

# ACCESS CONTROL II

Introduction to Computer and Network Security

*Silvio Ranise* [ [silvio.ranise@unitn.it](mailto:silvio.ranise@unitn.it) or [ranise@fbk.eu](mailto:ranise@fbk.eu) ]



UNIVERSITÀ  
DI TRENTO



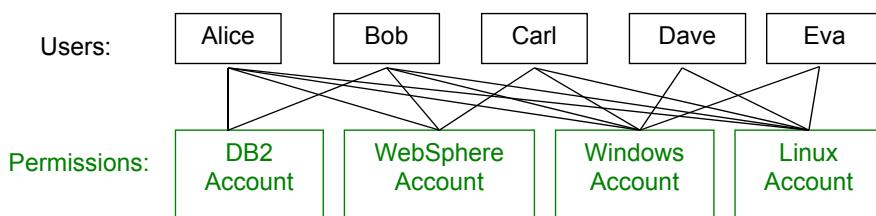
- Attribute Based AC (ABAC)
  - Overview
  - Reference architecture
- XACML
  - Language for specifying ABAC policies
  - Request/response protocol
  - Reference architecture
- OAuth 2.0
  - Key ideas
  - More details
    - Authentication Code Flow
  - Relationship to capabilities
  - Hints to OpenID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

## CONTENTS



## 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**

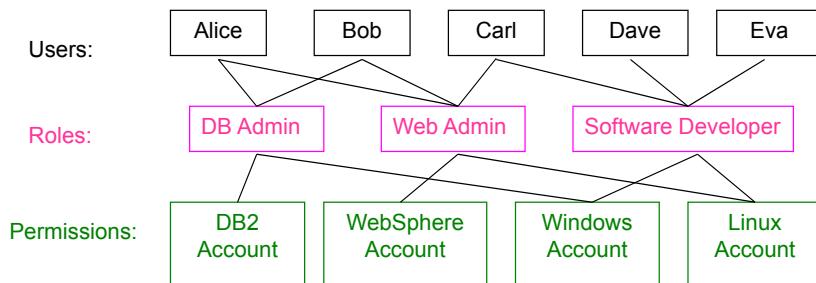


S. Ranise - Security &amp; Trust (FBK)

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



S. Ranise - Security &amp; Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

4

5

# ABAC

Attribute Based Access Control

S. Ranise - Security & Trust (FBK)

# 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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

7

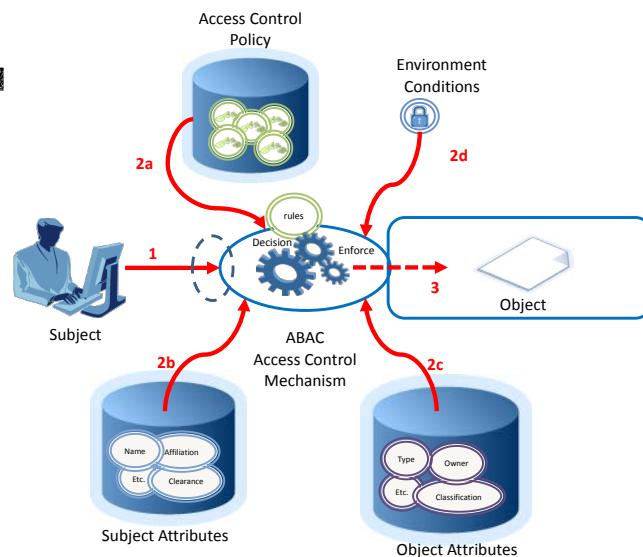
## COMPARING ABAC AND RBAC

- **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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

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
  - ...

S. Ranise - Security & Trust (FBK)

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
  - ...

S. Ranise - Security & Trust (FBK)

14

# ENVIRONMENT ATTRIBUTES

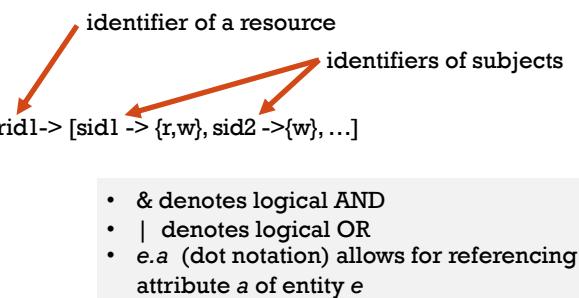
- **Describe the operational, technical, and even situational environment or context in which the information access occurs, e.g.,**
  - Current date
  - Current virus/hacker activities
  - Network security level
  - Any feature *not associated with a resource or subject*
- These attributes have been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

S. Ranise - Security & Trust (FBK)

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:  $\text{rid1} \rightarrow [\text{sid1} \rightarrow \{\text{r}, \text{w}\}, \text{sid2} \rightarrow \{\text{w}\}, \dots]$
  - add the following rules:
    - $\text{r.rid} = \text{rid1} \ \& \ \text{s.sid} = \text{id1} \ \& \ (\text{a.id} = \text{r} \mid \text{a.id} = \text{w})$
    - $\text{r.rid} = \text{rid1} \ \& \ \text{s.sid} = \text{id2} \ \& \ \text{a.id} = \text{w}$
    - ...



