Security and Cryptography 2022

X. ACCESS CONTROL

Mirosław Kutyłowski

ACCESS CONTROL (AC)

- decision whether a subject (user, etc) is allowed to carry out a specific action (operation) on an object (resource)
- policy = set of rules of (axioms + derivation rules) for making the decision
- model: there are
  - Principals: users, processes, etc.
  - Operations performed by Principals: read, write, append, execute, delete, delegate...
  - Objects: the objects on which the principals perform operations: files, hardware execution (opening door lock)...

- **goals**:
  - availability (false rejection might be a disaster)
  - no unauthorized access
  - simplicity (users must understand what is going on, complicated systems are prone to be faulty)

- **practical problems:**
  - **diversity of application scenarios**, no unique generalizations
  - **no trusted party:** administrator/system provider/product provider/user might be malicoius
  - **safety** issues (access control to industrial SCADA systems, ...)
  - **future:** constraints must respect future states
  - **no test environment:** designing a system much harder: hardware, software design are much easier to test, in access control one cannot create a complete environment for testing purposes
  - **cultural differences**: the users may react differently $\Rightarrow$ severe misunderstanding of the situation by a AC designer

Formal description

- **goal:** define secure and insecure states:
  - **secure state** is such that the principals get access to objects as claimed by some predefined axiomatic properties,
  - **insecure states**: when the things go explicitly wrong
  - **gray zone:** states that are neither secure nor insecure: AC policy violated but still no catastrophy

- AC defines rules define **a dynamic system** – changing access rights by principals
- **basic security question:**

  given: a secure state

  safety problem: is it possible to change the state to an insecure one through a number of moves that are admissible according to a given AC policy

Undecidability Theorem

the problem whether from a given secure state we can reach an insecure state is **undecidable**

Proof idea

- a Turing machine can be regarded as an access control system:
  - a head may be treated as a principal
  - the tape represents objects (each entry is a data)
  - transition function of TM specifies what is allowed concerning data and security status
- ask if a Turing machine can reach some state+tape contents (which is declared as insecure)
- the problem is in general undecidable

□

Consequences

It is necessary to restrict the freedom to define Access Control systems to avoid undecidability issues.

It is hard as many rewriting systems are equivalent to Turing machines (i.e. Post Correspondence Problem).

- restrictions must be strong in order to get low complexity of formal safety
- too strict restrictions make the system less flexible and hard to use

Main Models

- approaches:
  - discretionary – the objects owner defines the access rules
  - mandatory – the system takes care
- main models:
  - Access Control Matrix or Access Matrix (AM)

- Access Control List (ACL)
- Role-Based Access Control (RBAC)
- Attribute-Based Access Control (ABAC)
- lattice models

- main decisions:
  - principals: **least priviledge** given to achieve what is needed
  - objects: options are A) explicit permissions needed

    B) admitted unless prohibited
  - rule administration: this is the **core part** of the system

AM (Access Control Matrix)

- Objects, Subjects , Access function
- Access Function depicted as a Matrix: each entry defines allowed operations

  (e.g. read, write, delete, execute, ...)

application areas: database systems, operating systems, ...

the most common case: AC of apps in Android, IoS, ...

problems of AM:

i. lack of scalability, no flexibility

ii. enormous effort, unless an AC system is tiny

iii. central administration, single point of failure (attack the administrator to get unrestricted control over the system / confuse the user of an app)

ACL (Access Control List)

- object centric: for each object a list of admitted principals
- there are file systems based on ACL, cloud
- relational database systems, SQL systems, network administration
- door locks systems

advantages:

close to business models, simple, easy to implement in small and highly distributed systems

problem:

i. **difficult management of principals** (blacklisting a certain principal means scanning all ACL's)

ii. **poor scalability** unless the number of prinicipals is small and certain "locality" properties

iii. **safety problems:** lack of coordination between ACL may lead to problems

so ACL is not much useful for large and complicated systems

Kerberos - System based on ACL concept

Kerberos:

- early system based on tickets:
  1. user authentication against a central server, service request
  2. cryptographic ticket issued for the user and a given service
  3. ticket presented by the user to a server
- symmetric crypto used: secret shared between the central server and the service servers

SPKI: Simple PKI

- certificates issued by objects, certificate confirms a public key
- private key based authentication
- delegation of access rights
- local naming with the link to a public key
- concepts: short time certificates, "certificate of health", "suicide note"

RBAC (Role-Based Access Control)

- **organization**:
  - *subjects* assigned to *roles*
  - *permissions* assigned to *roles*

  in **matrix representations**
- **textbook example**: a hospital system with differents roles: doctor, nurse, patient, accounting staff, family members of a patient, insurance company controller, ... :
  i. a patient may read own record but not write or erase it
  ii. a doctor can write/read a record (but not erase)
  iii. a nurse can write records specifying activities, but not diagnosis part (reserved for the doctor's role)
  iv. ...

  Problems: even in this example we get into troubles: how to meet the requirements of GDPR: how to formalize "patient's own medical record"? Impossible with RBAC

RBAC1 standard

- hierarchy of roles (a directed acyclic graph)
- a role gets all permissions from its lower roles in the hierarchy,

advantages: compact description, logic of the system follows a strict military fashion

disadvantages: expressive power the same as for RBAC

RBAC2 standard: RBAC0+constraints for RBAC matrices:

- **mutual exclusion**: a user can get at most one role in a set, permission can be granted to only one role in a set, ...
- **cardinality**: the maximum number of subjects with a given role might be set
- **prerequisitives**: necessary to have role A before getting role B

- **RBAC3**=RBAC1 consolidated with RBAC2

advantages of RBAC:

- simple management of users
- suitable for large systems with a simple "military" architecture

problems of RBAC:

- no fine tuning of access rights
  - all users with the same role have the same permissions
- problems with dynamic changes
  - splitting roles?
  - adjustments must be global
- limited use in large global systems
  - the global assignment of roles to users is an utopia

LATTICE models:

many security levels organized in a directed acyclic graph

Confidentiality policy: Bell-LaPadula model

information reporting like in an army:

- **read-down:** read allowed only if the subject's security level dominates the object's security level
- **write-up:** write allowed only if the subject's security level is dominated by the object's security level (write-up)
- **tranquility property**: changing the security level of an object concurrent to its use is forbidden

