

ARTECH HOUSE

INFORMATION SECURITY AND PRIVACY SERIES

Attribute-Based Access Control

VINCENT C. HU • DAVID F. FERRARIO • RAMASWAMY CHANDRANLU • D. RICHARD KUHN

Attribute-Based Access Control

For a complete listing of titles in the *Artech House Information Security and Privacy Series*, turn to the back of this book.

Attribute-Based Access Control

Vincent C. Hu

David F. Ferraiolo

Ramaswamy Chandramouli

D. Richard Kuhn



**ARTECH
HOUSE**

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library.

Cover design by John Gomes

ISBN 13: 978-1-63081-134-1

© 2018 ARTECH HOUSE

685 Canton Street

Norwood, MA 02062

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

10 9 8 7 6 5 4 3 2 1

Contents

Preface

Acknowledgments

Intended Audience

1 Introduction

1.1 Overview

1.2 Evolution and Brief History of Access Control

1.2.1 Academic Contributions

1.2.2 Military Concerns

1.2.3 Bell and LaPadula Security Model

1.2.4 Trusted Computer Security Evaluation Criteria

1.2.5 Discontent

1.2.6 Role-based Access Control

1.2.7 Emergence of ABAC

References

2 Access Control Models and Approaches

2.1 Introduction

2.2 Terminology

2.3 Access Control Models and Policies

2.4 Policy Enforcement

2.5 Discretionary Access Control

2.6 Mandatory Access Control Models

2.6.1 Multilevel Security

2.6.2 Chinese Wall Policy and Model

2.6.3 Role-Based Access Control

References

3 ABAC Models and Approaches

- 3.1 Introduction
 - 3.2 ABAC Architectures and Functional Components
 - 3.3 Logical-Formula and Enumerated ABAC Policy Models
 - 3.4 ABAC Model—Applications Primatives
 - 3.5 Hierarchical Group and Attribute-Based Access Control
 - 3.6 Label-Based ABAC Model with Enumerated Authorization Policy
 - 3.7 Hybrid Designs Combining Attributes with Roles
 - 3.8 ABAC and RBAC Hybrid Models
 - 3.9 Complexities of RBAC Role Structures
 - 3.10 Complexities of ABAC Rule Sets
 - 3.11 Dynamic Roles
 - 3.12 Role Centric Structure
 - 3.13 Attribute-Centric Structure
 - 3.14 Conclusion
- References

4 ABAC Deployment Using XACML

- 4.1 Introduction
- 4.2 Business and Technical Drivers for XACML
- 4.3 XACML Standard—Components and Their Interactions
 - 4.3.1 XACML Policy Language Model
 - 4.3.2 XACML Context (Request and Response)
 - 4.3.3 XACML Framework (Data Flow Model)
- 4.4 ABAC Deployment Using XACML
 - 4.4.1 Access Policy Formulation and Encoding

- 4.4.2 Request/ Response Formulation
- 4.4.3 Policy Evaluation and Access Decision
- 4.5 Implementation of XACML Framework
- 4.5.1 Attribute Support and Management
- 4.5.2 Delegation
- 4.6 Review and Analysis

References

Appendix 4.A

5 Next Generation Access Control

- 5.1 Introduction
- 5.2 Policy and Attribute Elements
- 5.3 Relations
 - 5.3.1 Assignments and Associations
 - 5.3.2 Prohibitions (Denials)
 - 5.3.3 Obligations
- 5.4 NGAC Decision Function
- 5.5 Delegation of Access Rights
- 5.6 NGAC Administrative Commands and Routines
- 5.7 Arbitrary Data Service Operations
- 5.8 NGAC Functional Architecture
 - 5.8.1 Resource Access
 - 5.8.2 Administrative Access
- 5.9 Conclusion

References

6 ABAC Policy Verifications and Testing

- 6.1 Introduction
- 6.2 ABAC Policy Classes
 - 6.2.1 Static Policy Class

- 6.2.2 Dynamic Policy Class
- 6.2.3 Historical Policy Class
- 6.3 Access Control Safety and Faults
- 6.4 Verification Approaches
 - 6.4.1 Model Verification
 - 6.4.2 Coverage and Confinements Semantic Faults
 - 6.4.3 Property Confinement Checking
 - 6.4.4 Implementation Test
- 6.5 Implementation Considerations
- 6.6 Verification Tools
 - 6.6.1 Multiterminal Binary Decision Diagrams
 - 6.6.2 ACPT
 - 6.6.3 Formal Methods
- 6.7 Conclusion
- References

