PAW Related Work

Vladimir Kolovski and Lalana Kagal

Overview

- Frameworks
  - PCA [1]
  - PeerTrust [2]
  - Bonatti et al.[3]
- Policy Languages
  - WS-Policy [4]
  - SAML [5]
  - XACML [6]
  - KaOS [7]
  - WSPL [8]

Proof-Carrying Authorization

- Access control on the web as a general distributed authorization problem
- Builds on previous design tradition by uncoupling authentication from authorization
- Motivated by the problem of lack of interoperability between administrative domains (e.g., two universities)

- PCA is a framework for defining security logics based on higher-order logic.

PCA Properties

- Interoperability and Expressivity
  - security policies in PCA do not have to be based on the identity of the user
  - policies are completely general – the right to access a page can be described by an arbitrary predicate
  - Example, a particular security policy grants access only to people who are able to present the proof of Fermat's last theorem.
  - authentication servers are replaced with more general *fact* servers

PCA Properties

- Web access control system based on a reasoning framework by Appel and Felten, which is higher-order, undecidable logic
- Isn’t this infeasible, since a server might not be able to decide whether a set of credentials implies that access should be granted?
- Proof of access on client side can be described using a subset of higher-order logic that corresponds to a simple and decidable application-specific logic
- The proof of access along with the definition of the application-specific logic in terms of the higher-order logic, is sent to the server.

PCA Architecture

- Types of players:
  - web browsers
    - local proxy that intercepts a browser's request for a protected page and then executes the authorization protocol and generates the proof needed for accessing the page
    - the web browser sees only the result—either the page that the user attempted to access or a failure message.
  - fact servers
    - Fact servers hold the facts a client must gather before it can construct a proof
    - Each fact is a signed statement in the PCA logic.

PCA Architecture

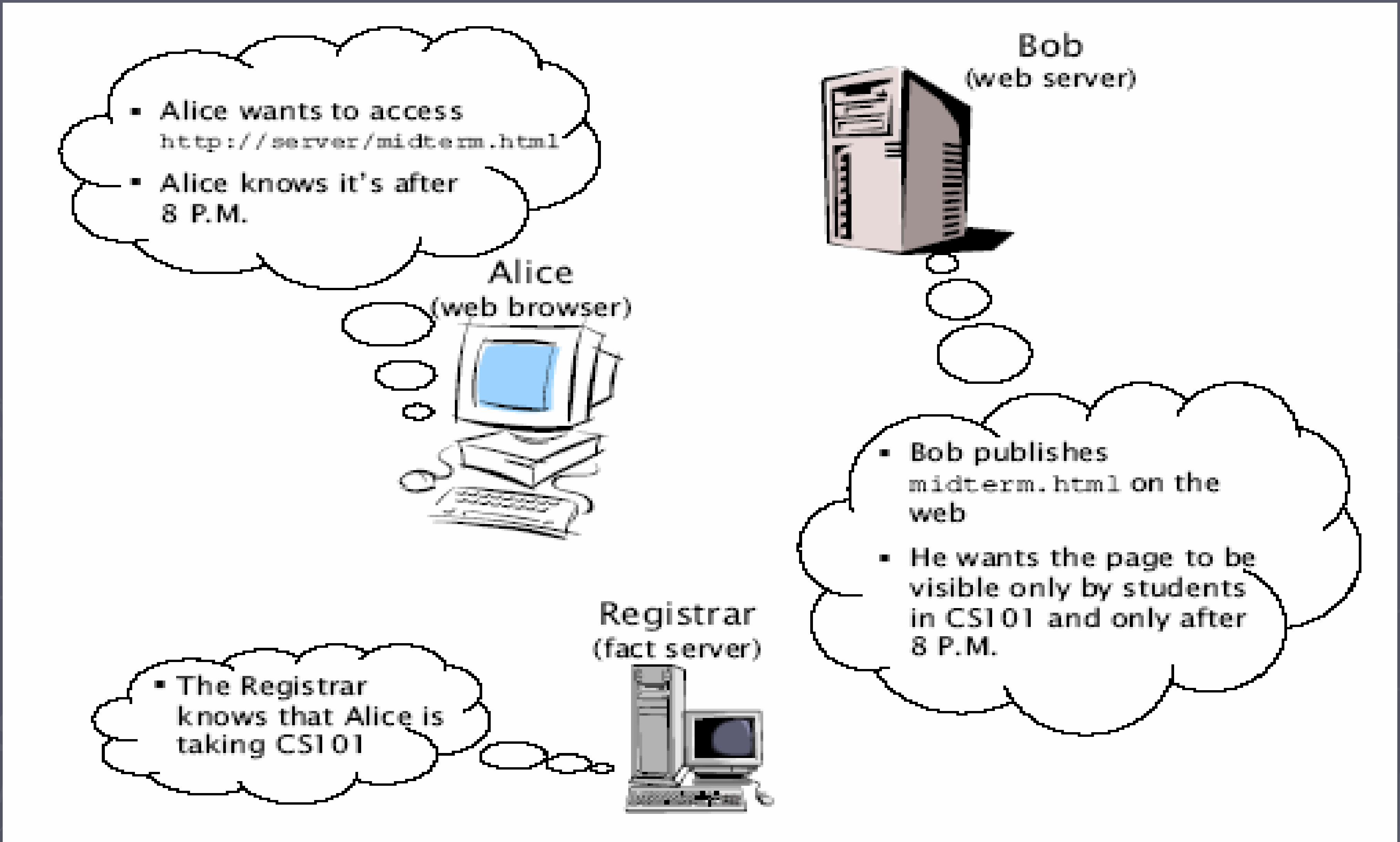
- Web Servers
  - Extended through the use of a servlet which intercepts and handles all PCA-related requests.
  - Two tasks that occur on server's side during an authorization transaction
    - generating the proposition that needs to be proved and
    - verifying that the proof provided by the client is correct.