Integrity Policy: Biba model

- dual to BLP, like assigning orders in an army
- write-down, read-up

Ring model

- a version of Biba model for an operating system, focus on integrity
- reading up is allowed, writing down is allowed. But within a specified ring (a, b) (between layers a and b)

Denning's Lattice Model

- a general model for information flow
- a **partial order** of different levels, not always a linear order
- **lub operation**: least upper bound of two nodes
- takes into account **implicit and explicit data flows**
- **static** or **dynamic binding to levels**

Explicit and implicit data flows

- **explicit:** e.g. via an assignment $b := a$
- **implicit:** via conditional: $\text{if } a = 0 \quad \text{then} \quad b := c$

  information on a is leaked via a change/no change in b
- programs considered composed of assignments via composing sequences and conditionals

Security rules

- explicit flow only if allowed by rules of information flows

  $b := a$ if data from level of a can be written in the the level of b
- for a sequence of assignments: each assignment must be secure
- a conditional is secure if
  - the flows in the body are secure
  - the implicit flow by conditional is secure

    $\text{if } a = 0 \quad \text{then} \quad b := c$ is secure if explicit flow $b := c$ is secure as well as the flow from a to b is secure

Binding to levels

- **static**: binding to levels is fixed (Biba, BLP, ...)
  - security checked during runtime
  - Data Mark Machine: mark data in the program, mark of a depends on its location
- **dynamic:**
  - huge problems with implicit data flows – the fact that a is at level L leaks something about a
  - **High Water Mark** technique for BLP:

    after executing $b := a$ the level of b is raised to the least upper bound of a and b
  - **Low Water Mark** for Biba:

    after executing $b := a$ the level of b is lowered to the highest lower bound of a and b

Decentralized Label Model

- any entity has a list of labels
- operation allowed if the lists of subject and object intersect
- semantics:
  - constructions such as “if appropriate” for exceptions

Chinese Wall (the name is misleading)

- focused on conflicts of interest, for a company that serves competing partners
- Conflict of Interest Classes (CIC) – from each class access to ≤ 1 dataset allowed
- In each CIC some number of datasets, datasets belong to companies
- each object in some dataset
- an extra CIC of one dataset of sanitized objects (data freely available)

Rules for allowed access

- S can **read** O only if
  - S has already read an object from the same dataset as O, or
  - S has not read any object from the CIC containing O

- **writing more complicated:**

  user A has access to dataset CompanyA in CIC x_1,

  user B has access to dataset CompanyB in CIC x_1,

  both have access to dataset BankA in CIC x_2.

  $\Rightarrow$ unconstrained writing in objects from BankA would enable leakage from CompanyA to CompanyB

- S can write into O only if
  - conditions for read fullfilled ("simple security rule"), and
  - "no object can be read which is in a different company dataset to the one for which write access is requested" - consequence is that no writing if a subject has read datasets of two companies

- an axiomatized approach enabling formal verification is possible

AC for group of principals

- examples:
  - exporting a secret key from HSM and enabling reconstruction iff k shares are used
  - access granted when a smart card and a smartphone of a user are involved in identification
- in general:
  - access granted iff a certain number of subjects involved simultaneously
  - the simplest scenario: threshold scheme – at least k out of n in a group request an access

General case: *access structure:*

$\rightarrow$ a family $\mathcal{F}$ of subsets of a group which is an ideal: if $A \in \mathcal{F}$ and $A \subset B$, then $B \in \mathcal{F}$

$\rightarrow$ it might be difficult to provide a compact description of an ideal, however the practical cases should enable formulation as a simple Boolean formula using $\wedge$ and $\vee$ operators (and no negation $\neg$)

$\rightarrow$ given such a formula it is easy to create an access system based on secret sharing and authentication with the key reconstructed from secret sharing:

i. $\wedge$ corresponds to secret sharing based on XOR

ii. $\vee$ corresponds to Shamir's scheme 1 out of k based on polynomial of degree 1

ABAC – Attribute Based Access Control

two approaches are covered by this lecture:

- XACML (XML Access control language)
- NGAC (New Generation Access Control)

both based on attribute based control

Challenges:

– how to design the system architecture?

– which logic? which evaluation mechanism?

OASIS specification with some examples:

see http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf

not covered by a patent

open source implementations available; Sun XACML (with all components)

from 2003, evolution

main issue addressed: compliance

XACML concept:

- **language**: – standard: eXtensible Access Control Markup Language (XACML)
- **policies:**
  - **rule** (stating when to allow or deny access) → **policy** as a set of rules → **policy sets** composed of policies
  - policies might be distributed, PIP (policy information point) keeps track where to find policies
  - unique policy names within one PIP
  - **policy evaluation engine** provides an **automatic evaluation** of a request based on policy sets and attributes
    - — the engine might be very complex, it is a proof system in a resticted logic

– **policy evaluation:** process answering access queries, computation based on entities, attributes and current policies – ADF (access decision function):

i. **identify policies** applicable to the current request,

ii. the **request evaluated against each rule** from these policies – it requires **evaluation of attributes and pattern matching**,

iii. the results are **combined within a policy**,

iv. the results **combined between policies**,

v. a decision revealed together with **obligations** (compulsory) and **advices** (optional to follow)

- **subjects:**
  - entities asking for access
- **attributes:**
  - e.g. name, ID, network domain, email address
  - numeric: age, and other multivalued
- **resource:**
  - actions on resources to be denied or permited (or left undecided)
- **data form:** tree structures (or forest) typical for XML
- **environment:**
  - it provides a set of logical and mathematical operators on attributes of the subject, resource and environment.