7 Attribute Consideration

- 7.1 Introduction
- 7.2 ABAC Attributes
- 7.3 Consideration Elements
- 7.4 Preparation Consideration
 - 7.4.1 Subject Attribute Preparation
 - 7.4.2 Object Attribute Preparation
 - 7.4.3 Environment Condition Preparation
 - 7.4.4 Metadata
- 7.5 Veracity Consideration
 - 7.5.1 Attribute Trustworthiness
 - 7.5.2 Attribute Value Accuracy
- 7.6 Security Consideration
 - 7.6.1 Attribute-at-Rest
 - 7.6.2 Attribute-in-Transit

- 7.7 Readiness Consideration
 - 7.7.1 Refresh
 - 7.7.2 Synchronization
 - 7.7.3 Cache
 - 7.7.4 Backup
 - 7.7.5 Log
 - 7.8 An Example of a General Attribute Framework
 - 7.9 Attribute Evaluation Scheme
 - 7.9.1 AES Examples
 - 7.9.2 Attribute Practice Statement
 - 7.10 Conclusion
- References

8 Deployments in Application Architectures

- 8.1 Introduction
- 8.2 ABAC for Distributed Systems
 - 8.2.1 Access Control Challenges of Distributed Systems
 - 8.2.2 BigData Access Control as a Distributed System Access Control Example
 - 8.2.3 Implementation Considerations
 - 8.2.4 Analysis and Conclusions
- 8.3 ABAC for Web Services
 - 8.3.1 Web Services— A Brief Background
 - 8.3.2 ABAC Suitability for Web Service Environments
 - 8.3.3 ABAC for Web Service Environments without Workflows
 - 8.3.4 ABAC for Web Service Environments with Workflows
 - 8.3.5 Combined Challenges in Using ABAC for Web Service Environments (with and without Workflows)
 - 8.3.6 Web Services Environment—Summary of Requirements
- 8.4 ABAC for Stand-Alone Workflow Processes
 - 8.4.1 Challenges and Requirements for ABAC Configuration for Stand-Alone Workflow Processes

8.4.2 ABAC Deployment for Stand-Alone Workflow Processes:
Integrated Approach

8.4.3 ABAC Deployment for Stand-Alone Workflow Processes:
Loosely Coupled Approach

8.4.4 Analysis and Conclusions

References

9 ABAC Life-Cycle Issues: Considerations

9.1 Introduction

9.2 Enterprise ABAC Concepts

9.2.1 Enterprise ABAC Policy

9.2.2 Attribute Management in Enterprise ABAC

9.2.3 Access Control Mechanism Distribution in Enterprise ABAC

9.3 ABAC Enterprise Considerations

9.3.1 Initiation Phase Considerations

9.3.2 Acquisition/Development Phase Considerations

9.3.3 Implementation/Assessment Phase Considerations

9.3.4 Operations/Maintenance Phase Considerations

9.4 Conclusion

References

10 ABAC in Commercial Products

10.1 Introduction

10.2 Axiomatics Data Access Filter

10.2.1 Product Architecture and Modules

10.2.2 Canonical Features in Product Modules

10.3 Jericho Systems EnterSpace 9

10.3.1 Product Architecture and Modules

10.3.2 Canonical Features in Product Modules

10.4 NextLabs ABAC Solution

10.4.1 Functional Architecture and Components

10.4.2 Canonical Features in Product Modules

References

11 Open Source ABAC Implementations: Architecture and Features

11.1 Introduction

11.2 NGAC PM: Functional Architecture

11.3 NGAC PM: ABAC Model Definition Capabilities

11.4 NGAC PM: Access Decision Process

11.5 NGAC PM: Design and Application Integration

11.6 Summary and Analysis

References

About the Authors

Index

Preface

Attribute-based access control (ABAC) is the latest development in an evolution of access control models going back more than 40 years. As with other models, it solves problems encountered in the practical application of access control solutions in a changing information technology environment. Early computing systems used simple access control lists (ACLs) of user IDs attached to each resource. As the number of resources and users multiplied into the tens or hundreds of thousands, setting up and managing ACLs became cumbersome and time-consuming. Role-based access control (RBAC) solved many of these problems by collecting permissions into roles that usually corresponded to user positions in an organization, and permitting access only through roles. This eliminated the need to tie individual user permissions to every resource, dramatically reducing the complexity of security administration. But RBAC's ease of management comes at a tradeoff with the cost of initial setup, which many organizations found to be challenging and time-consuming. RBAC also assumes that security is managed by a single organization or cooperating group of organizations, which is not always the case with today's vast Web-based applications.