  - Each is performed by a separate component, the proposition generator and the checker.

Proof generation/checking

- It is the client’s responsibility to prove that access should be granted
- All the server needs to do is verify that the client’s proof is valid, which can be done efficiently even if the proof is expressed in an undecidable logic.
- Client’s task is feasible because it does not need the full expressivity of the higher logic - only uses a decidable subset.

PCA Scenario



Proof Generation, revisited

- Proofs are generated automatically by a logic program
- The goal that must be proven is encoded as the statement of a theorem.
- Facts that are likely to be helpful in proving the theorem are added as assumptions.
- The logic program generates a derivation of the theorem; this is the “proof” that the proxy sends to the server.

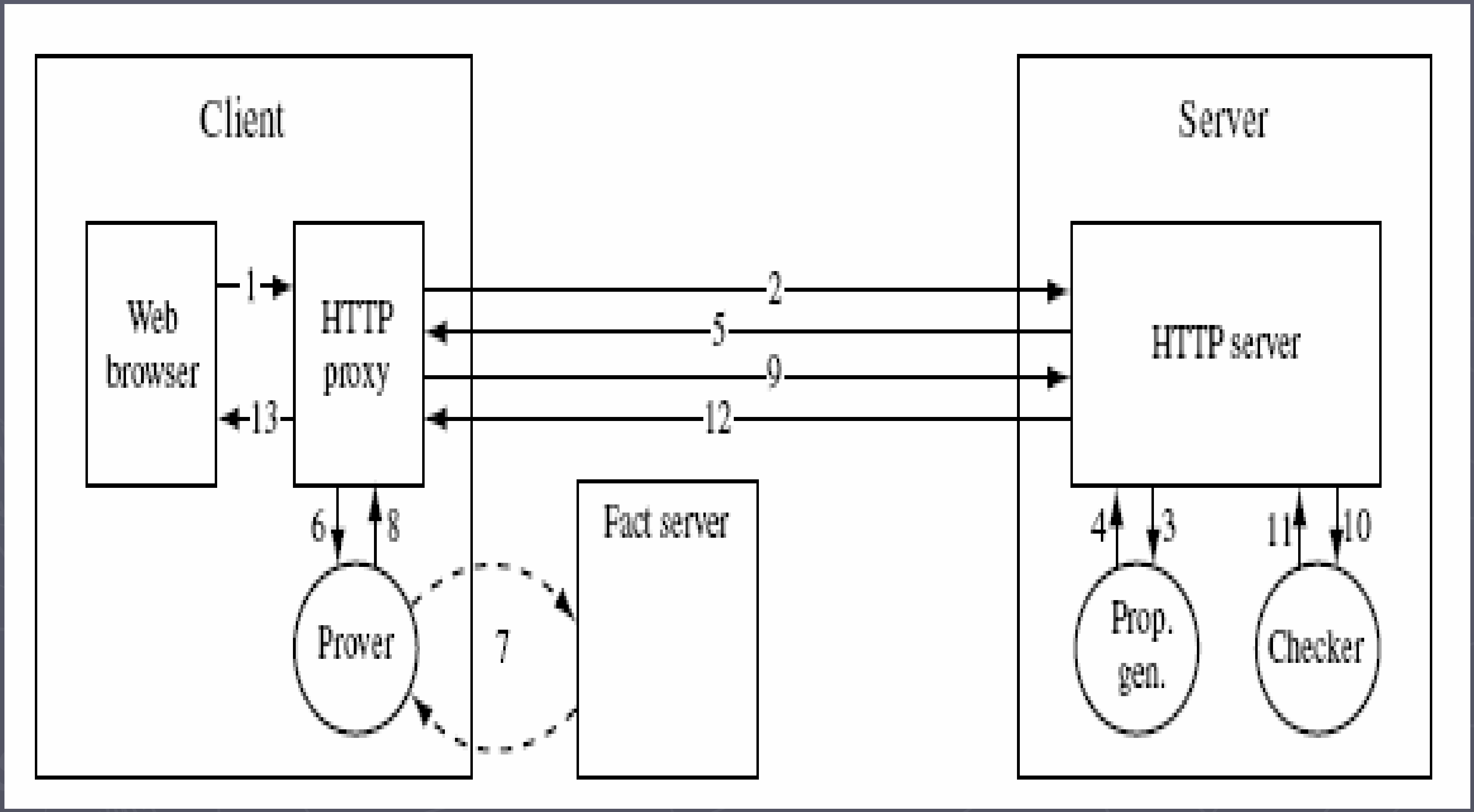
Proof generation, revisited

- Client’s job is to find all the assumptions that are required by the proof.
- Assumptions might include
  - statements made by the server about who is allowed to access a particular file,
  - guesses about time,
  - statements by which principals delegate authority to other principals, etc.
- Some assumptions might not be known to the client - need to be obtained from web pages (Iterative proving)

Proof Checking, revisited

- Proof checking reduced to type checking, where
  - The type of the term is the statement of the theorem that must be proven;
  - the body of the term is the proof itself.
- If the term is well typed, the client has succeeded in proving the proposition.
- Proofs have to be explicitly typed, which is practical only for small examples
- Preprocessing before submitted to the checker

PCA Diagram



Client
Server
Web browser
HTTP proxy
HTTP server
Fact server
Prover
Prop. gen.
Checker
1
2
5
9
12
13
6
8
7
4
3
11
10

Drawbacks of PCA