S. Ranise - Security & Trust (FBK)



## 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:
  - $\text{s.sid} = u \ \& \ \text{s.role} = r \ \& \ \text{r.permission} = p \ \& \ \text{r.rid} = \text{get-resource}(p) \ \& \ \text{a.id} = \text{get-action}(p)$
- We assume that a permission *p* is a pair  $(\text{rid}, \text{aid})$  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
  - $\text{get-resource}((\text{rid}, \text{aid})) = \text{rid}$
  - $\text{get-action}((\text{rid}, \text{aid})) = \text{aid}$

S. Ranise - Security & Trust (FBK)





# XACML

eXtensible Access Control Markup Language is an OASIS standard

S. Ranise - Security & Trust (FBK)

## XACML: MORE THAN A LANGUAGE FOR ABAC POLICIES

- XACML = eXtensible Access Control Markup Language 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** ...

S. Ranise - Security & Trust (FBK)



# 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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

21

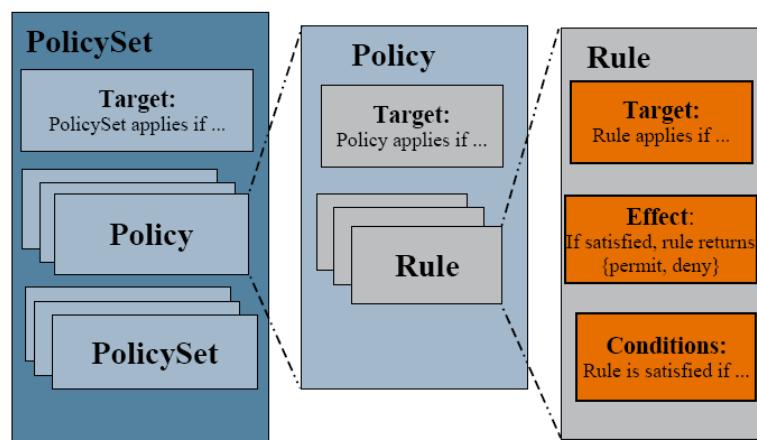
## XACML REQUESTS

- 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

S. Ranise - Security & Trust (FBK)

22

## XACML POLICY STRUCTURE



Copyright © 2007 Sun Microsystems, Inc. All rights reserved.

S. Ranise - Security & Trust (FBK)

23

# XACML POLICIES

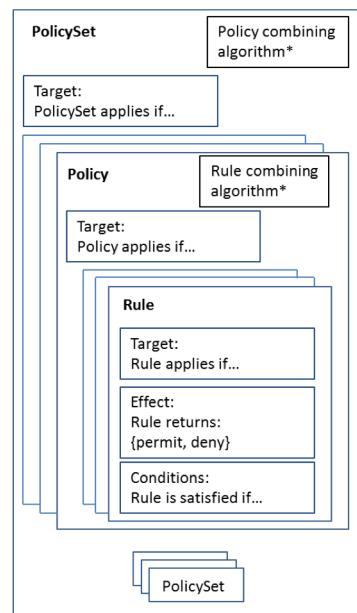
The account that follows on XACML is simplified in many respects to avoid technicalities and convey the main ideas

- 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

S. Ranise - Security & Trust (FBK)

24

# XACML POLICIES



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

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

S. Ranise - Security & Trust (FBK)

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...

S. Ranise - Security & Trust (FBK)

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
  - `PermitOverrides(rule 1, rule 2)` specifies that in case of conflict, we want to guarantee business continuity and possibly be less secure

S. Ranise - Security & Trust (FBK)

# 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

S. Ranise - Security & Trust (FBK)

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.         </Subjects>
19.       </Target>
20.     </Rule>
21.

```

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

S. Ranise - Security & Trust (FBK)

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

```

01. <Policy PolicyId="pol-0001" RuleCombiningAlgId=
02.   "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
03.   <Description>
04.     Some optional text that explains the purpose of the policy
05.   </Description>
06.   <Target>
07.     <Subjects>
08.       <Subject>
09.         <SubjectMatch MatchId=
10.           "urn:oasis:names:tc:xacml:2.0:function:string-equal">
11.             <AttributeValue DataType=
12.               "http://www.w3.org/2001/XMLSchema#string">
13.                 developer
14.               </AttributeValue>
15.             <SubjectAttributeDesignator>
16.               role
17.             </SubjectAttributeDesignator>
18.           </SubjectMatch>
19.         </Subject>
20.       </Subjects>
21.     </Target>
22.     <Rule RuleId="rul-0001" Effect="Permit"/>
23.   </Policy>

```

S. Ranise - Security &amp; Trust (FBK)

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

```

01. <PolicySet PolicySetId="pls-0001" PolicyCombiningAlgId=
02.   "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
03.   <Description>
04.     Some optional text that explains the purpose of the policy set
05.   </Description>
06.   <Target>
07.     <Subjects>
08.       <Subject>
09.         <SubjectMatch MatchId=
10.           "urn:oasis:names:tc:xacml:2.0:function:string-equal">
11.             <AttributeValue DataType="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.   <PolicyIdReference>
22.     pol-0001
23.   </PolicyIdReference>
24.   <Policy PolicyId="pol-0001" RuleCombiningAlgId=
25.     "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
26.       <Target/>
27.       <Rule RuleId="rul-0001" Effect="Permit"/>
28.     </Policy>
29.   </PolicySet>

```

S. Ranise - Security &amp; Trust (FBK)

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