An alternative is to grant or deny user requests based on attributes of the user and the object, and environment conditions that may be globally recognized and relevant to the policies at hand. For example, access to company facilities may be granted only if a subject has a company badge and the current time is within working hours. This approach is the essence of ABAC, where authorization to perform a set of operations is determined by evaluating attributes associated with the subject, object, requested operations, and, in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes.

The access control policies that can be implemented in ABAC are limited only by the expressiveness of representational language and the integrity of the available attributes. This flexibility enables the greatest

breadth of subjects to access the greatest breadth of objects without specifying individual relationships between each subject and each object. For example, a subject is assigned a set of subject attributes upon employment (e.g., Bob Smith is over 25 years of age, is licensed as a truck driver and a mechanic, and is assigned to the shipping department). An object is assigned its object attributes upon creation (e.g., a store of parts belonging to the shipping department). Objects may receive their attributes either directly from the creator or as a result of automated scanning tools. Security administrators create access control rules using attributes of subjects and objects to govern the set of allowable capabilities (e.g., only mechanics are permitted to requisition parts from company storage), making ABAC easy to set up. Access decisions can change between requests by simply changing attribute values, without the need to change the subject/ object relationships defining underlying rule sets. This provides a more dynamic access control management capability and limits long-term maintenance requirements of object protections. Further, ABAC enables object owners or administrators to apply access control policies without prior knowledge of the specific subject and for an unlimited number of subjects that might require access. A new subject assigned the attributes value mechanic automatically gets the permission to request parts from company storage. This benefit is often referred to as accommodating the external (unanticipated) user, and is one of the primary benefits of employing ABAC.

In recent years, attribute-based access control (ABAC) has evolved as the preferred logical access control methodology for many information systems. Vendors have begun implementing ABAC in a variety of products, and organizations are employing ABAC features to simplify operations. ABAC is also capable of enforcing both discretionary access control (DAC) and mandatory access control (MAC) based protections, and can coexist with RBAC. ABAC enables precise access control, which allows for a higher number of discrete inputs into an access control decision, providing a bigger set of possible combinations of those variables in order to reflect a larger and more definitive set of possible rules to express policies that satisfy access control requirements of organizations. However, when deployed across an enterprise, ABAC implementations can become complex—supported by the existence of an attribute management infrastructure, machine-

enforceable policies, and an array of functions that support access decisions and policy enforcement.

Until now, research on ABAC models and applications is dispersed throughout hundreds of research papers, but not consolidated in book form. This book also contains (in addition to its new content) materials that are derived from existing documents written by the authors, such as NIST SP 800-162 and NIST 800-178 that explain ABAC's history and model, related standards, verification and assurance, applications, and deployment challenges (Part 5). Specialized topics—including formal ABAC history, ABAC's relationship with other access control models, ABAC model validation through analysis, verification and testing, deployment frameworks such as XACML, Next Generation Access Model (NGAC), attribute considerations in implementation, ABAC applications in Web services/workflow domains, and ABAC architectures and feature sets in commercial and open source products. The combination of technical and administrative information for models, standards, and products in the book is thus intended to benefit researchers as well as implementers of ABAC systems. Certain software products are identified in this book, but such identification does not imply recommendation by the U. S. National Institute for Standards and Technology, nor does it imply that the products identified are necessarily the best available for the intended purpose.

Acknowledgments

We offer special thanks to Arthur Friedman of the National Security Agency. Much of the authors' involvement in ABAC can be traced to Arthur's early vision, persistence, and support. We would also like to thank Isabel Van Wyk for her extensive editing of the various drafts of this book and Gerry Gebel of Axiomatics for his careful review and comments that have improved the presentation, especially regarding emerging technologies. Tim Weil and Ed Coyne were key thinkers in the ANSI/INCITS project to add attributes to role based access control. We also thank Wayne Jansen for his many contributions pertaining to modern day ABAC concepts. Serban Gavrila, Gopi Katwala, and Joshua Roberts demonstrated the viability of NGAC standard implementations. Richard Fernandez developed an early implementation of an RBAC/ABAC hybrid model. Bill Fisher contributed feedback and advice on ABAC in commercial products. Adam Schnitzer, Kenneth Sandlin, Alan J. Lang, Paul Jacob, and Dylan Yaga contributed to the NIST ABAC definitions, models, and considerations document. Kim Schaffer and Raghu Kacker reviewed original drafts. Anne Anderson and the XACML community developed early standards that are applied to ABAC.