– **recommended architecture:**

→ **Policy Enforcement Point** (PEP) – handles different syntax from external systems (not everything in XACML, so a translation might be needed)

→ **Policy Information Poin**t (PIP) – responsible for gathering data on attributes, environment,

→ **Policy Administration Point** (PAP) - provides to PDP everything concerning policies

→ **Policy Decision Point** (PDP) - makes actual evaluation

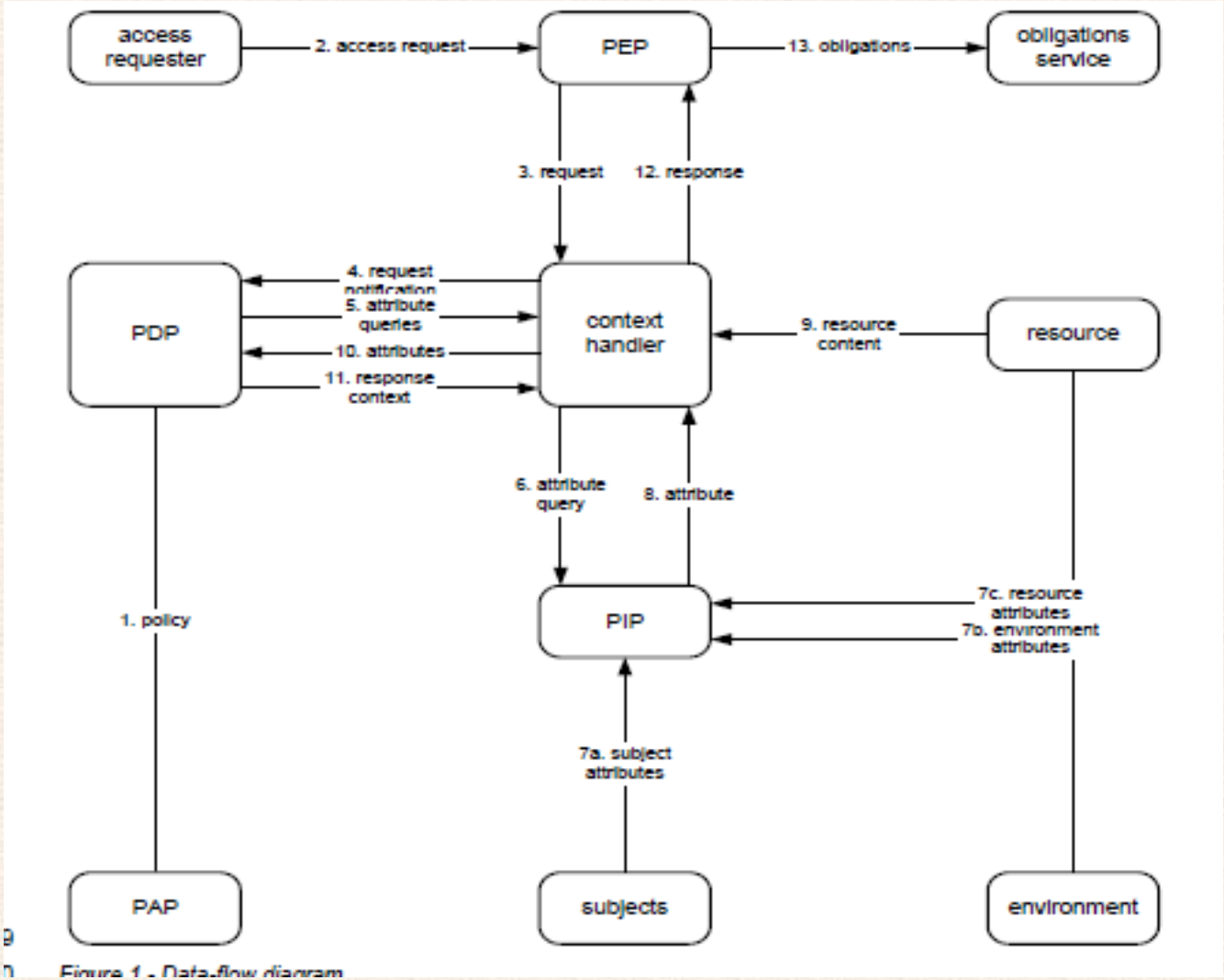


Figure 1 - Data-flow diagram

(figure from the ABAC manual)

steps :

- access request goes to PEP
- access request goes to context handler
- request for attributes value to PIP
- PIP gets attributes evaluation for environment, resources and subjects
- attributes sent to context handler
- data on resources from resources to context handler
- everything to PDP
- response from PDP with a decision
- response from context handler to PEP
- obligations from PEP to obligations service (not really within the standard what and how to do)

major problems:

- creating policies is a complicated task, errors are likely
- trade-off between expresiveness and complexity of evaluation

Rule

components: target, effect, condition, obligation, advice

- **target:** defines where the rule should be applied

  a) **if no target** given in the rule, then the (more general) target from the policy applies

  b) subject, resource and action defined by target **should match** the subject, resource and action from the request

c) **if not matched**, then the rule is not applicable

d) **shortcuts**, e.g.: <AnySubject/>, <AnyResource/>, <AnyAction/>

e) **subjects** interpreted as a subtree starting in a node;

- matching of the whole tree

f) **resources**: different **options** for matching:

i. the contents of the identified node only,

ii. the contents of the identified node and the contents of its immediate child nodes

iii. the contents of the identified node and all its descendant nodes

g) **XML pointers** may be used, values fetched during evaluation

Further components of a rule

- **effect:** "Permit" or "Deny" (to be applied if the rule is applicable)
- **condition:** Boolean expression to be satisfied (so in order to apply the rule we must have both matching with the target and condition evaluated to `true`)
- **obligation expression:** it is mandatory to be fulfilled by PEP
  - e.g. when giving access to an account in online banking to send an SMS to the client
- **advice expression:** may be ignored by PEP

Important feature of ABAC

- the rules need not to be consistent with each other
- it will be the job of a policy to resolve the conflicts

Policy components:

- **target:** one of more patterns, different approaches for combining definitions of targets (must be in one of targets from policy rules or in all targets of the rules)
- **rule-combining algorithm identifier** determines how to combine the rules' effects (e.g. all rules MUST have the effect Permit, or at least one Permit), some algorithms are predefined but one can define own ones
- **set of rules**
- **obligation expressions** and **advice expressions**

decisions of a policy:

- `permit`
- `deny`
- `not applicable`
- `indeterminate`

access granted only if **"permit" and obligations fulfilled**

attributes evaluation

a few flexible options:

- string matching algorithms
- XPath selection in an XML structure
- result might be *a bag* of attributes - all matchings according to the selection specified

```
<Rule   RuleId= "urn:oasis:names:tc:xacml:3.0:example:SimpleRule1"   Effect="Permit">
 <Description>
 Any subject with an e-mail name in the med.example.com domain can perform any action on any resource.
</Description>
<Target>
   <AnyOf>
      <AllOf>
            <Match   MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
            <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#string">
                med.example.com</AttributeValue>
            <AttributeDesignator
              MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
              AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
              DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
             </Match>
              ...
     </AllOf>
        ...
```

</AnyOf>
</Target>
</Rule>

target evaluation (warning –somewhat confusing!)