```

01. <Request>
02.   <Subject>
03.     <Attribute AttributeId="role"
04.       DataType="http://www.w3.org/2001/XMLSchema#string">
05.       developer
06.     </Attribute>
07.   </Subject>
08.   <Resource>
09.     <ResourceContent>
10.       <!--
11.           In case the Resource is an XML document, it can be inlined
12.           here, so that Rules can depend on the contents.
13.       -->
14.       <example/>
15.     </ResourceContent>
16.     <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
17.       AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
18.       /some/example.xml
19.     </Attribute>
20.   </Resource>
21.   <Action>
22.     <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
23.       AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
24.       retrieve.contents
25.     </Attribute>
26.   </Action>
27.   <Environment>
28.     <Attribute DataType="http://www.w3.org/2001/XMLSchema#dateTime"
29.       AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime">
30.       2010-06-14T11:12:03Z
31.     </Attribute>
32.   </Environment>
33. </Request>

```

S. Ranise - Security &amp; Trust (FBK)

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.

```

01. <Response>
02.   <Result ResourceId="/some/example.xml">
03.     <Decision>Permit</Decision>
04.   </Result>
05. </Response>

```

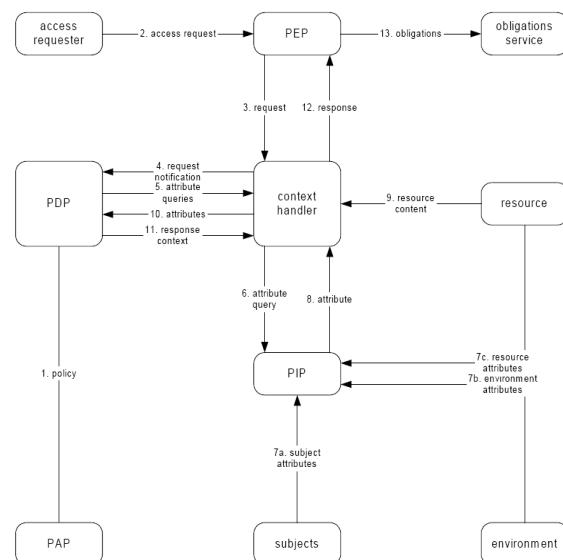
S. Ranise - Security &amp; Trust (FBK)

33

# XACML ARCHITECTURE FOR ENFORCEMENT

S. Ranise - Security &amp; Trust (FBK)

34



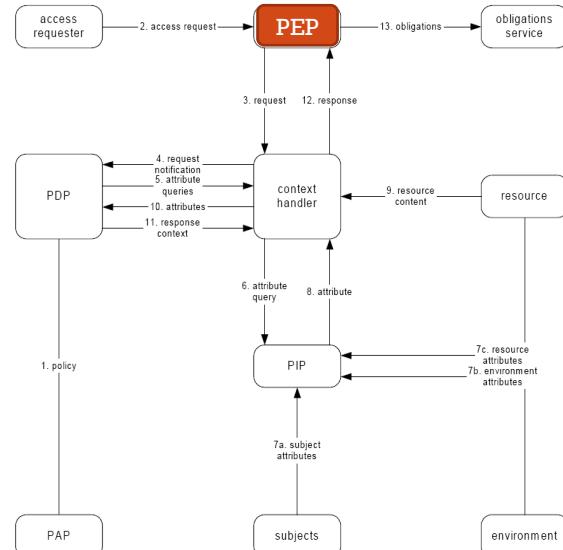
# XACML ARCHITECTURE FOR ENFORCEMENT

## Policy Enforcement Point (PEP)

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