Many researchers in ABAC have contributed to the field; among others, these include Tao Xie, Jeehyun Hwang, Antonios Gouglidis, Qinghua Li, Xin Jin, Ang Li, Jia Di, Eric Yuan, Jin Tong, Luigi Logrippo, Bin Xie, Ulrich Lang, Elisa Bertino, Vijay Atluri, Hai-bo Shen, Fan Hong, Jingwei Huang, Duminda Wijesekera, Tim Schmoyer, Mohammad Soleimani, Indrakshi Ray, Stephanie McVitty, and Don Graham.

Finally, we give special thanks to Ravi Sandhu, Ram Krishnan, Prosunjit Biswas, Daniel Servos, and Sylvia Osborn for their leadership in advancing the theory and visibility of ABAC.

Intended Audience

This book is designed to be useful to three groups of readers: (1) security professionals, technology managers, and users in industry, government, and military organizations, including system administrators responsible for security, policy officials, and technology officers; (2) software developers for database systems, enterprise management, security and cryptographic products; and (3) computer science and IT students and instructors.

1

Introduction

1.1 Overview

Access control—or authorization in computer systems—is the administrative and automated process of defining and limiting which system users and processes can perform which system operations on which system resources. Every organization typically has a unique set of policies that dictate the circumstances and conditions under which users are permitted access to resources. Access control mechanisms use components that work together to bring about policy-preserving access. These components include access control data for expressing policies and representing attributes, as well as a set of functions for trapping access requests, and for computing and enforcing access decisions over those requests in accordance with policies.

Access controls are implemented in many places and at different levels within an IT infrastructure. Access control is used in operating systems to protect files and directories; in database management systems to regulate access to tables, records, and fields; and to protect information managed by applications such as time and attendance, payroll processing, and health benefits management.

Regardless of where it is implemented, access control's primary objective is the enforcement of policies. Access control policies are sets of rules that, when implemented, provide a means of protecting resources. The ability to enforce access control policy can be of great economic and mission importance. Although access control is often specified in terms of limitations or protections, the ability of an organization to enforce access control policy is what ultimately enables the sharing of greater volumes of data and resources to a greater and more diverse user community.

Policy objectives are diverse and span the breadth of activities of the enterprise, governing such elements as individual privacy, selective access to proprietary information, national security protection, fraud prevention, data integrity, and conflicts of interest. These policies can be derived from laws and regulations, but can also stem from business culture and organizational tolerance for risk. Therefore, policies vary considerably from one institution to another. For example, the policies of a hospital will obviously differ dramatically from those of a financial institution or military agency. Policies pertaining to system resources are often dependent on administrators that create and manage access control information. This is especially true in a federated or collaborative environment where governance policies require different organizational entities to have different responsibilities for administering different aspects of policies and their dependent attributes. While policy enforcement may be application-and system-specific, enforcement is just as likely to pertain to policies of some organizational unit or even of global concern.

Controlling and managing access to sensitive data has been a subject of research for decades, and it continues today. Specific issues pertain to assured embodiment of the right policies, visualization and management of user and object attributes and policy expressions, usability, non-bypassable enforcement of policies, granularity of control, delegation of access rights, performance, and comprehensive and interoperable protection across system services.

Attribute-based access control (ABAC) represents the latest milestone in the evolution of authorization approaches. ABAC enables the granting or denying of user access requests based on designated attributes of users and resources as well as optionally on environmental conditions that may be locally or globally recognized and tailored to the policies at hand. It provides an attribute-based approach to accommodate a wide breadth of access control policies and simplifies access control management. ABAC is more than just another policy model, it is also a fundamental reworking of traditional access control into a form suited to the needs of the modern, distributed, and interconnected enterprise. ABAC is based on a flexible framework that can provide access control services for many different types of resources accessed by many different types of applications and users. The ABAC framework can

support policies of various types simultaneously while remaining manageable in the face of changing technology.

Before delving into specifics, it is important to understand and appreciate the evolutionary and historical events that have led to today's concept of ABAC. Although often portrayed as a straight line of technological advancements, this evolution, like many others, includes a series of setbacks and pitfalls. It also includes security principles that some suggest have been forgotten or simply ignored out of convenience. The historical and evolutionary events that comprise the remainder of this chapter are meant to justify, motivate, serve as a backdrop, and provide context by which to judge ABAC both in the present and future.

