J2SE Use Case

Sekhar Vajjhala
Sun Microsystems

XACML F2F #3
Jan 23−24 2002

Agenda

- Problem we are trying to solve
  - Use of SAML/XACML in J2SE
- J2SE architecture
  - Java Permission Model
  - Policy Enforcement in J2SE platform
  - Policy Evaluation in J2SE platform
- Example J2SE policy files in XACML syntax
- Issues

Use of SAML/XACML in J2SE

- Two use cases
  - XACML as replacement for default policy syntax
    - Use of XACML for configuring security policies − not as a data interchange format
    - Issue: Non−SAML inputs
  - Integrate third party Policy providers into the J2SE platform.
    - Issue: SAML schema extensions/changes
  - Other uses for XACML and SAML...

J2SE Permission

- Controls access to a resource
- Represented by a Class
  - e.g. java.io.FilePermission
- J2SE platform defines permissions.
- Applications can define permission classes.

Access Control

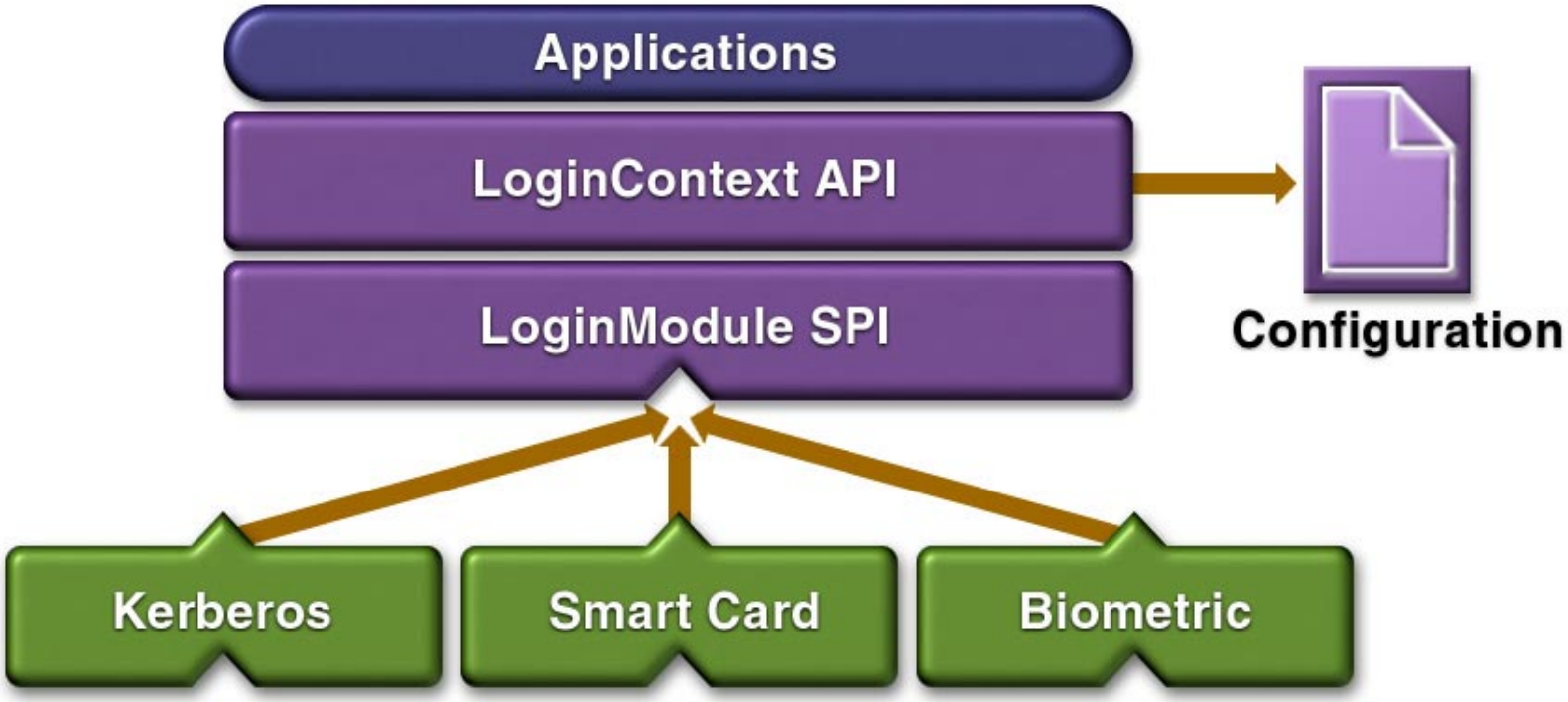
- Permission must be granted for access to a resource.
  - No negative permissions
  - Access denied if no permission granted
- Permissions are granted based on
  - code where it originated
  - a executing principal
  - or both of the above

