# Control de acceso basado en atributos (Attribute-based access control, ABAC)

El control de acceso basado en atributos (ABAC) es un modelo de control de acceso que utiliza atributos (como roles de usuario, propiedades de recursos y condiciones ambientales) para tomar decisiones de control de acceso. Es una forma flexible y dinámica de gestionar el acceso a recursos protegidos.

# ¿Qué es el control de acceso basado en atributos (ABAC)?

ABAC es un modelo de Control de acceso (Access control) que utiliza atributos para tomar decisiones de control de acceso. Estos atributos pueden incluir varios factores, como:

- Atributos de usuario: por ejemplo, roles, departamento, ubicación, etc.
- Atributos de recursos: por ejemplo, nivel de sensibilidad, propietario, tipo, etc.
- Atributos ambientales: por ejemplo, hora de acceso, ubicación, dispositivo, etc.

Al evaluar estos atributos y ejecutarlos a través de un conjunto de reglas, ABAC puede determinar si un sujeto (por ejemplo, usuario, servicio) debe tener acceso a un recurso. Este enfoque permite un control de acceso detallado y una aplicación dinámica de políticas basadas en el contexto.

# ¿Cómo funciona el ABAC?

ABAC utiliza un enfoque basado en políticas para el control de acceso. Una política ABAC típica consta de:

- **Subject**: La entidad que solicita acceso (por ejemplo, usuario, servicio, dispositivo).
- **Action**: La operación que se realiza en el recurso (por ejemplo, leer, escribir, eliminar).
- **Resource**: La entidad a la que se accede (por ejemplo, archivo, base de datos, API).
- **Environment**: El contexto en el que se solicita el acceso (por ejemplo, hora, ubicación, dispositivo).
- **Attributes**: Las propiedades del sujeto, recurso y entorno que se evalúan para tomar decisiones de acceso.
- **Policies**: Un conjunto de reglas que definen las condiciones bajo las cuales se concede o se niega el acceso.

Las políticas ABAC son más complejas que los modelos de control de acceso tradicionales como Control de acceso basado en roles (Role-based access control, RBAC) . Por otro lado, ABAC proporciona más flexibilidad y granularidad en las decisiones de control de acceso.

# Ejemplo de políticas ABAC

Por ejemplo, un sistema tiene varias políticas ABAC:

1. **Política 1**: Permitir el acceso si:
   - (Subject) El rol del sujeto es `manager`.
   - (Attribute) El nivel de sensibilidad del recurso es `high`.
   - (Environment) La ubicación es `internal`.
   - (Action) Cualquier acción.
   - (Environment) El tiempo es entre las 9 AM y las 5 PM (horas de oficina).
2. **Política 2**: Denegar el acceso si:
   - (Subject) El rol del sujeto no es `manager`.
   - (Attribute) El nivel de sensibilidad del recurso es `high`.
   - (Environment) Cualquier ubicación.
   - (Action) Cualquier acción.
   - (Environment) Cualquier momento.
3. **Política 3**: Permitir el acceso si:
   - (Subject) El rol del sujeto es `employee` o `manager`.
   - (Attribute) El nivel de sensibilidad del recurso es `low`.
   - (Environment) Cualquier ubicación.
   - (Action) Acción `read`.
   - (Environment) Cualquier momento.

El motor de evaluación de políticas verificará estas políticas en orden, y la primera política que coincida con las condiciones determinará la decisión de acceso. Mientras tanto, se aplica una política de denegación predeterminada si ninguna otra política coincide.

Veamos algunos escenarios para entender cómo funciona el ABAC:

> **Escenario 1**. Un usuario quiere acceder (realizar la acción `read`) a un documento de alto nivel de sensibilidad (recurso) fuera de la oficina (entorno). El usuario tiene el rol de `manager` almacenado en el sistema.

**Decisión**: El acceso se niega porque el usuario está fuera de la oficina (la ubicación no es `internal`).

> **Escenario 2**. Un usuario quiere acceder (realizar la acción `read`) a un documento de alto nivel de sensibilidad (recurso) durante las horas de oficina (entorno) en la red de la oficina (ubicación=`internal`). El usuario tiene el rol de `manager`.