- **AnyOf** : result="match" if all options "match", if one option gives "no match", then result="no match", otherwise result="indeterminate"
- **AllOf:** result="match" if at least one option is "match', if all options are "no match" then result="no match", otherwise result="indeterminate"
- **Match:** all components must be matched

rule evaluation:

target:	**condition:**	**rule value:**
match or no target	true	effect
match or no target	false	not applicable
match or no target	indeterminite	indeterminate
no match	*	not applicable
indeterminite	*	not applicable

Table 1.

policy evaluation:

target:	rule value:	policy value:
match	at least one rule with Effect	according to rule-combining algorithm
match	all rules with NotApplicable	not applicable
match	at least one rule Indeterminite	according to rule-combining algorithm
no match	*	not applicable
indeterminite	*	indeterminite

standard rule combining algorithms:

a) deny-overrides

b) permit-overrides

c) first-applicable

d) only-one-applicable

security issues:

i. replay attack – works unless some freshness safeguards

ii. message insertion – it may create a lot of harm, message authentication needed

iii. message deletion – the system should prevent permitting access if some message undelivered

iv. message modification - obviously one could change the decision (authentication required)

v. NotApplicable - dangerous since sometimes automatically converted to Permit (webservers...)

vi. negative rules - not always effective: in some cases evaluation result is undeterminite, so there is no False and access is granted

vii. DoS - via loops in evaluating policies

Constructing a policy

Rule 1: A patient may read records concerning him or her.

Rule 2: The same concerns a parent or guardian if a patient is under 16 years of age.

Rule 3: The designated primary care physician can write, provided that an email is sent to the patient.

Rule 4: An administrator shall not be permitted to read or write any record.

How to express it as a policy? One should be careful:

- how to check that the record corresponds to the requester?
- what about a parent? How to check the current date? (no access in the day of 16th birthday and later), can a child enter own medical record?
- what if the administrator gets sick? Then Rule 1 and 4 are contradictory
- can a doctor be an administrator?

Rule 1

```
[1] <?xml version="1.0" encoding="UTF-8"?>
[2] <Policy
[3] xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
[4] xmlns:xacml ="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
[5] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[6] xmlns:md="http://www.med.example.com/schemas/record.xsd"
[7] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:1"
[8] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"
        Version="1.0">
[10] <PolicyDefaults>
[11] <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>
[12] </PolicyDefaults>
[13] <Target/>
```

```
[14] <VariableDefinition VariableId="17590034">
[15] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[16] <Apply
[17] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[18] <AttributeDesignator
[19] MustBePresent="false"
[20] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
[21] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:patient-number"
[22] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[23] </Apply>
[24] <Apply
[25] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[26] <AttributeSelector
[27] MustBePresent="false"
[28] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[29] Path="md:record/md:patient/md:patient-number/text()"
[30] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[31] </Apply>
[32] </Apply>
[33] </VariableDefinition>
```

[34] <Rule

[35] RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"

[36] Effect="Permit">

[37] <Description>

[38] A person may read any medical record in the

http://www.med.example.com/schemas/record.xsd namespace

for which he or she is the designated patient

[41] </Description>

```
[42] <Target>
[43] <AnyOf>
[44] <AllOf>
[45] <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
[46] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
[47] >urn:example:med:schemas:record</AttributeValue>
[48] <AttributeDesignator
[49] MustBePresent="false"
[50] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[51] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
[52] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
 [53] </Match>
```

```
[54] <Match
[55] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
[56] <AttributeValue
[57] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
[58] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[59] >md:record</AttributeValue>
[60] <AttributeDesignator
[61] MustBePresent="false"
[62] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[63] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
[64] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
[65] </Match>
[66] </AllOf>
[67] </AnyOf>
```

```
[68] <AnyOf>
[69] <AllOf>
[70] <Match
[71] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[72] <AttributeValue
[73] DataType="http://www.w3.org/2001/XMLSchema#string"
[74] >read</AttributeValue>
[75] <AttributeDesignator
[76] MustBePresent="false"
[77] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
[78] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
[79] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[80] </Match>
[81] </AllOf>
[82] </AnyOf>
[83] </Target>
[84] <Condition>
[85] <VariableReference VariableId="17590034"/>
[86] </Condition>
[87] </Rule>
[88] </Policy>
```

Rule 2

[1] <?xml version="1.0" encoding="UTF-8"?>

[2] <Policy

[3] xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"

[4] xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"

[5] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

[6] xmlns:xf="http://www.w3.org/2005/xpath-functions"

[7] xmlns:md="http:www.med.example.com/schemas/record.xsd"

[8] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"

[9] Version="1.0"

[10] RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining- algorithm:deny-overrides">