S. Ranise - Security &amp; Trust (FBK)

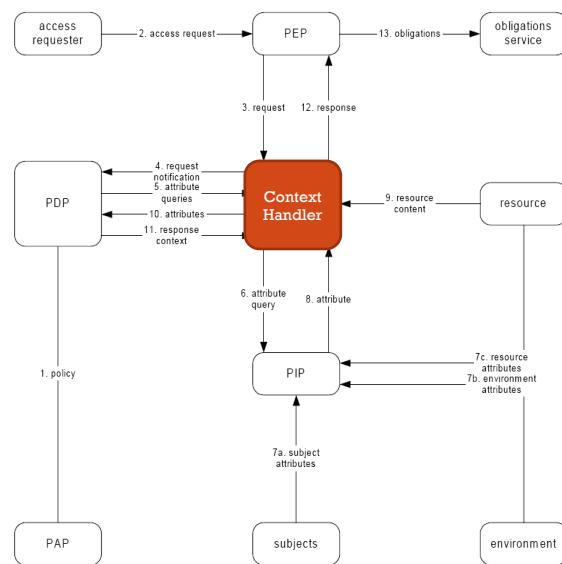
35



# 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.

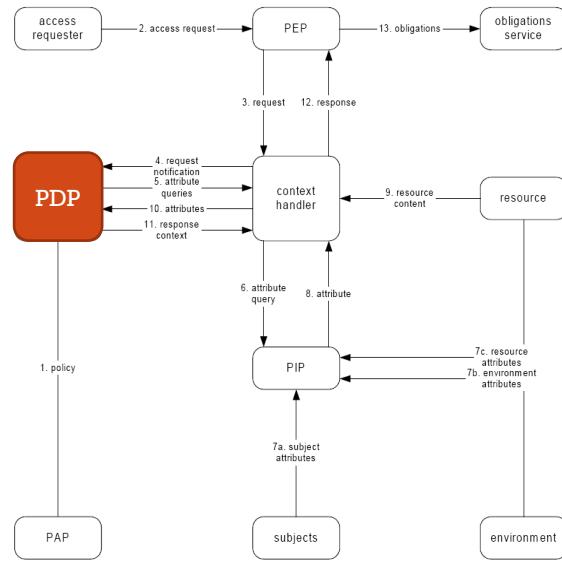


36

# 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



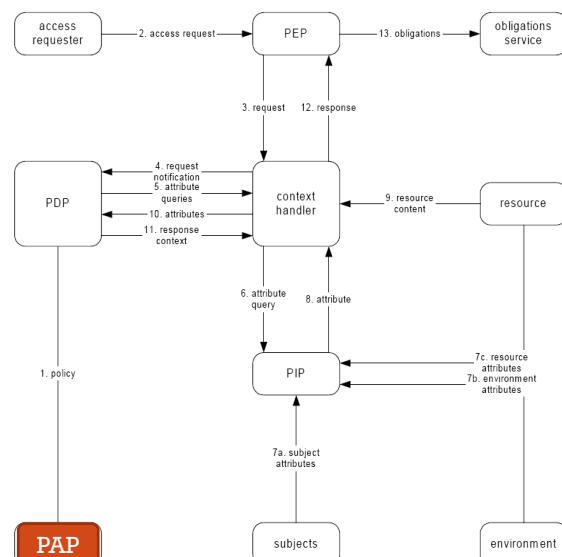
37

# XACML ARCHITECTURE FOR ENFORCEMENT

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

S. Ranise - Security &amp; Trust (FBK)

38

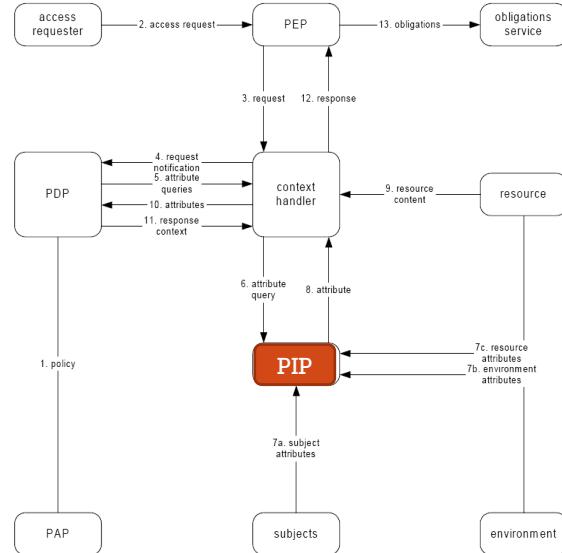


# XACML ARCHITECTURE FOR ENFORCEMENT

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

S. Ranise - Security &amp; Trust (FBK)

39



40

# OAUTH 2.0: KEY IDEAS

S. Ranise - Security &amp; Trust (FBK)

## 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**

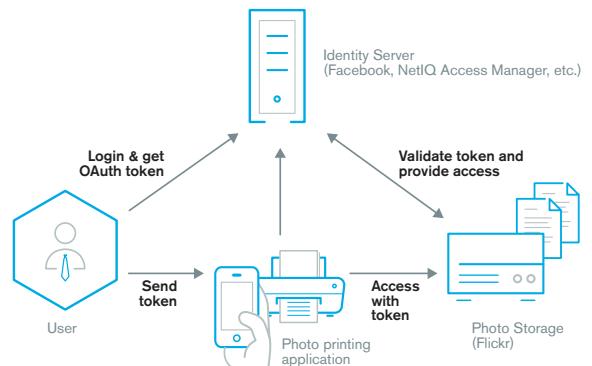


S. Ranise - Security &amp; Trust (FBK)

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)

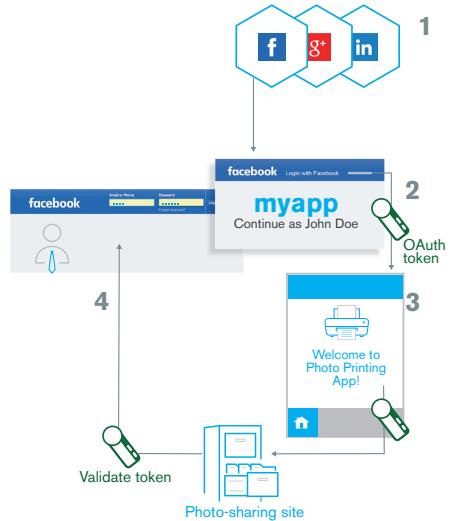


S. Ranise - Security &amp; Trust (FBK)

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)



S. Ranise - Security &amp; Trust (FBK)

43

44

# OAUTH 2.0: MORE DETAILS

S. Ranise - Security &amp; Trust (FBK)

## 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**

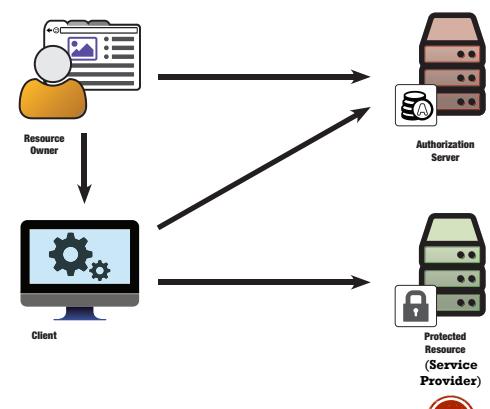
OAuth 2.0 is a *delegation protocol* that lets users allow applications to access resources on their behalf

S. Ranise - Security &amp; Trust (FBK)

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



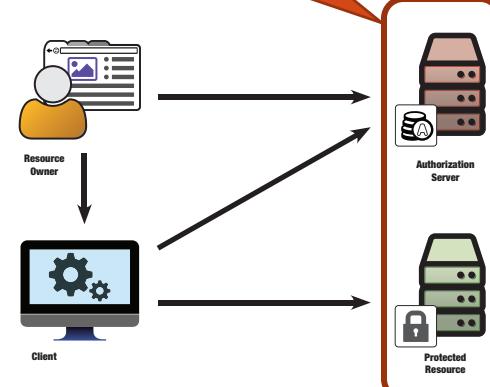
S. Ranise - Security &amp; Trust (FBK)

46

## 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

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



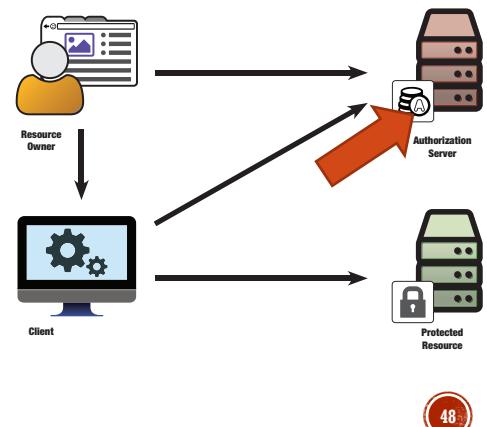
S. Ranise - Security &amp; Trust (FBK)

