Approved for Public Release: Distribution Unlimited
Case # 03-0214

Toward Scalable Exchange of Security Information

Arnon Rosenthal
The MITRE Corporation arnie@mitre.org

Abstract

One way to simplify distributed systems security is to have a unified interface for requesting attributes that security policies need. We explore requirements for getting attributes, and examine issues of federated data, trust, and confidentiality. Our goal is to insulate most participants from exactly how and where these attributes are accessed, and to encourage implementation of each feature at the appropriate level of generality.

Introduction

Today's systems were designed to apply security policies in a relatively fixed, well known environment. In many government systems, policies are executed by what we call policy *deciders*, which get security-relevant information (*attributes* of the user request) from multiple providers. For example, authenticated user identity and submission site might be in the request object, submission time come from an application server, clearances from a directory, approval signatures on individual requests from an approvals database. Furthermore, each decider partners with systems it knows well, so it can rely on their veracity and confidentiality. The tight coupling inhibits scalability and change. It becomes very hard to accept requests from other application servers, or access new attributes, or switch an attribute (e.g., clearance) from on-demand to "as part of the request message".

Our contribution is to identify requirements and opportunities for general solutions, especially in areas where individual communities (e.g., PKI) are exploring narrower services. Specifically:

- We point out the benefits of passing *all* needed attributes through a single interface, and how we now fall short.
- We propose a simple, decentralized, way to manage an extensible set of attribute types.
- We suggest that trust and confidentiality services apply comprehensively, but be modularized (at least from the administrators' viewpoint).

The basic functionality for getting attributes exists today – split among several interfaces. The policy decider software (and policy writers) should see a *unified* interface for getting arbitrary *attributes*, insulating them from how these attributes are provided. This interface should also hide whether an attribute was predefined in a standard, or belongs to one provider's extension. In essence, we want a *comprehensive* virtual *federated* directory.

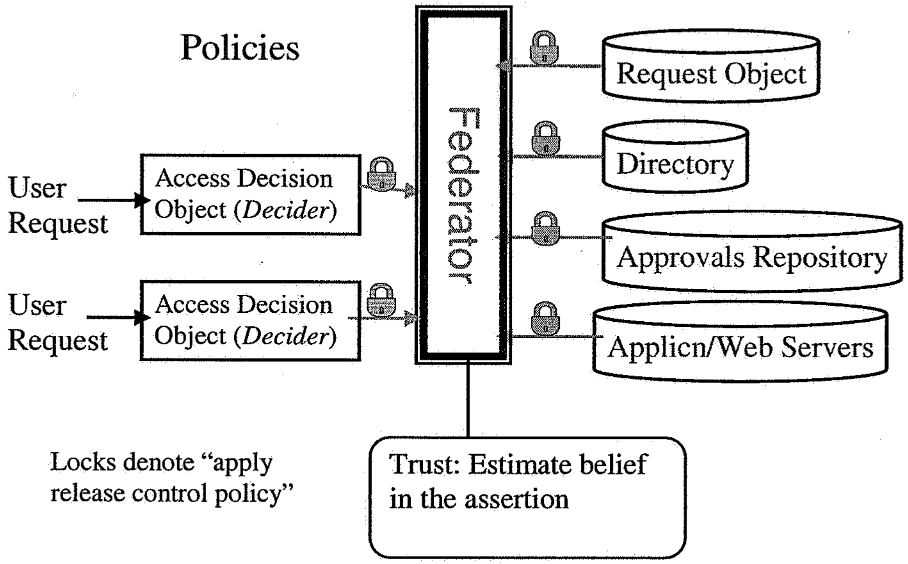
For concreteness, imagine a command get(requestObject, AttribSet)[1]. The figure below contains an architecture sketch.

We next discuss management requirements for a federation. Afterwards, we seek simpler policy specification, by pushing attribute trust and confidentiality issues away from administrators. Our goal is to encourage developers to use simple techniques more broadly and modularly, rather than to propose great novelties.

Federation Issues: We need a stable structure for organizing the attribute set, so providers can describe what they offer and deciders can describe what they need. Yet every participant must be

[1] LDAP and SAML both offer commands to get a list of attributes. LDAP seems a good choice for exploiting existing skills, and within an enclave where the decider trusts all attributes reported by servers.

able to extend the set of attribute types – otherwise deciders will need separate mechanisms for accessing standard versus nonstandard ones. We propose a simple compromise between static versus chaotic. Take a "standard" logical partition of the attribute space, and within each locality provide a "free" space to which anyone can add (as described below).