- Too much work for client?
  - Each of the operators in client's decidable logic subset should be given a definition in higher-order logic, and each of the inference rules should be defined as a lemma.
  - Has to define its own application-specific, decidable logic, construct a proof of access in that logic and then submit the proof together with a mapping of its terms to higher order logic to the server
  - Proofs blowup in size (every term has to be typed)
  - Client's work cannot be fully automated

Example client inference rule

- SPEAKSFOR-E is simple delegation

A **says** (*B* **speaksfor** *A*)   *B* **says** **goal**(*URL*;*nonce*))

A **says** (**goal**(*URL*;*nonce*))

Semantics in Higher Order logic

Figure 3.6: Proof of the SPEAKSFOR-E theorem.

1	A **says** (B **speaksfor** A)	premise
2	B **says** (**goal**(U,N))	premise
3	$[\forall U' \forall N'. B$ **says** (**goal**$(U',N')) \longrightarrow A$ **says** (**goal**$(U',N'))]$	assumption
4	B **says** (**goal**$(U,N)) \longrightarrow A$ **says** (**goal**(U,N))	$\forall_{U,N}$ e 3
5	$(\forall U' \forall N'. B$ **says** (**goal**$(U',N')) \longrightarrow A$ **says** (**goal**$(U',N')))$	
	$\longrightarrow (B$ **says** (**goal**$(U,N)) \longrightarrow A$ **says** (**goal**$(U,N)))$	$\longrightarrow$ i 3–4
6	A **says** $((\forall U' \forall N'. B$ **says** (**goal**$(U',N')) \longrightarrow A$ **says** (**goal**$(U',N')))$	
	$\longrightarrow (B$ **says** (**goal**$(U,N)) \longrightarrow A$ **says** (**goal**$(U,N))))$	SAYS-I2 5
7	A **says** $(\forall U' \forall N'. B$ **says** (**goal**$(U',N')) \longrightarrow A$ **says** (**goal**$(U',N')))$	$\stackrel{\text{def}}{=}$ e 1
8	A **says** (B **says** (**goal**$(U,N)) \longrightarrow A$ **says** (**goal**(U,N)))	SAYS-I3 6, 7
9	A **says** (B **says** (**goal**(U,N)))	SAYS-I2 2
10	A **says** (A **says** (**goal**(U,N)))	SAYS-I3 8, 9
11	A **says** (**goal**(U,N))	SAYS-TAUT 10