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

OAuth 2.0
TLS
HTTP

OpenID Connect
OAuth 2.0
TLS
HTTP

## 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)

S. Ranise - Security & Trust (FBK)

50

51

## DIGRESSION ON JWT

S. Ranise - Security & Trust (FBK)

## 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**

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

base64url encoded string: eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K

S. Ranise - Security & Trust (FBK)

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**

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

base64url encoded string: eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K

S. Ranise - Security & Trust (FBK)

- 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

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: eyBhbGcgOiBIUzI1NiwdHlwIDogSldUIH0K

- **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

S. Ranise - Security & Trust (FBK)

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

S. Ranise - Security & Trust (FBK)

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)

S. Ranise - Security & Trust (FBK)

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**

S. Ranise - Security & Trust (FBK)

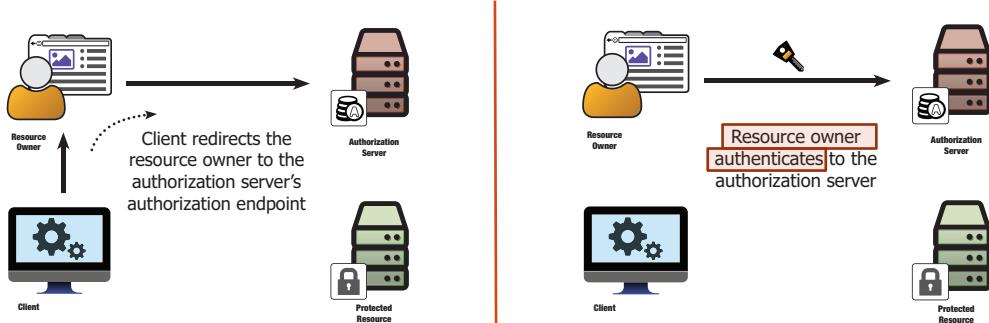
57

58

# END OF DIGRESSION ON JWT

S. Ranise - Security &amp; Trust (FBK)

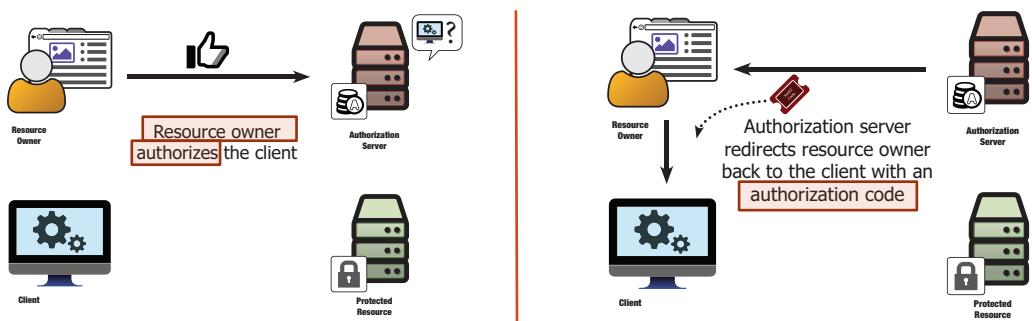
## AUTH CODE FLOW: STEP 1-2



S. Ranise - Security &amp; Trust (FBK)

59

## AUTH CODE FLOW: STEP 3-4

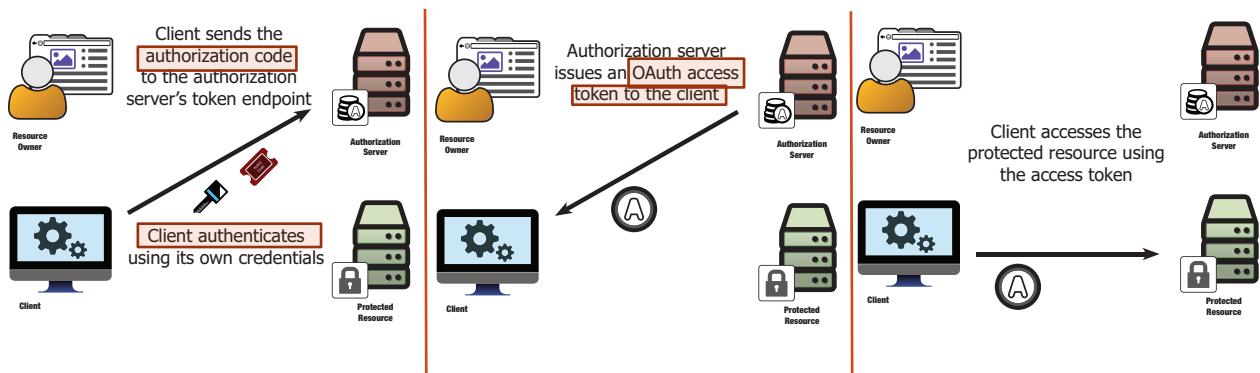


S. Ranise - Security &amp; Trust (FBK)

60

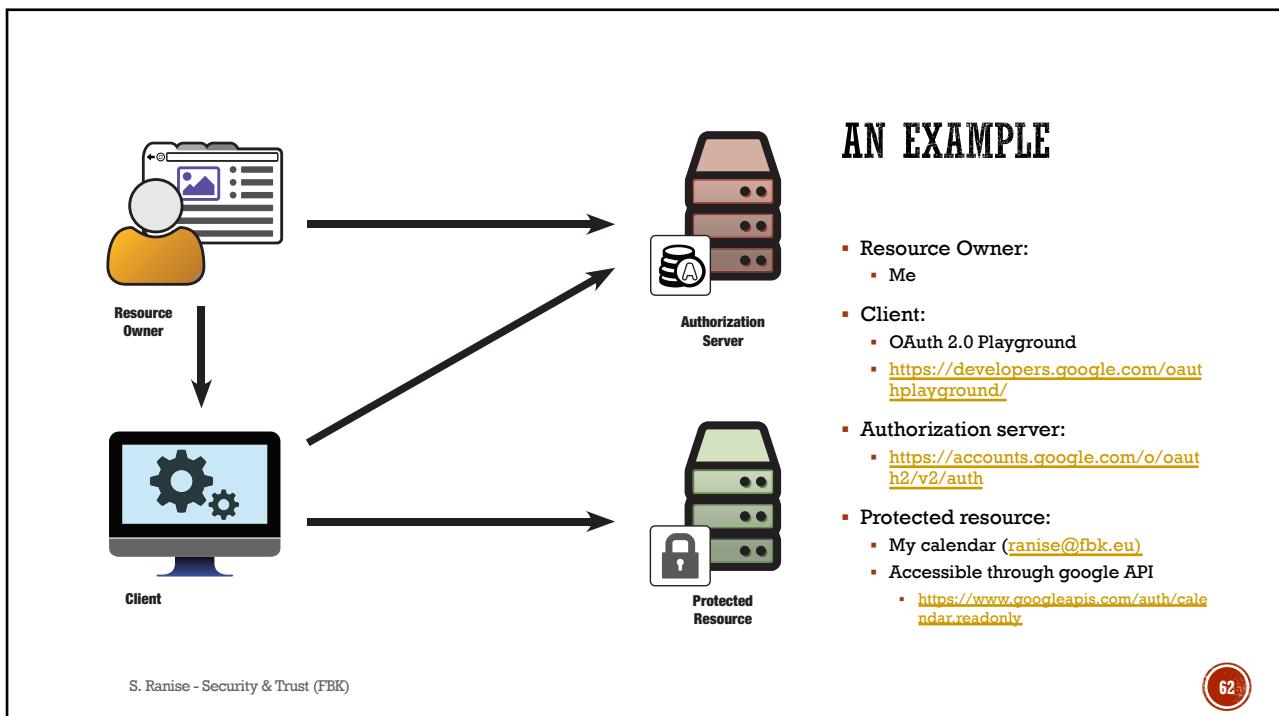