1.2 Evolution and Brief History of Access Control

Although the concept of access control in one form or another can be traced back over the decades, its need in computer systems was first acutely recognized in the late 1960s and early 1970s by both the academic and military communities in response to the emergence of time-sharing computer systems. These systems were the first to allow a large number of users, programs, and data to interact and coexist concurrently within a single computer. While time-sharing dramatically lowered the cost of computing, it also presented a need for controlled access to shared system resources.

1.2.1 Academic Contributions

In the late 1960s, computer science began to emerge as an academic discipline with numerous specializations associated with it. Uncontrolled access to system resources was not only considered a security and privacy issue but was counter to the science of predictable and verifiable behavior [1]. Lampson, as a member of a curriculum-design team, first introduced the notion of a protection system, which quickly evolved into his now well-known access matrix [2-3]. It includes three major components: a set of objects called X , a set of domains called D , and an access matrix called A . Objects were abstractly defined as describing the things in the system which have to be protected, to include files, processes, segments, and terminals. Domains were defined as the entities that have access to objects, with the essential property that a domain has

potentially different accesses than other domains, and objects can be shared between domains. The access of domains to objects is determined by the access matrix A . Its rows are labeled by unique and globally-recognized domain names, and its columns by unique and globally-recognized object names. An access matrix provides a representation of the access rights to an object for which a domain is authorized. [Figure 1.1](#) provides a simple illustration of an access matrix. Each row i of the matrix represents a domain, d_i , while each column represents an object, x_i , which may include a domain. Each entry, $a_{i,j}$, at the intersection of a row and column of the matrix, contains the set of access rights for the domain to the object. The relatively simple access matrix model can still successfully express a broad range of policies because it is based on a general form of an access rule (e.g., domain, access right, and object) and imposes little restriction on the rule itself.

As another feature of his protection system, Lampson introduced four rules for manipulating entries in the access matrix and, consequently, the propagation and rescinding of privileges (e.g., domain, access right, and object). These rules pertain to what can be considered administrative capabilities and provide semantics for creating, copying, adding, and removing access rights.

| | | Objects | | | | | |
|---------|-------|---------|-------|-------|-----|--|--|
| | | ... | x_i | x_j | ... | | |
| Domains | d_i | | | | | | |
| | d_j | | | | | | |
| | d_k | | | | | | |
| | d_l | | | | | | |

The diagram shows a grid representing an access matrix. The columns are labeled with objects: ..., x_i , x_j , The rows are labeled with domains: ..., d_i , d_j , d_k , d_l , The entry in the i -th row and j -th column is labeled a_{ij} . A vertical line points from the label a_{ij} down to the cell in the grid, and a horizontal line points from the label a_{ij} right to the cell in the grid.

Figure 1.1 An access matrix.

Domains do not typically require access rights to most objects, so the matrix is often sparse. Several more space-efficient representations were suggested as alternatives. An authorization relation, for example, represents an access matrix as a list of triples of the form $(d_i, a_{i,j}, x_j)$. Each triple represents the access rights of a domain to an object.

Access control and capability lists were suggested as two other forms of representation. An access control list (ACL) is associated with each object in the matrix and corresponds to a column of the access matrix. Similarly, a capability list is associated with each domain and corresponds to a row of the matrix. ACLs and capability lists are discussed further in [Chapter 2](#) as a means of implementing discretionary access control (DAC). An important aspect is their ability to review and easily manage policy on a per object or per domain basis.

Reformulating and extending Lampson's work to include the use of the now familiar term subject in lieu of domains, Graham and Denning [4] went further in specifying operations in the form of well-structured commands for manipulating the access matrix. These commands not only provided methods for manipulating entries in the matrix, as Lampson did, but also provided methods and associated authorizations for creation and deletion of rows (subjects) and columns (objects) of the matrix. Graham and Denning also added a degree of rigor to their exposition by casting protection in the form of a state machine. All information specifying the types of access subjects have to objects constitutes a protection state of the system and is represented by the access matrix. Subject accesses to objects are only permitted by the matrix, and a subject's ability to alter the matrix is only conducted in certain ways and embodied by the matrix.