PeerTrust Motivation

- Access control in a p2p network that connects commercial e-learning providers with learning management systems
- Suppose E-Learn Associates manages a Spanish course, and Alice wishes to access that course
- Access Policy: free of charge to all police officers who live and work for the state of California.
- Alice is reluctant to share her police badge and driver’s license freely – she has her own policy for sharing

Trust Negotiation

- Access control no longer unilateral
- In the example, E-Learn will have to show that satisfies the access policies for Alice's credential
- In doing so, E-Learn might have to disclose additional credentials of its own – but only after Alice demonstrates she satisfies the policies for each of *them*
- Peertrust uses automatic trust negotiation for this purpose

Trust Negotiation

- trust is established by exchange of information
- trust establishment process is bi-directional
- PeerTrust uses digital credentials (signed assertions) to manage trust establishment
- Trust is established incrementally through an iterative process which involves gradually disclosing credentials and requests for credentials

PeerTrust Language

- Peertrust’s policy and negotiation language is based on guarded distributed logic programs.
- Based on first order Horn rules
  - lit <- lit_1, lit_2, … , lit_n
  - each lit_i is a positive literal
- closed-world semantics
- the literals in the clauses can represent external procedure calls.
- can be used to call authentication libs and check environmental conditions mentioned in a policy

PeerTrust Language Example

- eOrg:
  preferrred(X) <- student(X) @ UMD

- eOrg:
  student(X) @ UMD <- student(X) @ UMD @ X

- eLearn:
  freeEnroll(Course,Requester) $ Requester <-
  policeOfficer(Requester) @ csp @ Requester,
  spanishCourse(Course)

PeerTrust & PAW

- PeerTrust's policies are sensitive and not freely shared
- Most of their work about policy protection and bilateral iterative disclosure of credentials
- PeerTrust's trust negotiation is analogous to PAW's proof exchange
  - its negotiation protocol goes through a lot of stages
  - no guarantee that it will even terminate
  - PeerTrust's policy language can only be used for positive authorization, delegation is simple
  - Similar to PAW in the aspect of decentralized proof generation. But we are working with unilateral trust

Uniform Framework for Regulating Service Access

- closely related to PeerTrust
- provides a means for formulating and reasoning about both services access and information disclosure constraints
- same as in PeerTrust, this project gives the client the ability to present counter-requests to servers and put restrictions on information disclosure
- Identification and authentication requirements can be expressed through the language itself

- Keeps some state information on all parties
- Assumption about semi-structured organization of credentials that allows querying for specific data (name and address in a drivers license)
- Their work addresses two issues:
  - policy filtering – the process of selecting the rules that should be sent to the client
  - service renaming – used in cases where servers wish to hide the details of the services they provide

Client's Policy Evaluation

- given the server's requirements (with filtered and renamed policies), the client searches its portfolio for a set of credentials/declarations that satisfy them
- using XSB and the server's requirements as input, a top down proof is constructed
- credential and declaration atoms are gathered as needed
- description of system implementation bit unclear, not finished yet

Relation between PAW and Bonatti's Work

- most of the PeerTrust differences apply here, too
- this work targets different types of policies, where clients are reluctant to share them freely,
- hence most of the work is done in the area of protecting the policies
- also, they keep persistent and negotiation state

WS-Policy

- Extremely simple
  - Assertion sets
    - Arbitrary XML for domain knowledge
    - Generic engines treat as atomic propositions
  - (Exclusive-)disjunctive normal form
    - <wsp:All> == conjunction
    - <wsp:ExactlyOne> == exclusive-disjunction
  - Two “operations”
    - Merge (more conjunction)
    - Intersection

WS-Policy Example

```
<wsp:Policy>
    <wsp:ExactlyOne>
      <wsp:All>
        <wsse:SecurityToken>

      <wsse:TokenType>wsse:Kerberosv5TGT</wsse:TokenType>
        </wsse:SecurityToken>
      </wsp:All>
      <wsp:All>
        <wsse:SecurityToken>
                <wsse:TokenType>wsse:X509v3</wsse:TokenType>
        </wsse:SecurityToken>
      </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```

Mapping to OWL

- Extremely simple
  - Assertions == Class (atomic as first approx)
  - <wsp:All> == owl:intersectionOf
  - <wsp:ExactlyOne> == owl:unionOf + owl:complementOf the owl:intersectionOf
- Issue: OWL is first order
  - So open world assumption
    - Being ExactlyOne can be tricky
    - Reasoner might return “unknown”
  - No unique name assumption
- Implementation
  - XSLT (with customization for assertion sets)

Policy Processing

- Policy Analysis…
  - Conformance == class membership
    - If *x* is rdf:type SomePolicy, then it *conforms* to SomePolicy
  - *containment* (and equivalence)
    - If *x* meets policy A, then it meets policy B
  - *incompatibility*
    - If *x* meets policy A, then it can't meet policy B
  - *incoherence*
    - Nothing can meet policy A
- Debugging and Explanation of policies

Update on WS-Policy

- Implemented XSLT that converts both
  - the WS-Policy constructs (ExactlyOne, All)
  - the assertions themselves
    - use OWL constructs to recover structure – they're not treated atomic anymore