This is not yet the access token that allows the client to access the protected resource... can you guess why?

## AUTH CODE FLOW: STEP 5-7



S. Ranise - Security &amp; Trust (FBK)

61



62

**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.

**Request / Response**

No request.

**Calendar API v3**

- 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

**Authorize APIs**

**Step 2 Exchange authorization code for tokens**

**Step 3 Configure request to API**

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.

**Request / Response**

No request.

**Step 2 Exchange authorization code for tokens**

**Step 3 Configure request to API**

**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

**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](#)

**65**

S. Ranise - Security & Trust (FBK)

**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/tAFj6y124H92L8glPjPAoQHRBwwC75gI\_V-TK9a2yV

**Exchange authorization code for tokens**

Refresh token: 1/04mMhvNRlkOu0CgYIARAAGAQSNwF-L9Ir2H-uCL86D4

Access token: ya29.II-wB4v4wPsaF\_e\_J9BL2gyMy

**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/tAFj6y124H92L8glPjPAoQHRBwwC75gI_V-TK9a2yV&tosdrZMV62-774w0sgtuxm08ckRoIrg9S2K-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: GSE
Date: Fri, 09 Nov 2019 15:16:42 GMT
Content-type: application/json; charset=utf-8

{
  "access_token": "ya29.II-wB4v4wPsaF_e_J9BL2gyMykY3ddqppggaMGGyJz_T0jWobNl0Ng5YjpMsT36kXx7dXuy5EjBZTdex1C05T1UsXss28dLQvm7uIPXb6N53x50jiSLVUH6tWOP5ZPtw",
  "scope": "https://www.googleapis.com/auth/calendar.readonly",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "1/04mMhvNRlkOu0CgYIARAAGAQSNwF-L9Ir2H-uCL86D4r76m9n9RN44AqcMC3OPPajq4qlBuXEir2-03Wg7ckLWodxJCLD91YXDU"
}
```