Policy File Entry

```
01    /**
02     * CodeSource based grant entry
03     */
04    grant Codebase "www.sun.com", Signedby "duke" {
05        java.io.FilePermission "/tmp/*",
"read,write";
06    }

01    /**
02     * CodeSource and principal based grant entry
03     */
04    grant Codebase "www.sun.com", Signedby "duke",
05        Principal com.sun.Principal "charlie" {
06        java.io.FilePermission "/tmp/*",
"read,write";
07    }
```

JAAS Pluggable Authentication



JAAS Subject

- JAAS subject
  - container for principals, credentials
    - JAAS based on PAM
      - stacked authentication
      - multiple principals for a subject
      - different terminology from SAML
  - Populated upon authentication
  - associated with a thread by the application
  - principals used in authorization

ProtectionDomain

- A ProtectionDomain
  - CodeSource, Classloader, array of principals, Permission collection
- Class belongs to a single ProtectionDomain
- Permission checking
  - classes mapped to protection domain
  - permission for protection domain are checked

Policy Enforcement

- Application
  - FilePermission perm = new java.io.FilePermission(“/tmp/*, “read,write”);
  - AccessController.checkPermission(perm);
- AccessController
  - does a stack walk
  - for each caller in the stack
    - map the class to the ProtectionDomain
      - (ignoring the ability of caller to assert its privilege here)
    - Use the Policy Provider to check if the permission is granted to the protection domain

J2SE Policy Evaluation

- J2SE Policy provider must implement abstract Policy class.
- Important Methods
  - getPermissions(ProtectionDomain domain)
    - Returns a PermissonCollection
  - implies(ProtectionDomain, Permission)
    - Issue: Can the information in a ProtectionDoman be sent in a SAML AuthorizationQuery to a PDP for policy evaluation ?

Implies method

- Used for equivalances
  - CodeSource.implies()
  - Permission.implies()
    - permission p1 can imply permission p2
      - subset test rather than an equality test
- implies() method should continue to be used
  - pass class information and parameters in SAML AuthorizationQuery to instantiate classes on the PDP.

Issue: saml:Action

- saml:Action is defined as a String
- Action can be arbitrarily complex
  - forces policy writer to revert to non XML syntax
- Proposed change to SAML schema

```
<xs:complexType name="ActionAbstractType" abstract="true"/>
  <element name="Action" type="ActionAbstractType">
```

J2SE Permission Schema

```
<xs:schema targetNamespace="JavanamespaceURI">

    <element name=“action” type = “string”>

    <xs:complexType name="SignerType">
     <attribute name="class" type="string">
     <attribute name="id" type="string"
    </xs:complexType>

   <xs:complexType name="JavaPermission>
     <xs:complexContent>
      <xs:extension base="saml:ActionAbstractType">
       <xs:sequence>
        <element name = ”javans:action" minOccurs = 0 maxOccurs = "unbounded" >
       </xs:sequence>
        <attribute name = "class" type = ”string" use = "required" />
       <attribute name = "Signer" type = "SignerType" use = "optional">
     </xs:complexContent>
   </complexType>
  </xs:schema>
```

J2SE Permission Example

```
<applicablePolicy majorVersion="1" minorVersion="0" issuer="??????" >
    <target resourceClassification="/tmp/*”
            resourceToClassification=regular−expression−uri >
      <javans:JavaPermission class="java.io.FilePermission">
        <javans:action>read</javans:action>
        <javans:action>write</javans:action>
      </javans:JavaPermission>
    </target>
    ……..
```

Principal based grant entry

```
 grant principal javax.security.auth.x500.X500Principal "cn=Alice"
{
      permission java.io.FilePermission "/home/Alice", "read, write";
   };
```

```
<AuthorizationQuery Resource="/home/alice" >
    <! Note: I am not sure if /home/alice is a valid form for anyURI
    <Subject>
      <saml:NameIdentifier SecurityDomain = "javax.security.auth.x500.X500Principal”
              name="cn:Alice"/>
    </Subject>
    <saml:Actions Namespace="JavaNameSpaceURI">
       <javans:JavaPermssion class="java.io.FilePermission">
          <javans:action>read</javans:action>
    </saml:Actions>
 </AuthorizationQuery>
```

Principal based grant entry (Contd)

- Policy evaluated as follows:
  - principal names in policy must match the one in AuthorizationQuery
  - princpal class name must match the one in AuthorizationQuery
  - Instantiate a permission class
    - grantedpermission.implies(requestedpermission) must return true