[11] <PolicyDefaults>

[12] <XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</XPathVersion>

[13] </PolicyDefaults>

[14] <Target/>

```
[15] <VariableDefinition VariableId="17590035">
[16] <Apply
[17] FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-less-or-equal">
[18] <Apply
[19] FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
[20] <AttributeDesignator
[21] MustBePresent="false"
[22] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
[23] AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
[24] DataType="http://www.w3.org/2001/XMLSchema#date"/>
[25] </Apply>
[26] <Apply
[27] FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-add-yearMonthDuration">
```

```
[28]       <Apply
[29]         FunctionId="urn:oasis:names:tc:xacml:1.0:function:date-one-and-only">
[30]         <AttributeSelector
[31]           MustBePresent="false"
[32]           Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[33]           Path="md:record/md:patient/md:patientDoB/text()"
[34]           DataType="http://www.w3.org/2001/XMLSchema#date"/>
[35]         </Apply>
[36] <AttributeValue
[37] DataType="http://www.w3.org/2001/XMLSchema#yearMonthDuration"
[38] >P16Y</AttributeValue>
[39] </Apply>
[40] </Apply>
[41] </VariableDefinition>
```

```
[42] <Rule
[43] RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
[44] Effect="Permit">
[45] <Description>
[46] A person may read any medical record in the
[47] http://www.med.example.com/records.xsd namespace
[48] for which he or she is the designated parent or guardian,
[49] and for which the patient is under 16 years of age
[50] </Description>
```

```
[51] <Target>
[52] <AnyOf>
[53] <AllOf>
[54] <Match
[55] MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
[56] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI"
[57] >urn:example:med:schemas:record</AttributeValue>
[58] <AttributeDesignator
[59] MustBePresent="false"
[60] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[61] AttributeId= "urn:oasis:names:tc:xacml:2.0:resource:target-namespace"
[62] DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
[63] </Match>
[64] <Match
[65] MatchId="urn:oasis:names:tc:xacml:3.0:function:xpath-node-match">
[66] <AttributeValue
[67] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"
[68] XPathCategory="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[69] >md:record</AttributeValue>
```

```
[70] <AttributeDesignator
[71] MustBePresent="false"
[72] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[73] AttributeId="urn:oasis:names:tc:xacml:3.0:content-selector"
[74] DataType="urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"/>
[75] </Match>
[76] </AllOf>
[77] </AnyOf>
```

```
[78] <AnyOf>
[79] <AllOf>
[80] <Match
[81] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[82] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
[83] >read</AttributeValue>
[84] <AttributeDesignator
[85] MustBePresent="false"
[86] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
[87] AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
[88] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[89] </Match>
[90] </AllOf>
[91] </AnyOf>
[92] </Target>
```

```
[93] <Condition>
[94] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
[95] <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[96] <Apply
[97] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[98] <AttributeDesignator
[99] MustBePresent="false"
[100] Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
[101] AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:parent-guardian-id"
[102] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[103] </Apply>
[104] <Apply
```

```
[105] FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
[106] <AttributeSelector
[107] MustBePresent="false"
[108] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
[109] Path="md:record/md:parentGuardian/md:parentGuardianId/text()"
[110] DataType="http://www.w3.org/2001/XMLSchema#string"/>
[111] </Apply>
[112] </Apply>
[113] <VariableReference VariableId="17590035"/>
[114] </Apply>
[115] </Condition>
[116] </Rule>
[117] </Policy>
```

Policy Set

```
[1] <?xml version="1.0" encoding="UTF-8"?>
[2] <PolicySet
[3] xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
[4] xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
[5] PolicySetId="urn:oasis:names:tc:xacml:3.0:example:policysetid:1"
[6] Version="1.0"
[7] PolicyCombiningAlgId=
[8] "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:deny-overrides">
[9] <Description>
[10] Example policy set.
[11] </Description>
[12] <Target>
[13] <AnyOf>
[14] <AllOf>
[15] <Match
[16] MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
[17] <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
[18] >urn:example:med:schema:records</AttributeValue>
```

[19] <AttributeDesignator

[20] MustBePresent="false"

[21] Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"

[22] AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target-namespace"

[23] DataType="http://www.w3.org/2001/XMLSchema#string"/>

[24] </Match>

[25] </AllOf>

[26] </AnyOf>

[27] </Target>

[28] <PolicyIdReference>

[29] urn:oasis:names:tc:xacml:3.0:example:policyid:3

[30] </PolicyIdReference>

[31] <Policy

[32] PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:2"

[33] RuleCombiningAlgId=

[34] "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides"

[35] Version="1.0">

[36] <Target/>

[37] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:1"

```
[38] Effect="Permit">
[39] </Rule>
[40] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:2"
[41] Effect="Permit">
[42] </Rule>
[43] <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:4"
[44] Effect="Deny">
[45] </Rule>
[46] </Policy>
[47] </PolicySet>
```

Administration of Policies

- single Issuer (Trusted)
- multiple issuers:
  - concept of delegation
  - delegation chain: like SPKI
  - access policies (resolve access rights) and administration policies (delegate rights)
  - policies of the Issuer (root) “trusted policies”, remaining ones: “untrusted policies”

Rules of delegation

- the right of further delegation must be explicitly given
- *situation* of the delegated policy must be a subset (not necessarily a proper one) of the *situation* in the delegating policy

  where: situation is the set of attribute tuples where the policy is applicable

Evaluation Principles

- find all access policies that are applicable
- for each untrusted policy reconstruct the delegation chain, disregard if the chain is invalid
- apply rules for policy set

NGAC

- initial ANSI standard 2013, result of NIST projects
- provisional patent
- Github open source reference distribution
- based on standardized and generic set of relations and functions to be reused in policies

a clear description of ideas (for your convenience, examples used directly in this lecture):

https://csrc.nist.gov/publications/detail/sp/800-178/final

NGAC

user, operation, object

instead of

subject, action, resource (XACML)

new components:

- **processes** (ID, memory, descriptors for resource allocations - handles)
- **administrative operations**
- **policy classes**

attributes: for users and objects

objects: refer to data

NGAC Assignments and Associations

assignment: $x \rightarrow y$

meaning:

- x belongs to y
- x and y might be: users, user attributes, objects, object attributes, policy classes
- e.g.
  - `John Smith` $\rightarrow$ `CZD -Hospital-doctors`
  - `CZD-Hospital-doctors` $\rightarrow$ `doctors`
  - `file A` $\rightarrow$ `personal-data`
  - `personal-data` $\rightarrow$ AC-secured-data
- somewhat analogous to roles of users in RBAC