Wrap Lines

**OAuth 2.0 Playground**

**Step 1 Select & authorize APIs**

**Step 2 Exchange authorization code for tokens**

**Step 3 Configure request to API**

Construct your HTTP request by specifying the URI, HTTP Method, headers, content type and request body.  
Then click the "Send the request" button to initiate the HTTP Request.

HTTP Method: GET Add headers 0

Request URI: ps://www.googleapis.com/calendar/v3/calendars/ranise@fbk.eu

Enter request body Content-Type: application/json

**Send the request** List possible operations

Note: The OAuth access token in Step 2 will be added to the Authorization header of the request.

**Request / Response**

```
GET /calendar/v3/calendars/ranise@fbk.eu/events HTTP/1.1
Host: www.googleapis.com
Content-length: 0
Authorization: Bearer ya29.II-wB4v4wPsaF_e_J9BL2gyMykY3ddqppggaMGGyJz_T0jWobNl0Ng5YjpMsT36kXx7dXuy5EjBZTdex1C05T1UsXss28dLQvm7uIPXb6N53x50jiSLVUH6tWOP5ZPtw

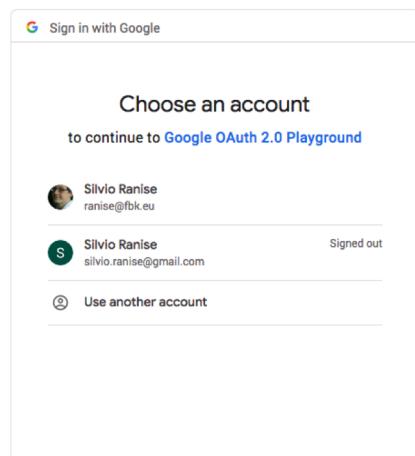
HTTP/1.1 200 OK
Content-length: 542136
X-xss-protection: 1; mode=block
Content-location: https://www.googleapis.com/calendar/v3/calendars/ranise@fbk.eu/events
X-content-type-options: nosniff
Transfer-encoding: chunked
Expires: Fri, 09 Nov 2019 15:21:16 GMT
Vary: Origin, X-Origin
Server: GSE
Content-encoding: gzip
Cache-control: private, max-age=0, must-revalidate, no-transform
Date: Fri, 08 Nov 2019 15:21:16 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

{
  "nextPageToken": "CkkKO182MHJqGN0zZncGp1yjlqNzBvNGNi0Ws2c3FrY2I5cDc0bzNlYmE2NjhvMzBjaTI2aDFqZ2MyNjc0GA
EggIDA8o0YtpgVg01ABIAgMD-7srw2uUC",
  "kind": "calendar#events",
  "defaultReminders": [
    {
      "minutes": 10,
      "method": "popup"
    }
  ],
  "items": [
    {
      "status": "confirmed",
      "kind": "calendar#event",
      "end": {
        "date": "2012-12-04"
      },
      ...
    }
  ]
}
```

