Proposed GENIAPI AM Changes to Support ABAC Authorization

Ted Faber, USC/ISI

John Wroclawski, USC/ISI

Version 1.0

31 October 2011

Corresponding to GENIAPI version 1.0

Table of Contents

1 Introduction

This document describes an initial integration of the Attribute-Based Access Control (ABAC) implementation from USC/ISI and Sparta (version 0.1.2)[1] with the current implementation of the GENIAPI AM (version 1.0)[2] from BBN. The implementation passes the GENIAPI tests included with the system, with minimal modification to the test scaffold to support a change in the unstandardized Clearinghouse interface. This document makes recommendations to the GENIAPI specification based on that implementation and describes the next steps in ABAC design motivated by furthering this kind of access control.

The current GENIAPI implementation provides an aggregate manager that makes resources available within the slice-based facility architecture of GENI. The interface between users (or their agents) and the aggregate manager is well-defined, though interfaces to other parts of the system (slice managers, clearinghouses) are still being standardized.

ABAC[3], which had its formal conceptual design done at Stanford with initial implementations and collaborations at Sparta (then Network Associates), is an access control system based on formal derivation of attributes possessed by principals making requests. These attributes and the rules about reasoning with them allow access control to proceed using the same kind of reasoning that real world decisions encompass. Proof of an attribute may involve multiple rounds of negotiation.

In ABAC, principals assign attributes to other principals. This is analogous to USC asserting that Ted Faber is an employee. Principals can also assert rules for deriving attributes from other attributes: if USC says someone is an employee the campus bookstore says they get a discount. The current ABAC library supports the RT0 logic, which allows multi-layer delegation and intersection of attributes as well as simple derivation.

Incorporating ABAC also separates access policy specification and access validation, provides enhanced logging of access control decisions by both the aggregate manager and the clients, and enables multi-round access control negotiation. ABAC access control requests are either satisfied by a proof of access rights in terms that both sides understand, or a partial proof of access that can be used as a basis for continued negotiations. Complete proofs can be logged as an audit trail.

While the GENIAPI interface leaves room in the specification for much of this integration to proceed, the interface must be tweaked in several important ways to get the best out of the more flexible and powerful ABAC system. Key parts of the interface need to be widened to support ABAC negotiation, support added for self-validating identities, and the user/agent to clearinghouse and aggregate manager to clearinghouse/slice manager interfaces need to be standardized.

In the GENIAPI, the general request/response exchange pattern tends to assume most requests succeed, resulting in very little information from failed requests. Once a request has failed, a user has little idea what changes to the request, if any, may result in success. To implement

ABAC's multi-round negotiation, the system must communicate the successful proof of access or alternatively a partial proof to act as the starting point for the next round.

2 Proposed GENIAPI Changes

The primary issues with adding ABAC to the current GENIAPI are the request/response model and limited semantics of the return values, and the imposition of hierarchical names. We also comment on the importance of separating access control attributes and object names in coming development of the specifications.

2.1 Return Values

Though ABAC can require multiple rounds of negotiation and the GENIAPI supports request/response interactions, it is possible to support the extended negotiation of the former in the simple model of the later, if the GENIAPI response to a denied request includes enough information. In practice this means always returning a structure with error codes and more access control data, including new credentials and ABAC proofs/partial proofs. This allows future multi-round negotiation because ABAC captures the entire state of the negotiation in its proofs/partial proofs. It also allows current implementations to log ABAC proofs of success and failure.

A general ABAC exchange consists of several messages negotiating access to a resource where each principal reveals a little more about their attributes until each knows enough to agree that the access is valid. We show such an exchange in Figure 1.

Simple exchanges between principals who have little information to hide look like conventional request/response exchanges that the GENIAPI supports. Longer exchanges can be carried out in the request/response paradigm as long as successive messages requesting or denying the access provide the other player with new information to advance the negotiation.