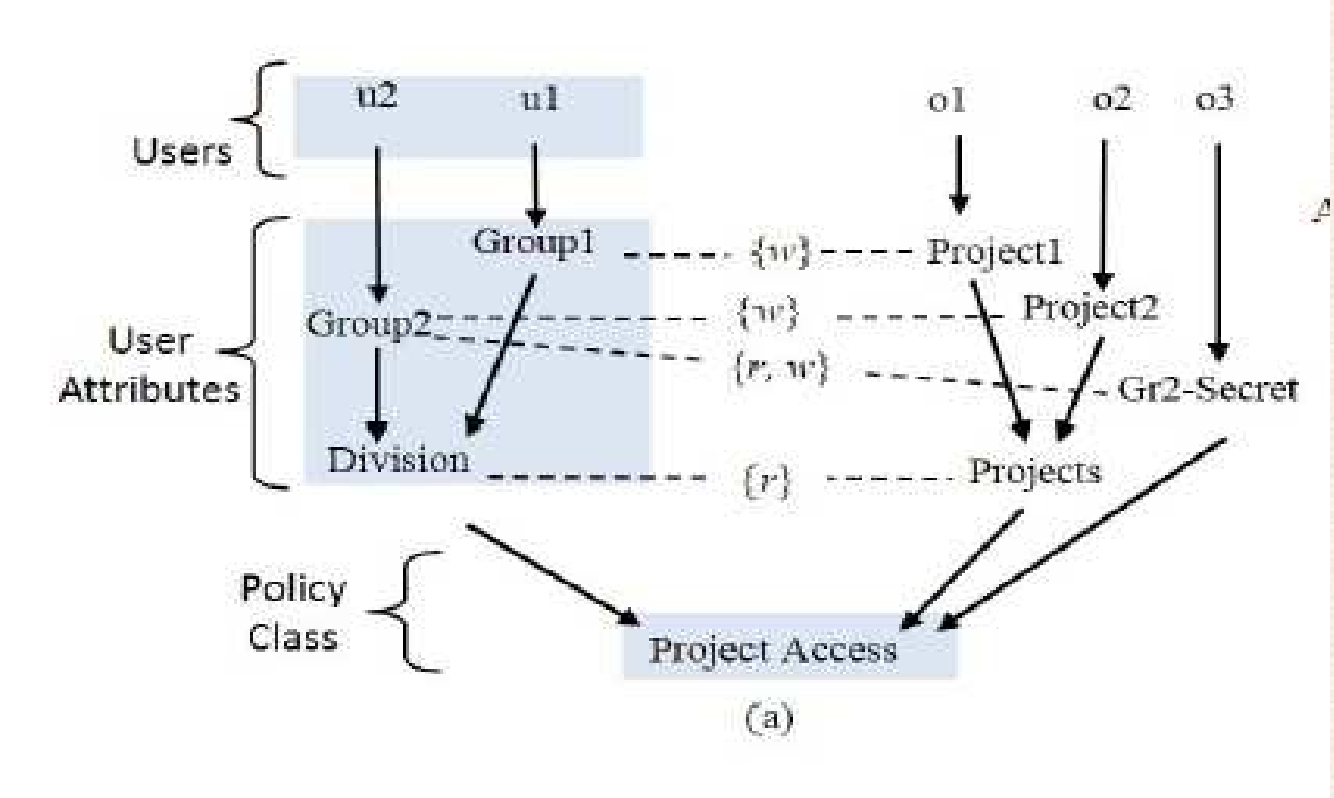
association:

$(\mathrm{ua} -\!\!- \mathrm{ars} -\!\!- \mathrm{at})$ where

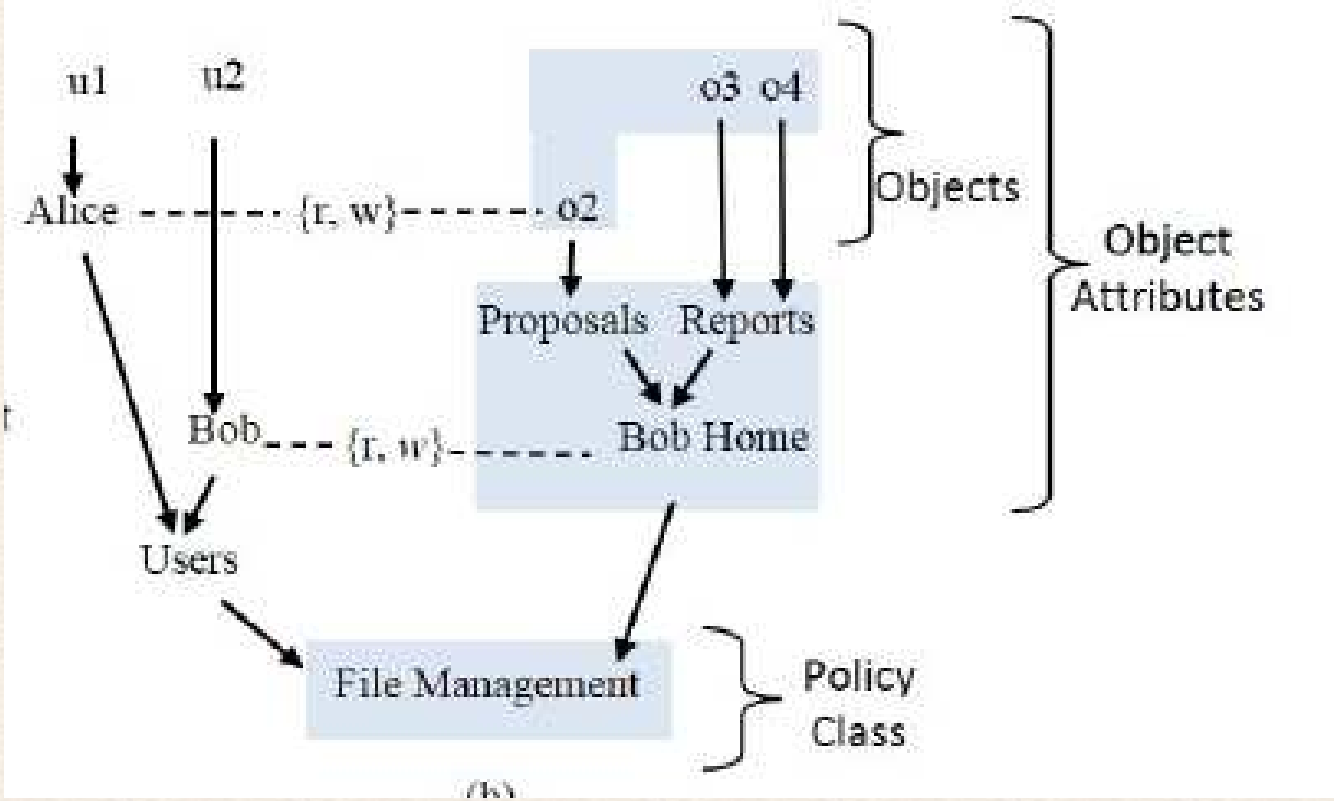
- ua is a user attribute
- ars is a set of access rights
- at is an attribute (user attribute or object attribute)

meaning: users in ua have rights in ars on policy elements referenced by at

NGAC example (from the NIST paper)