Graham and Denning's work provided a solid foundation for Harrison, Ruzzo, and Ullman [5] to develop a formal proof that tracking privilege propagation is undecidable in general. If the system is started with a set of access rights to objects, it is impossible to assert that the system will not eventually grant access rights that are not in the original matrix. Although the proof of this result is somewhat technical, the underlying rationale is that subjects can give away access rights at their discretion. If the system has no control over what rights are passed from one user to another, there is no way to be sure that an unauthorized subject will not eventually receive access rights improperly through some chain of rights delegation.

1.2.2 Military Concerns

In the 1960s, the U.S. military's reliance on time-sharing computing increased. The Department of Defense's Advanced Research Projects Agency (DARPA) supported early time-share work to include funding for MIT's Multiple Access Computer. At the same time, the U.S. defense sector realized that time-sharing computing systems posed serious security risks. While efforts on the part of universities would focus on devising controls to allow the granting of permissions to read and write files at a user's discretion, the military had an entirely different view of security, which stemmed from well-documented mandatory policies surrounding the handling of classified information. Much of the concern for protecting classified information came from top actors at the National Security Agency (NSA) with national responsibilities for protecting this information.

In response to these concerns, a task force was set up in 1967 with representations from the NSA, Department of Defense (DoD), defense contractors, and academia to address the security problem that resulted in a RAND Corporation report in 1970 [6]. The report provided a comprehensive analysis and design of security for the DoD computer systems. Included in the report was the definition of a method to implement multilevel access control (documents classified by security levels: confidential, secret, top secret, etc.) on a resource-sharing system with separate considerations for local access and remote access where a password-based authorization would be required. This document also discussed the basic requirements for controlling access to information based on a user's clearance level and the classification level of files stored on the system. The controls closely mirrored rules that are imposed on documents in the all-paper world with a focus on what would be referred to later as the no-read-up rule, which is simply the computer implementation of conventional military classified document rules. For example, in accordance with this rule, a user with a secret clearance would be permitted to read confidential and secret information (at or below the user's clearance level), but not top-secret (above the user's clearance level).

Although significant in highlighting the problems facing the military, the RAND report, in the eyes of the Electronic Systems Division (ESD) of the U.S. Air Force, offered few actionable solutions. Thus, an

additional study was commissioned by ESD and led by Anderson, a well-respected computer consultant. The study, now famously referred to as the Anderson report, was delivered in October 1972—only eight months from commission [7]. It incorporated three important design principles that were first espoused by Rodger Schell of ESD: the reference monitor, security kernel, and policy model.

The reference monitor (see [Figure 1.2](#)) is a concept for achieving execution control over user programs. Its function is to validate all references made by programs in execution against those authorized for a subject. Conceptually, the reference monitor represents the hardware and software portion of an operating system that is responsible for the enforcement of the security policy of the system. When some subject attempts to perform an operation (e.g., read or write) on an object, the reference monitor must perform a check to compare the attributes of the subject with that of the object.

The requirements of a reference monitor are comprised of three fundamental implementation principles, described as follows:

- *Completeness*: It must be always invoked and impossible to bypass.
- *Isolation*: It must be tamper-proof.
- *Verifiability*: It must be small enough to be subject to analysis and testing to ensure that it is properly implemented.

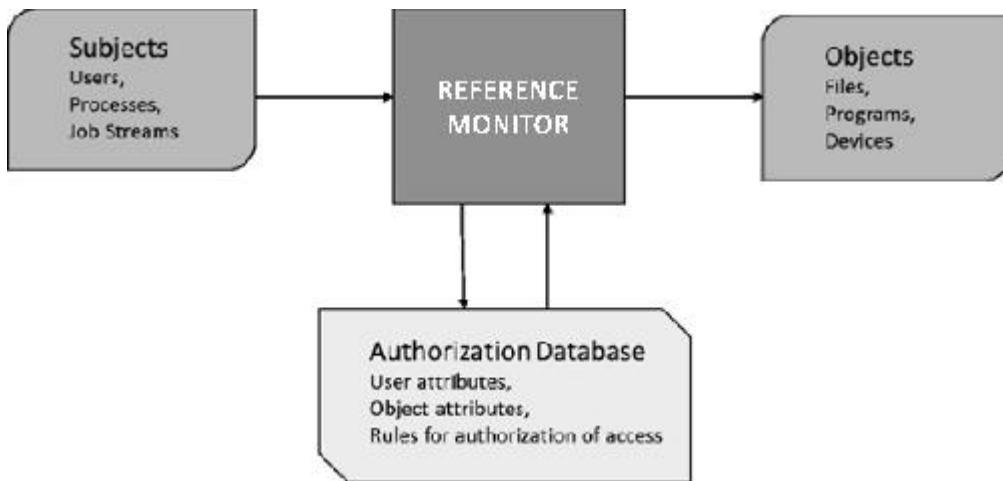


Figure 1.2 A reference monitor. A concept in which subjects make reference to objects using a set of authorization rules in terms of user and object attributes.