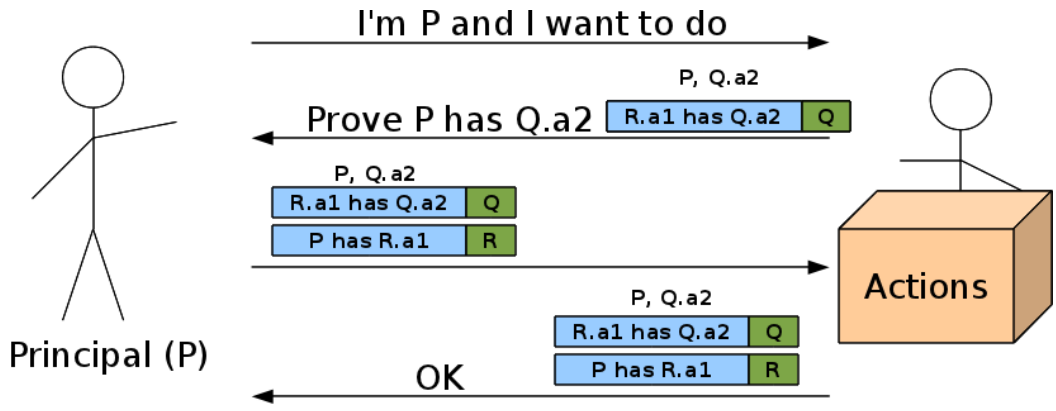

Figure 1: An ABAC exchange

An ABAC partial proof contains the entire state of the negotiation. Passing the partial proof back as a return value in a failed request allows the recipient to extend the proof and try again. The extended proof contains the complete state for the other side to continue the negotiation. The result of a series of denied accesses, extensions, and retries constitutes a negotiation.

In principle, such exchanges are easily integrated with the GENIAPI, but in practice the responses do not carry enough information to support a negotiation. In many cases there is not enough information returned to support detailed auditing information.

Many requests include parameters of sets of attribute value pairs, for example the options field of the `ListResources` operation, which provides an extensible interface. Return values should have a similar structure that includes ABAC partial proofs. Where such fields already exist, current code appends that information.

The current reference AM expresses access control failures as XMLRPC faults, which have limited message capacity. To support ABAC, encode all access denials in the same extensible format, including error code and ABAC partial proofs. Faults are too format constrained for this purpose. We believe XMLRPC faults should only be used to communicate low-level protocol errors.

Though this change is motivated by adding ABAC, there are other good reasons to add multi-valued return codes, and other parties are proposing extended error codes as well. Some failures can be retried for reasons other than presenting insufficient access controls – e.g., a busy internal resource. Other failures could include hints for a modifying subsequent requests, for example, requesting less constrained resources or lower bandwidth. Providing such iterative interfaces benefits the API in many ways going forward.

2.2 Proposed Structures

We propose the following return structures:

- ListResources: a three-element structure
    - code: a success or failure code
    - manifest: the Rspec as a string, or an empty string on failure
    - proof: a binary encoding the ABAC proof or partial proof
- CreateSliver: a three element structure:
    - code: a success or failure code
    - manifest: the Rspec representing allocated resources as a string, or an empty string on failure
    - proof: a binary encoding the ABAC proof or partial proof
    - credentials: new credentials for the user as a list of strings or binary objects (ABAC credentials would be binary)
- SliverStatus: a five element structure

  - code: a success or failure code for the operation
  - geni_urn: a string containing the sliver URN
  - geni_status: a string with the status
  - geni_resources: a structure as described in the current specification
  - proof: a binary encoding the ABAC proof or partial proof
- RenewSliver: a two-element structure
  - code: a success or failure code
  - proof: a binary encoding the ABAC proof or partial proof
  - credentials: new credentials for the user as a list of strings or binary objects (ABAC credentials would be binary)
- DeleteSliver: a two-element structure
  - code: a success or failure code
  - proof: a binary encoding the ABAC proof or partial proof
- Shutdown: a two-element structure
  - code: a success or failure code
  - proof: a binary encoding the ABAC proof or partial proof

3 Summary

This document summarizes design changes necessary to integrate ABAC with the GENIAPI. Other changes are being suggested from other quarters.

References

[1]LibABAC Home Page, *http://abac.deterlab.net*, 2011.

[2]GCF Project Home Page, *http://trac.gpolab.bbn.com*, 2011.

[3]Ninghui Li, John C. Mitchell, and William H. Winsborough, "Design of a Role-Based Trust Management System," *in Proceedings of the 2002 IEEE Symposium on Security and Privacy*, (May, 2002).