NGAC example

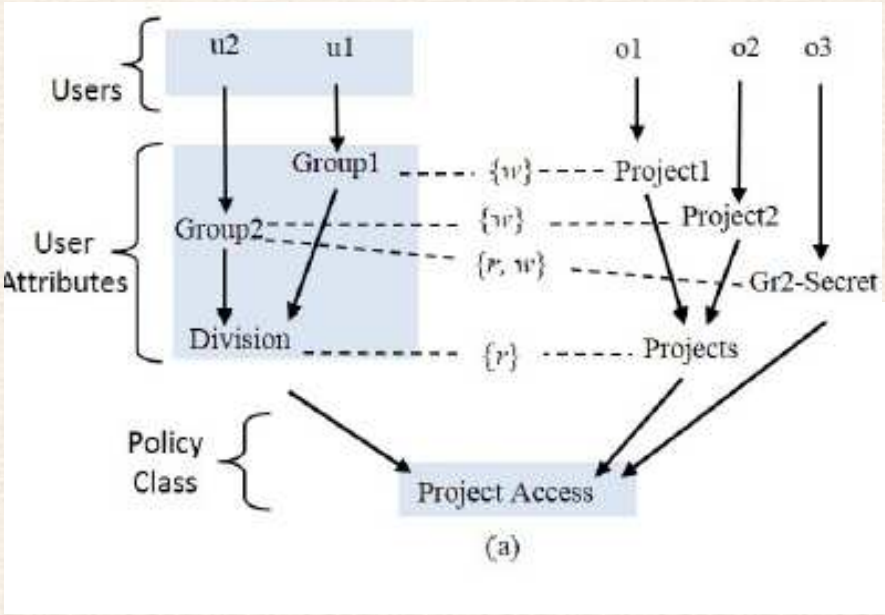


NGAC deriving privileges

(u, ar, e) is a privilege following from $(\mathrm{ua} \text{---} \mathrm{ars} \text{---} \mathrm{at})$ iff

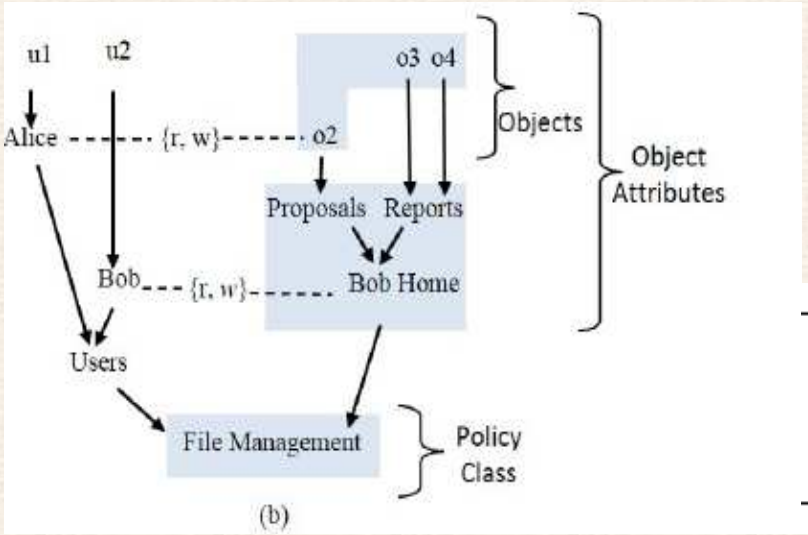
- The user u is contained by the user attribute of an association;
- The element e is contained by the attribute at of that association;
- The attribute at of that association is contained by the policy class pc; and
- The access right ar is a member of the access right set of that association.

NGAC



(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1),
(u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3)

NGAC



(u1, r, o2), (u1, w, o2), (u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)

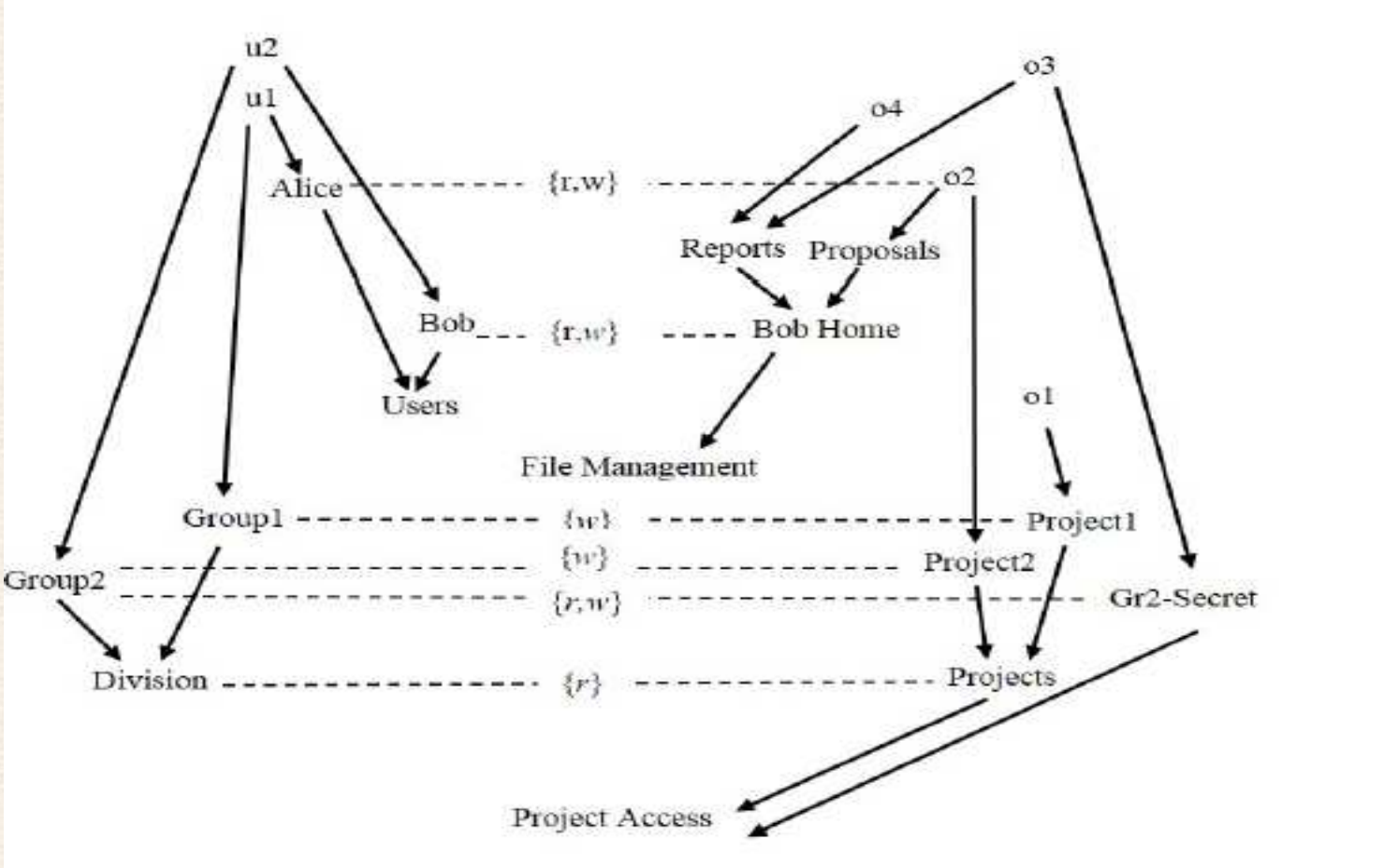
NGAC priviledges - combining policy sets

