Beyond Access
Using ABAC frameworks to implement privacy and security policies
A brief history of access control

<table>
<thead>
<tr>
<th>Physical</th>
<th>Identity</th>
<th>Role</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access is controlled by a physical attribute of a device or piece of physical media.</td>
<td>Access is a function of who is requesting access to a resource.</td>
<td>Access is a function of the user's role(s), and authorization associated with those roles.</td>
<td>Access is a function of attributes of the user, the data, source and destinations, the nature or purpose of a computation, and other policy constraints.</td>
</tr>
<tr>
<td>Example: write-protect tabs on a floppy disk</td>
<td>Example: UNIX® permissions, traditional ACLs.</td>
<td>Examples: RBAC, group membership tests</td>
<td></td>
</tr>
</tbody>
</table>
Attribute Based Access Control

- **First appeared in 2000**

- **Multiple standards!**
  - OASIS XACML (2001)
  - NIST SP 800-162 (2014)
  - Microsoft SDDL
  - ... and so forth

- **Common groups of attributes:**
  - Subject
  - Action
  - Object
  - Context
Two general approaches

**Object attributes**

Pros:
- Fast: rules can be evaluated inline
- Transparent: rule is explicit at the policy enforcement point

Cons:
- Decentralized: rule updates can be painful
- Not every useful attribute is static

**Policy service**

Pros:
- Centralized: rules can be updated and take effect immediately
- Dynamic attributes can be computed or looked up on demand

Cons:
- Much slower
- Service must be reachable
Access is a special case
... of "should this computation proceed?"
... or "should this computation include this data?"

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Jurisdiction</th>
<th>Public Policy</th>
<th>Internal Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>User interaction</td>
<td>Different countries have different rules</td>
<td>&quot;Public&quot; does not mean unconstrained (see RTBF)</td>
<td>&quot;Keeping honest people honest&quot;</td>
</tr>
<tr>
<td>Personalization</td>
<td>Even jurisdiction is a function, not a static</td>
<td>Users have expectations even around &quot;public&quot;</td>
<td></td>
</tr>
<tr>
<td>Monetization</td>
<td>attribute</td>
<td>information about them</td>
<td></td>
</tr>
<tr>
<td>Research</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security / anti-abuse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ancilliary uses</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Think beyond access

- Many privacy policies are not about who can access what, they are about purposes and jurisdiction.
- Expectations, agreements, and regulations change. A layer of indirection makes it much easier to adapt.
- Ask why: access rules (especially RBAC) are often proxies for more abstract rules that can be computed dynamically.
- If you can't write a piece of code to evaluate a policy, ask:
  - What information is missing?
  - Can it be recorded somewhere a policy service can read it?
  - Do other systems expose state that affects whether a policy applies?
Thank You.

Amanda Walker  VP, Engineering  awalker@nuna.com