Wrap Lines

## OAUTH 2.0: KEY POINTS (1)

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

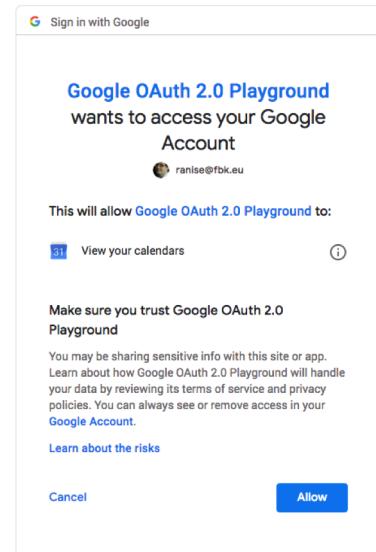


S. Ranise - Security & Trust (FBK)

68

## 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
  - **Type of access**
    - User is offline, limited number of accesses



S. Ranise - Security & Trust (FBK)

69

## 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!

The screenshot shows the OAuth 2.0 Playground interface. On the left, there are three steps: Step 1 (Select & authorize APIs), Step 2 (Exchange authorization code for tokens), and Step 3 (Configure request to API). Step 3 is active, showing a configuration for a GET request to `https://www.googleapis.com/calendar/v3/calendars/ranise@fbk.eu/events`. The request body contains the OAuth token obtained in Step 2. The response on the right shows a JSON object representing a list of events, with one event detailed below:

```

{
  "start": {
    "date": "2012-12-04T15:00:00+01:00"
  },
  "end": {
    "date": "2012-12-04T16:00:00+01:00"
  },
  "summary": "Test event"
}
  
```

S. Ranise - Security & Trust (FBK)

70

## 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

S. Ranise - Security & Trust (FBK)

71

## 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

### 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/IAFj6y124H92L8gliPjPAoQHRBwwC75gI_V-TK9a2yV`

**Exchange authorization code for tokens**

Refresh token: `1//04mMhvNRlkOu0CgYIARAAGAQSNwF-L9lr2H-uCL86D4`

Access token: `ya29.II-wB4v4wPsaF_e_J9BL2gyMy` **Refresh access token**

Auto-refresh the token before it expires.

The access token will expire in **3514** seconds.

S. Ranise - Security & Trust (FBK)

72

## 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**

S. Ranise - Security & Trust (FBK)

73

74

# USING OAUTH 2.0 FOR AUTHENTICATION?

It turns out to be a bad idea...

S. Ranise - Security & Trust (FBK)

## 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

S. Ranise - Security & Trust (FBK)

75

# OPENID CONNECT

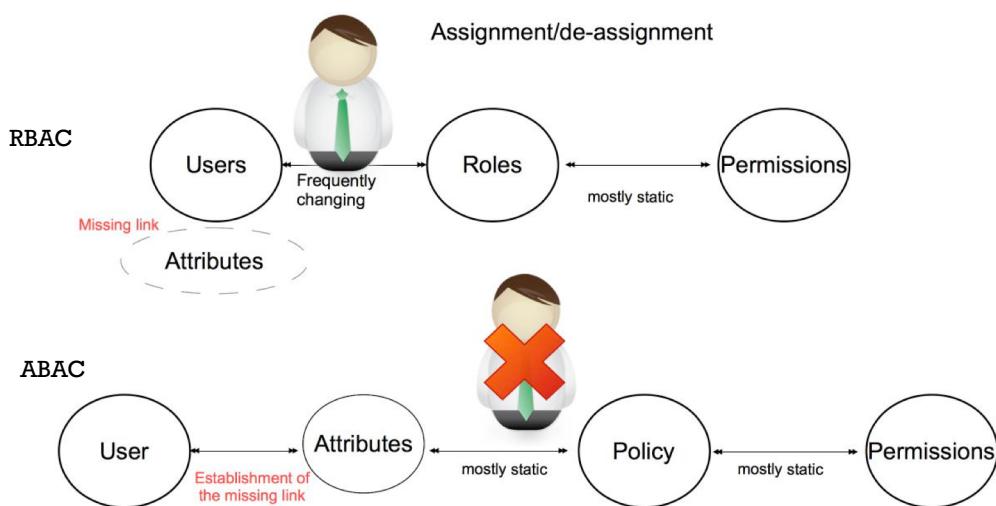
<https://openid.net/connect/>

- 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...

S. Ranise - Security & Trust (FBK)

76

# TAKEAWAYS: ABAC

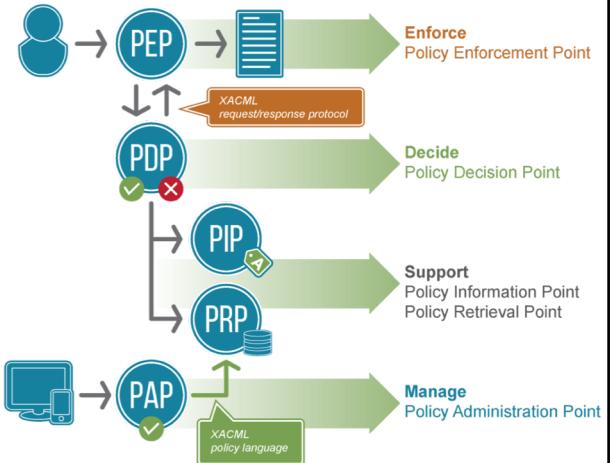


S. Ranise - Security & Trust (FBK)

77

## 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**



S. Ranise - Security & Trust (FBK)

78

## 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

S. Ranise - Security & Trust (FBK)

79

## 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?

S. Ranise - Security & Trust (FBK)

80