Policies
User Request
Access Decision Object (*Decider*)
User Request
Access Decision Object (*Decider*)
Federator
Request Object
Directory
Approvals Repository
Applicn/Web Servers
Locks denote "apply release control policy"
Trust: Estimate belief in the assertion

One might initially simplify the system move code that understands the providers from deciders to a hard coded federator. Federated directories would enable new attribute types to simply be registered, without changing code. Eventually, federated database query processors may provide powerful, self-managing optimization.

Example Implementation Approach for Managing the Attribute Types: Use a labeled tree of types, as in LDAP (or XML), to be instantiated as a tree. Nonterminals will be called *folders*. For example, the top level Request might have child types describing the request's User, Submission, and Approvals. Request.User.Clearance is a terminal, which takes on a value for each request. As in LDAP, the folder tree describes logical relationships. Physical layout will be beyond our scope; a wide variety of techniques are available.

To balance control and chaos, let *every* folder type include a distinguished subfolder, denoted *Freespace*, where any stakeholder can insert new child types (and become that child type's default administrator). This removes the temptation to go outside the system. It also avoids creating a single giant uncontrolled area. It encourages putting new nodes in roughly the right place, and scanning what's already there to see if it can be reused.[2] Occasionally, "standard" node administrators may promote definitions from Freespace.

Trust and Confidentiality for Security Information: This section discusses opportunities to make (at least some) approaches in the literature more modular and more comprehensive.

Modularity: We should seize the opportunity to simplify policy specification by insulating subject matter experts from issues of system trustworthiness. An ethicist who judges appropriateness of researchers seeing medical data should not be concerned with trustworthiness

[2] Note that only those interested in the new type will instantiate it in their folder instances. For now, we assume each type appears once; a fuller facility might provide templates or crosslinks.

of different PKI roots, or legitimacy of delegation steps. Put another way, the ethicist's policy should see attribute assertions and an estimate of their trustworthiness; trust arbiters are separate from substantive policy. (We optimized for the normal, simple case; more detail can always be provided as additional attributes).

Splitting trust (i.e., attribute belief estimates) from substantive policy offers other benefits. By omitting path-tracing (recursion), one may be able to give most users a simpler policy language. Also, a single trust estimator is likely to be used by many objects' policies. Finally, estimation of believability will not be confined to security-relevant assertions; for example, an intelligence-evaluation system might also use it. One can argue that modularity is surface, not foundational, but so are a building's doors and windows.

Comprehensiveness: Trust management, negotiation, and confidentiality go beyond just authentication, to all security information, and beyond. Attribute Based Access Control formalisms described in [Se, WL] allow the desired breadth. Their release policy frameworks seems to apply to all data – it is only the decision about the next step that seems specifically about negotiations and security information[3].

We will mention two challenges. First, the "objects" of a policy negotiation tend to be fine grained – release of a string, or of an (object, attribute, value) triple. This is a poor match for servers that protect tables, files, or methods, or documents. Second, release policies must be applied after the Federator has identified the attribute source.

Final Comments: We identified opportunities to exchange security metadata more flexibly. We discussed the need to insulate each player from the others, in terms of attribute sets, trust, and confidentiality. We want to be comprehensive: to cover *all* the attributes needed, not separately for directories, application servers, request objects, nonstandard attributes, etc.

Our hope is to improve this aspect of large government systems – incrementally, by developers, without giant initiatives or committees of agency heads. We hope to structure incentives (e.g., documentation and interface requirements) to make it cheaper for a decider to go to the getAttributes interface than to all providers individually. Initial steps may be within single trust domains (i.e., all servers are trusted). Trust policies would then be added modularly.

References

[Ro94] A. Rosenthal, J. Williams, W. Herndon, B. Thuraisingham, "A Fine-Grained Access Control Model for Object-oriented DBMSs", *IFIP 11.3 Conf on Database Security*, 1994.

[Se01] K. Seamons et. al., "Requirements for Policy Languages for Trust Negotiation", *IEEE Workshop on Policies for Distributed Systems and Networks, 2001. pp 68-79*

[WL01}, W. Winsborough, N. Li, "Towards Practical Automated Trust Negotiation", *IEEE Workshop on Policies for Distributed Systems and Networks, 2001. pp 92-103*

[3] Attribute acknowledgement policy" constructs may be unneeded, since secure database models (e.g., [Ro]) give unauthorized users no hint about an object's existence. [WL] explores undesired inference; however, attribute negotiations may not justify an industrial-strength solution to this notorious problem.