**Decisión**: El acceso se concede porque se cumplen todas las condiciones de la Política 1.

> **Escenario 3**. Todas las condiciones en el escenario 2 son las mismas, pero el usuario tiene el rol de `employee` en lugar de `manager`.

**Decisión**: El acceso se niega porque el rol del usuario no coincide con las condiciones de la Política 1.

> **Escenario 4**. Un usuario quiere acceder (realizar la acción `read`) a un documento de bajo nivel de sensibilidad (recurso). El usuario tiene el rol de `employee`.

**Decisión**: El acceso se concede porque se cumplen todas las condiciones de la Política 3.

> **Escenario 5**. Un usuario quiere eliminar (realizar la acción `delete`) un documento de bajo nivel de sensibilidad (recurso). El usuario tiene el rol de `employee`.

**Decisión**: El acceso se niega porque no hay ninguna política que permita la acción `delete` en documentos de bajo nivel de sensibilidad.

Podrás notar que no se requieren todos los atributos en cada política. Esta flexibilidad permite un mecanismo de control de acceso más dinámico y consciente del contexto.

# Lenguaje Extensible de Control de Acceso (XACML)

XACML es un estándar basado en XML para expresar políticas de control de acceso. Aunque no define un modelo específico de control de acceso, XACML se utiliza a menudo para implementar políticas ABAC. Veamos un ejemplo no normativo de cómo se puede usar XACML para representar las políticas ABAC del ejemplo anterior:

```
<PolicySet PolicySetId="ABAC_Policies" PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides">
  <Description>ABAC Policies</Description>
  <Policy PolicyId="Policy1" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides">
    <Description>Employees can read data</Description>
    <Target>
      <AnyOf>
        <AllOf>
```

```
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <AttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
          DataType="http://www.w3.org/2001/XMLSchema#string"
          MustBePresent="true"
        />
      </Match>
    </AllOf>
  </AnyOf>
</Target>
<Rule RuleId="Rule1" Effect="Permit">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">employee</AttributeValue>
          <AttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            MustBePresent="true"
          />
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</Rule>
<Rule RuleId="Rule2" Effect="Deny">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">user</AttributeValue>
          <AttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#string"
```

```
            MustBePresent="true"
          />
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
 </Rule>
</Policy>
 <!-- ...other policies... -->
</PolicySet>
```

# Consideraciones de implementación

Si bien el ABAC ofrece una forma poderosa de gestionar el control de acceso, también presenta algunas consideraciones de implementación:

- Complejidad del sistema: Las políticas ABAC pueden volverse complejas a medida que aumenta el número de atributos y reglas. La gestión y prueba adecuada de políticas son más laboriosas que los modelos de control de acceso más simples.
- Rendimiento: Evaluar políticas ABAC complejas puede afectar el rendimiento del sistema. Las técnicas de almacenamiento en caché y optimización pueden ayudar a mitigar este problema.
- Conflictos de políticas: Las políticas en conflicto pueden llevar a decisiones de acceso impredecibles. La revisión periódica de políticas y la resolución de conflictos deben ser parte del proceso de gestión de políticas.

# ABAC vs. RBAC

Comparar ABAC con Control de acceso basado en roles (Role-based access control, RBAC) puede ayudarte a comprender las diferencias entre los dos modelos:

| | **RBAC** | **ABAC** |
|---|---|---|
| Política de control de acceso | Basada en roles | Basada en atributos |
| Granularidad | Gruesa | Fina |
| Flexibilidad | Limitada | Muy flexible |
| Complejidad | Más simple | Más compleja |

| | RBAC | ABAC |
|---|---|---|
| Impacto en el rendimiento | Mínimo | Puede ser significativo |
| Gestión de acceso | Gestión de roles | Gestión de políticas |
| Mejor para | Estructuras de permisos bien definidas | Control de acceso dinámico y consciente del contexto |