These principles provide architectural guidance pertaining to the design and development process of an access control system. As an

abstraction, the reference monitor does not dictate any specific policy to be enforced by the system, nor does it address any particular implementation. Rather, the reference monitor defines an assurance framework for highly secure IT systems and serves as the foundation in subsequent evaluations of multiuser computing systems. A security kernel, as depicted in [Figure 1.3](#), was introduced as a design concept for achieving the properties of the reference monitor.

Instead of implementing security controls in an operating system, the controls are implemented at a more primitive level that allows it to directly interact with the operating system on one side and with system hardware on the other. Through this implementation strategy, the portion of the system that can affect the functionality of access control is reduced from that of the entire operating system to that of just the kernel. A security kernel must implement a specific security policy as it can only be verified as being secure with respect to some specific security goals. A policy model provides the means for specifying what is necessary to be secure. It can be thought of as a specification of the authorization mechanism for establishing what shared resources are permitted to a given user or a program executed on behalf of a user. Adding to the complexity of designing and modeling a secure multiuser system is the notion of a Trojan horse, first introduced in the Anderson Report with attribution given to Dan Edwards of NSA and a member of the committee that wrote the report. In a Trojan horse attack, a process operating on behalf of a user may contain malware that surreptitiously performs other actions unbeknownst to the user. Of concern at the time was the threat of a Trojan horse reading classified information and writing it to a file labeled at a lower classification.

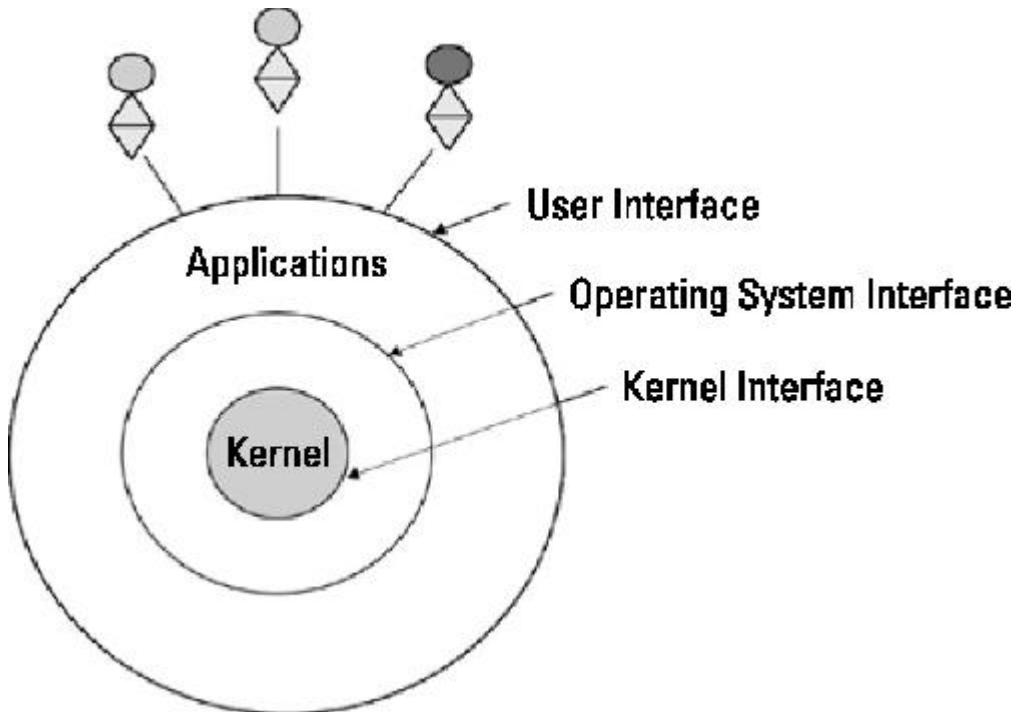


Figure 1.3 The security kernel is an implementation realizing the properties of the reference monitor.