```
<wsse:Integrity>
        <wsse:Algorithm Type="wsse:AlgEncryption"
            URI="http://www.w3.org/2001/04/xmlenc#3des-cbc" />
</wsse:Integrity>
```

- Also have a mapping for Merge operator

SAML

- It's an XML-based framework for exchanging security information
  - XML-encoded security assertions
  - XML-encoded request/response protocol
  - Rules on using assertions with standard transport and messaging frameworks
- Useful for Single Sign On, Distributed Transaction, Authorization service

SAML Intro

- SAML is different from other security approaches because of its expression of security in the form of assertions about subjects
- Other approaches use a central certificate authority to issue certificates that guarantee secure communication from one point to another within a network
- With SAML, any point in the network can assert that it knows the identity of a user or piece of data. It is up to the receiving application to accept if it trusts the assertion.

What SAML is not

- SAML is an authentication *protocol* that is used between servers.
- You still need something that actually performs the login for you.
- For example, when an LDAP server authenticates a user, the authentication authority is the LDAP server even though the LDAP server may be using SAML to communicate the authorization.
- Tightly integrated, but different than XACML
  - SAML addresses authentication and provides a mechanism for transferring authentication and authorization decisions, XACML focuses on the mechanism for arriving at those authorization decisions.

Assertions & Statements

- Assertions are declarations of facts about a subject according to the issuer
  - E.g. John says the sky is blue
- An SAML assertion may contain multiple statements
- Three kinds of statements
  - Authentication
  - Attribute
  - Authorization decision
- You can extend SAML to make your own kinds of assertions and statements
- Assertions can be digitally signed

Content of Assertions

- Issuer ID and issuance timestamp
- Assertion ID
- Subject
  - Name plus the security domain
  - Optional subject confirmation, e.g. public key
- Conditions under which assertion is valid
  - SAML clients must reject assertions containing unsupported conditions
  - E.g. NotBefore, NotOnOrAfter, OneTimeUse, Audience

Example Assertion

```
<saml :Assertion
    xmlns:saml = "urn:oasis:names:tc:SAML:2.0:assertion"
    Version="2.0"
    AssertionID="example-123-0"
    Issuer="w3c.prg"
    IssuerInstant="2005-08-23T14:57:47Z">

    <saml:Conditions
        NotBefore="2005-08-23T14:57:47Z"
        NotAfter="2005-08-24T12:00:00Z"/>

    <saml:Subject>
        <saml:NameIdentifier
            SecurityDomain="w3.org"
            Name="uberuser" />
    </saml:Subject>

</saml:Assertion>
```

Authentication Statements

- Structure
  - subject S
  - was authentication by means M
  - at time T
- Does not actually check credentials
- Just shows that subject was authenticated sometime in the past by the sender
- Useful for Single Sign On

Authentication Example

```
<saml:Assertion
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"  Version="2.0”
  AssertionID=” example-123-1">
      <saml:Issuer>http://w3.org/issuer</saml:Issuer>
      <saml:Subject>
            <saml:NameIdentifier
                  Format="urn:oasis:names:…format:emailAddress">
              uberuser@w3.org
            </saml:NameIdentifier>
       </saml:Subject>
      <saml:AuthnStatement AuthnInstant = "2005-08-23T14:57:47Z”>
              <saml:AuthnContent>
                     <saml:AuthnContextClassRef>
                             urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
                     </saml:AuthnContextClassRef>
              </saml:AuthnContext>
      </saml:AuthnStatement>
       <saml:Conditions
              NotBefore=“2005-08-23T14:57:47Z”
              NotAfter=“2005-08-24T12:00:00Z”/>
</saml:Assertion>
```

Attribute Statement

- Structure
  - subject S
  - has attributes A, B, …
  - with value “a”, “b”, “c”, …
- Useful for distributed transactions and authorization services

Attribute Example