Rule: the condition mentioned must be

(u, ar, e) is a privilege iff for each policy class

- The user u is contained by the user attribute of an association;
- The element e is contained by the attribute at of that association;
- The attribute at of that association is contained by the policy class pc; and
- The access right ar is a member of the access right set of that association.

NGAC example



(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1), (u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)

Note:

- `(u1,r,o1)` is a priviledge, since `o1` is only in the "Project Access" class, and it is a priviledge of "Project Access"
- `(u1,w,o2)` is not a priviledge, since `o2` is in both policy classes, it is ok for "File Management" but not for "Project Access"

NGAC prohibitions

- u_deny(u,ars,pe) user deny
- ua_deny(ua,ars,pe) attribute deny
- p_deny(p,ars,pe) process deny

where pe is policy element

example: write right to own tax record by a tax auditor of tax authority

effect: deny overides privileges

NGAC prohibitions with negation

u_deny(u,ars,¬pe) means that user u is denied any element that is not in pe

similarly ua_deny(ua,ars,¬pe), p_deny(p,ars,¬pe)

NGAC obligations

(ep,r) means when ep do r

- ep is called "event pattern"
- r is called "response"
- event is an execution of an operation on an object by a process (on behalf of a user)
- event may specify parameters: user, attributes , operation, attributes of the object, opeartional conditions

Application examples:

1. handling conflicts of interest (if a process of user A has read a file H, then it must be prevented to read file J)

2. approval of another user

NGAC decision function

control of accesses in terms of processes

- `process_user(p)`
- access requests: `(p,op, argseq)` where
  - → `p` is a process, op is an operation, `argseq` are arguments of the operation
- mapping: to access right and policy element pairs involved: {ar,pe}

A process access request $(p, op, argseq)$ with mapping $(op, argseq) \rightarrow \{(ar, pe)\}$ is granted iff for each (ar_i, pe_i) in $\{(ar, pe)\}$, there exists a privilege (u, ar_i, pe_i) where u = process_user(p), and (ar_i, pe_i) is not denied for either u or p.

NGAC administrative access rights

non-administrative access rights: pertain to activities on protected rosoures

administrative access rights: pertain to activities on policy and attributes

examples:

- creating files, directories
- assigning attributes

NGAC administrative access rights example

administration over assignments:

ProjectAccessAdmin --- {create-u-to, delete-u-from, create-ua-to, delete-ua-from, create-uua-from, create-uua-to, delete-uua-from, create-uaua-from, create-uaua-to, delete-uaua-from, delete-uaua-to }---Division

administration over associations:

ProjectAccessAdmin --- {create-assoc-from, delete-assoc-from} --- Division.

ProjectAccessAdmin --- {create-assoc-to, delete-assoc-to, r-allocate, w-allocate} --- Projects.

NGAC delegation of administrative access rights

- initially superuser with empty data elements, attributes, relations,... but with all administrative rights
- delegating administrative rights

– administrative access request $\Rightarrow$ administrative routine = set of administrative commands

syntax:

cmdname (x_1: $type_1$, x_2: $type_2$, …, x_k: $type_k$)
…preconditions …
{
$Y' = f(Y, x_1, x_2, \ldots, x_k)$
}

concrete example:

createAssoc (x, y, z)
$x \in UA \wedge y \in ARS \wedge z \in PE \wedge (x, y, z) \notin ASSOC$
{
$ASSOC' = ASSOC \cup \{(x, y, z)\}$
}

NGAC example for routine

```
create-file-mgmt-user(user-id, user-name, user-home) {
     createUAinUA(user-name, Users);
     createUinUA(user-id, user-name);
     createOAinPC(user-home, File Management);
     createAssoc(user-name, {r, w}, user-home);
     createAssoc(user-name, {create-o-to, delete-o-from}, user-home);
     createAssoc(user-name, {create-ooa-from, create-ooa-to,
                             delete-ooa-from, create-oaoa-from,
                             create-oaoa-to, delete-oaoa-from}, user-home);
     createAssoc(user-name, {create-assoc-from, delete-assoc-from}, Users);
     createAssoc(user-name, {create-assoc-to, delete-assoc-to, r-allocate,
                             w-allocate}, user-home);}
```

run create-file-mgmt-user(u1, Bob, Bob-Home) to create Bob's capabilities form the example discussed before

NGAC functional architecture