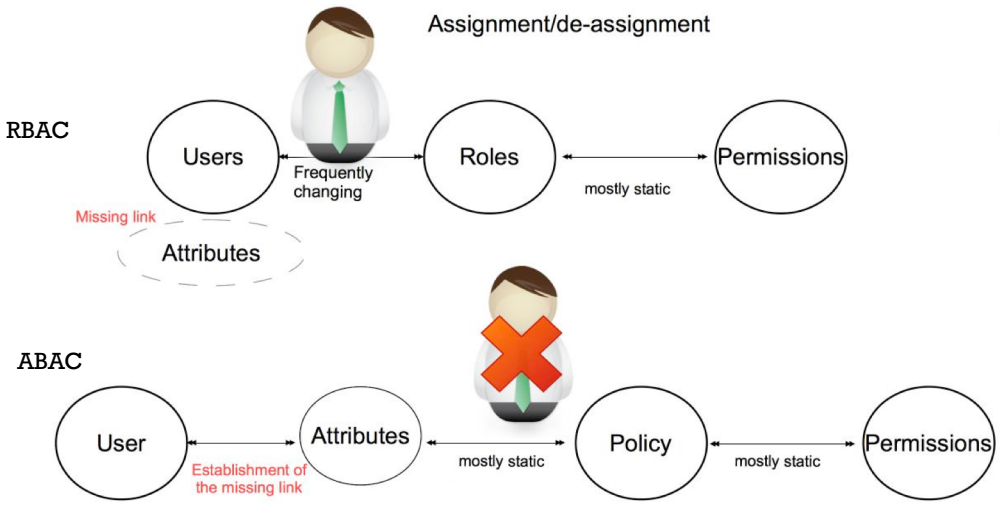
- Excerpt from the OAuth 2.0 standard https://oauth.net/articles/authentication/
  - Authenticating resource owners to clients is **out of scope** for this specification
- Key observation
  - The assumption that **possession of a valid access token is enough to prove that a user is authenticated** is **true only in some cases** (when the access token was freshly minted)
  - There are other ways to obtain a valid access tokens than authenticating resource owner; e.g., using the **refresh token**
- Authentication is about the user and their **presence** with the application

OPENID CONNECT

https://openid.net/connect/

- Open standard published in early 2014 that defines an interoperable way to use OAuth 2.0 to perform user authentication
- Instead of building a different protocol to each potential identity provider, an application can speak one protocol to as many providers as they want to work with
- OpenID Connect is built directly on OAuth 2.0
- OpenID Connect manages to avoid many of the pitfalls discussed above by adding several key components to the OAuth base…

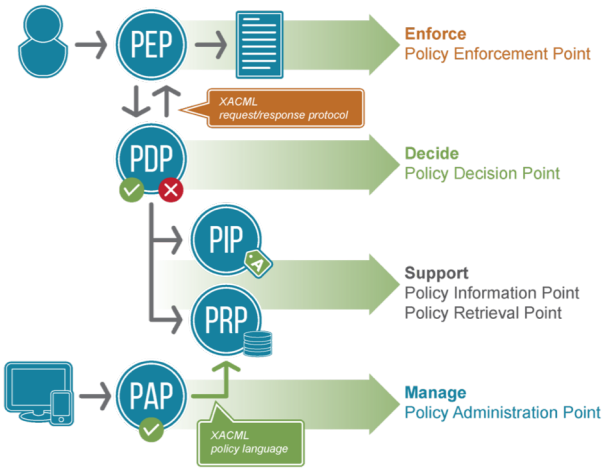
TAKEAWAYS: ABAC



78

TAKEAWAYS: XACML

- XACML is an OASIS standard
  - It has even a RBAC profile (for both core and hierarchical)
- A **standardized** approach to authorization
  - Authorization rules no more embedded in the SW of individual systems
  - Focus is on corporate policies rather than the technicalities of SW environments
- An **externalized** approach to authorization
  - PDP offers authorization as a service in the infrastructure
  - Authorization algorithms can be removed from the application logic of individual information systems, which will then query the PDP via their own Policy Enforcement Points (PEP)
- An **attribute** and **policy based** approach to authorization
  - XACML policies introduce abstract logic to replace previous static assignments of user permissions
  - Instead of "Bob can access document X" a rule "any user belonging to company X with security clearance equal to or higher than the security classification of a document should be granted access to that document"
  - To determine whether Bob should be granted access to document X, his security clearance as well as the document classification needs to be gathered. These descriptive pieces of information are called **attributes**



79

TAKEAWAYS: OAUTH 2.0

- Key mechanism to delegate permissions without distributing credentials
- Recall to use TLS to secure communications
- It can be used as the starting point for authentication protocols
  - OpenIdConnect

80

RECAP QUESTIONS

- What is the problem solved by OAuth? Which entities are involved in Oauth?
- What is an OAuth token? Is it opaque for which entity involved in OAuth?
- What is ABAC? What is an ABAC policy? What are the advantages of ABAC over RBAC?
- What is XACML? What is a XACML target, effect, condition, rule policy, and policy set? What are the XACML policy combining algorithms?
- Describe the XACML architecture
- What is the problem that the OAuth 2.0 standard solve?
- Describe the Authorization Code flow
- Explain why an access token is not immediately created for the client and instead an authorization code is created
- Describe the structure of JWT
- Why is a bad idea to use OAuth 2.0 for authentication? What should you use instead?

81