```
<saml:Assertion AssertionID=" example-123-2">

  <saml:AttributeStatement>
      <saml:Subject>
          <saml:NameIdentifier
              SecurityDomain="w3.org"
              Name="uberuser" />
      </saml:Subject>
      <saml:Attribute
        AttributeName="Group"
        AttributeNamespace="http://w3.org/group">
          <saml:AttributeValue>
            AdvisoryCouncil
          </saml:AttributeValue>
      </saml:Attribute>
  </saml:AttributeStatement>

  </saml:Assertion>
```

Authorization Decision Statement

- An issuing authority decides whether to grant the request
  - by subject S
  - for access type A
  - to resource R (web page or a service)
  - given evidence E (one of more assertions used to make decision)

Authorization Decision Example

```
<saml:Assertion AssertionID=" example-123-1">

   <saml:Subject>
      <saml:NameIdentifier
          Format="urn:oasis:names:…format:emailAddress">
        uberuser@w3.org
      </saml:NameIdentifier>
    </saml:Subject>

    <saml:AuthzDecisionStatement
        Resource="http://w3.org/secret.html"
        Decision="Permit">
        <saml:Action
         Namespace="urn:oasis:names:tc:SAML:2.0:action:ghpp">
         GET
        </saml:Action>
    </saml:AuthzDecisionStatement>

</saml:Assertion>
```

SAML Requests

- You can query for specific kinds of assertion
  - Authentication query
  - Attribute query
  - Authorization decision query
- You can also ask for an assertion with a particular ID
  - By providing an ID reference
  - By providing a SAML “artifact”

Authentication Request

- Structure
  - Please provide
  - authentication information
  - for subject S
- A successful response is in the form of an assertion containing an authentication statement
- It is assumed that the requester and responder have a trust relationship
  - They are talking about the same subject
  - The response with the assertion is a “letter of introduction” for the subject

Attribute Request

- Structure
  - Please provide information
  - on attributes A or all
  - for subject S
- If the requester is denied access to some of the attributes either a partial list is returned, or no attributes at all

Authorization Decision Request

- Structure
  - is subject S
  - allowed to perform action A
  - on access resource R
  - given this evidence E
- This is is a yes-or-no question

Request Example

```
<samlp:AuthzDecisionQuery
     ID=”example-123-4”
    Version="2.0"
    IssuerInstant=“2005-08-23T14:57:47Z”
     xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
     xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol”
    Resource=“http:w3c.org/secret.html”>
    <saml:Subject>
        <saml:NameIdentifier
             Format="urn:oasis:names:…format:emailAddress">
          uberuser@w3.org
        </saml:NameIdentifier>
     </saml:Subject>
     <saml:Action Namespace="urn:oasis:names:tc:SAML:1.0:action:ghpp">
      GET
     </saml:Action>
  </samlp:AuthzDecisionQuery>
```

SAML Response

- Assertions
- Status codes
  - Success
  - VersionMismatch
  - Receiver
  - Sender
- Responses can be signed

SAML Summary

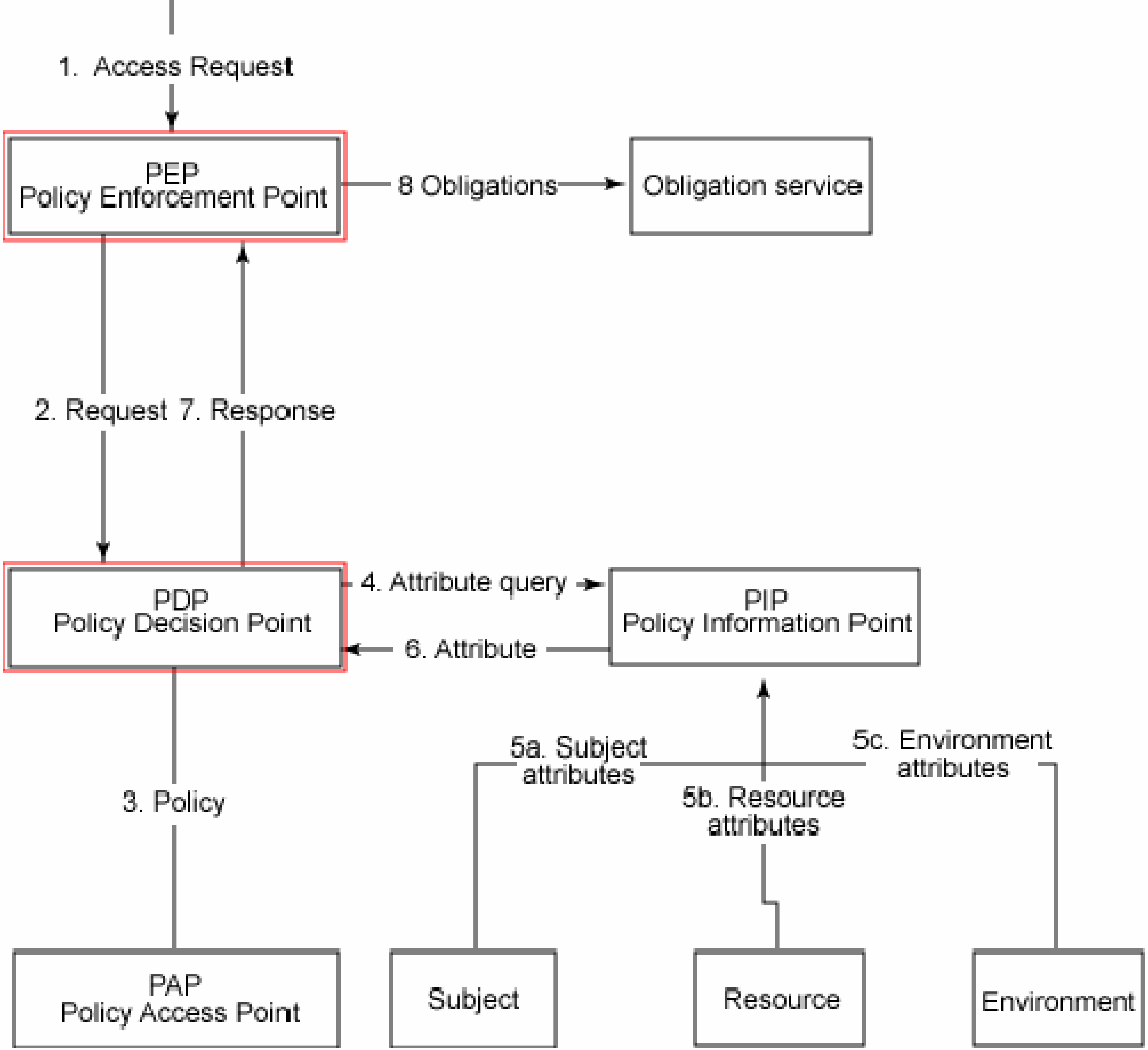
- It's an XML-based framework for exchanging security information
- Useful for Single Sign On, Distributed Transaction, Authorization service

- Could be used in PAW to exchange authentication and authorization information while proof checking
  - E.g. Within John's proof for why he can access w3.org/secret.html he says that Steve says that he is a W3C member. PAW can use SAML to request authentication info from Steve.

XACML

- OASIS eXtensible Access Control Markup Language
- Includes policy language and request/response language
  - policy language is used to describe general access control

- SAML standard provides interfaces that allow third parties to send their requests for authentication and authorization.
- XACML not only processes the authorization requests, but it defines the mechanism for creating the complete infrastructure of rules, policies, and policy sets to arrive at the authorization decisions



1. Access Request
PEP
Policy Enforcement Point
8 Obligations
Obligation service
2. Request
7. Response
PDP
Policy Decision Point
4. Attribute query
6. Attribute
PIP
Policy Information Point
3. Policy
5a. Subject attributes
5b. Resource attributes
5c. Environment attributes
PAP
Policy Access Point
Subject
Resource
Environment

XACML Policy Language

- Policy Sets made of Policies and Rules
- Policies have targets to check the suitability of a policy for a given request
  - simplified conditions for the Subject, Resource, and Action

```
<Target>
    <Subjects/>
    <Resources>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
    DataType="http://www.w3.org/2001/XMLSchema#string">SampleServer</AttributeValue>
        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resources>
    <Actions><AnyAction/></Actions>
  </Target>
```

XACML Policy Language

- Rules associate boolean conditions with an effect (deny, permit…)
  - Any user with an e-mail name in the "med.example.com" namespace is allowed to perform any *action* on any *resource*.

```
<Rule RuleId= "urn:oasis:names:tc:xacml:2.0:example:SimpleRule1" Effect="Permit">
  <Target>
    <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
             <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">med.example.com
             </AttributeValue>
             <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                  DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
             </SubjectMatch>
    </Subject>
  </Target>
 </Rule>
```

- Conditions are boolean combinations of attribute-value pairs

XACML Policy Language

- Supports several datatypes like date, time, boolean, string, integer
- Combining algorithms for conflict resolution
  - Deny-overrides, ordered-deny-overrides, permit-overrides, ordered-permit-overrides, first-applicable, only-one-applicable

```
<Policy PolicyId="SamplePolicy"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
```

- A policy can include an obligation
  - When policy is evaluated, the obligation is passed to the enforcing mechanism

Attributes

- Conditions are made up of attributes
- Attributes are characteristics of the Subject, Resource, Action, or Environment in which the access request is made
- A Policy resolves attribute values either in the request document or elsewhere through two mechanisms
  - AttributeDesignator
    - Lets the policy specify an attribute with a given name and type, and optionally an issuer as well
    - There is one for each of the types of attributes in a request: Subject, Resource, Action, and Environment

```
<Actions>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
                              AttributeId="RequestedAction"/>
        </ActionMatch>
 </Actions>
```

Attributes

- AttributeSelector
  - Allow a policy to look for attribute values through an XPath query
  - A data type and an XPath expression are provided
- Both AttributeDesignator and AttributeSelector return multiple values

XACML Functions

- Functions are used to compare multiple values that AttributeDesignators and AttributeSelectors return
  - Functions work on any combination of attribute values, and can return any kind of attribute value supported in the system
  - Arithmetic, string, numeric converters, logical operators, date and time, bag, set, xpath
  - Functions can also be nested
  - Custom functions can also be written

XACML Function Example

```
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
            <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
            <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
                                AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
        </Apply>
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">17:00:00</AttributeValue>
    </Apply>
</Condition>
```

XACML Request

- Request
  - (subject, resource, action)

```
<Request>
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
     DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
     <AttributeValue> bs@simpsons.com</AttributeValue>
   </Attribute>
  </Subject>
   <Resource>
     <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource859
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
     <AttributeValue> file://example/med/record/patient/BartSimpson </AttributeValue>
     </Attribute>
   </Resource>
 <Action>
   <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
     DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue> read </AttributeValue>
   </Attribute>
 </Action>
</Request>
```

XACML Response

- Response
  - Permit
  - Deny
  - Indeterminate (an error occurred or some required value was missing, so a decision cannot be made)
  - Not Applicable (the request can't be answered by this service).

XACML Summary

- Policy language in XML
- Can be used with SAML's request/response protocol

- Comparison to Rei(n)
  - Non rule based
  - Using combining algorithms for conflict resolution
  - Priorities cannot be set for policies or rules for conflict resolution
  - Lots of syntax
  - No delegation
  - Sun has an XACML implementation

KAoS

- Is an ontology-based policy language
  - Relies on the features of OWL to express policies
- Uses JTP to reason over policies
- A KAoS policy is an instance of the appropriate policy type that defines the associated values for its properties
- The context for the action is defined through various property restrictions in the action type
- Provides static policy conflict detection
  - Uses subsumption reasoning between classes
- Conflict resolution
  - By ordering policies according to their precedence

KAoS Example

```
<owl:Class rdf:ID=”ExampleAction">
    <rdfs:subClassOf rdf:resource="#EncryptedCommunicationAction" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#performedBy" />
            <owl:toClass rdf:resource="#MembersOfDomainA" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDestination" />
            <owl:toClass rdf:resource="#MembersOfDomainA " />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

KAoS Example (Cont)

<policy:PosAuthorizationPolicy rdf:ID="Example">

<policy:controls rdf:resource="#ExampleAction" />

<policy:hasSiteOfEnforcement rdf:resource="#ActorSite" />

<policy:hasPriority>10</policy:hasPriority>

<policy:hasUpdateTimeStamp>4237445645589</policy:hasUpdateTimeStamp>

</policy:PosAuthorizationPolicy>

KAoS Summary

- Is an OWL based policy language
- Comparison with Rei(n)
  - Non rule-based so less expressive
  - Simple delegation mechanism
  - Static conflict detection
  - GUI for developing policies
  - Has an enforcement framework

Why higher order logic?

- Many security logics have higher-order features like relations that range over formulas :

$$\frac{A\ \textbf{says}\ (B\ \textbf{speaksfor}\ A) \qquad B\ \textbf{says}\ (\textbf{goal}(URL;nonce))}{A\ \textbf{says}\ (\textbf{goal}(URL;nonce))}$$

References

1. PCA http://www.cs.princeton.edu/research/techreps/TR-677-03
2. PeerTrust http://www.l3s.de/%7Eolmedilla/pub/trustVLDB04.pdf
3. Bonatti's work

   http://seclab.dti.unimi.it/Papers/jcs-cred.ps
4. WS-Policy http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/
5. SAML http://xml.coverpages.org/saml.html
6. XACML http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
7. KaOS http://www.coginst.uwf.edu/Papers/KAoS_Policy_Service_2003.pdf
8. WSPL

   http://research.sun.com/projects/xacml/Policy2004.pdf