1.2.3 Bell and LaPadula Security Model

With the Anderson Report providing a solid theoretical basis for the design and development of a robust authorization mechanism, focus turned to developing a formal model with well-defined security properties and proofs that the properties would ensure that information labeled as classified could not be read by users lacking appropriate clearances. Although the RAND Corporation report provided a basis for such a model with its no-read-up rule, Schell and others were dissatisfied, especially considering the potential of a Trojan horse attack. A more complete definition of security was sought and eventually produced through funding by ESD to the Mitre Corporation, where Bell and LaPadula developed the security model that has since been identified by their names [8]. It was important to have a means of preventing the writing of classified information to a lower classified file without an appropriate change in classification. Although a user could be trusted not to perform such actions, a user's processes were not assumed to possess similar trust.

The proposed solution could be summarized with two rules expressed in terms of subjects and objects. Subjects comprise both users and their processes that establish and take on a security level (at session creation) less than or equal to the user's clearance level. Objects are passive entities that represent protected resources with a security level commensurate with their classification level. The first rule, referred to as the simple security property (like the no-read-up rule), required that subjects be allowed to read information only if the security level of the subject was greater than or equal to the security level of the object. The second rule, referred to as the *-property (pronounced "star property"), required that subjects be allowed to possess simultaneous read access to one object and write access to another only if the security level of the second object was greater than or equal to the security level of the first. For example, if a user is logged in at the secret level, programs and processes acting on behalf of that user are permitted to read information at the secret and confidential level (in accordance with the simple security property) as well as write information at the secret and top secret level (in accordance with the *-property). Recognize that for a user to realize their full breadth of privileges requires logging on, then off, and back on at various levels.

In addition to the formulation of the simple security property and *-property, Bell and LaPadula also defined the basic security theorem, which facilitated subsequent proofs of security based on inductive reasoning. By assuming an initial secure state of a system, the system would be known to always remain in a secure state if changes to the state were to comply with the simple security property, *-property, and a discretionary security property (rules for manipulating the access matrix).

1.2.4 Trusted Computer Security Evaluation Criteria

With optimistic conclusions of DoD-sponsored research, availability of experimental and commercially viable multilevel secure implementations, and the establishment of the DoD Computer Security Center, the stage was set for the creation of a metric for inspiring the development and subsequent evaluation of secure operating systems.

Codification of such a standard took place with the DoD publication of the Trusted Computer Security Evaluation Criteria (TCSEC) in 1983,

also referred to as the Orange Book due to its orange cover [9]. The TCSEC provided four ascending divisions—*D*, *C*, *B*, and *A*, each with one or more classes for establishment and evaluation of trust. Division *D* included one class that was reserved for systems that had been evaluated and failed to meet all security requirements of any higher evaluation class. Division *C* included two escalating evaluation classes, *C1* and *C2*, which, among other requirements, provided for discretionary access control (DAC). First appearing in *C2* was a requirement for DAC to limit propagation of access rights (e.g., the ability to explicitly deny a user access to an object).

DAC, as the name implies, permits the granting and revocation of access permissions be left to the discretion of users. Inherent to DAC is the notion of ownership—a concept in which one user has total control over access to an object. In practice, a user who creates an object is normally given responsibility for controlling (granting and revoking) access to the object.

In addition to DAC requirements, Division *B* provided three escalating classes of requirements—*B1*, *B2*, and *B3*—to provide mandatory access control (MAC) protection. Distinguishing requirements included granularity of control, existence of a formal security policy model, and compliance to the reference monitor concept to include use of a security kernel. Division *A* included one class with requirements for formal design specification and verification (showing its kernel is a correct, non-bypassable, and requires minimal implementation of its MAC policy).

Implemented as either a row-wise or column-wise implementation of the access matrix, discretionary controls were prescribed to meet the need-to-know regulations for accessing classified information. Mandatory controls provided the multilevel security policy as formalized by the Bell-LaPadula Model. For systems *B1* or higher, objects were subject to protection under DAC and MAC through the combined use of two independent mechanisms.

By the mid-1990s, virtually every major computer vendor had one or more products evaluated by the DoD Computer Security Center with specialized systems meeting the requirements of *B2* or higher classes and the vast majority coming in at the *C2* level.

1.2.5 Discontent

Although the TCSEC was developed to address the security requirements of the DoD for accessing classified information, the prevailing argument was that the discretionary control of the C1 and C2 evaluation classes would promote trusted computer products that would meet the security requirements of the non-DoD market place. After all, the combined work of Lampson, Graham, Denning, Harrison, Ruzzo, and Ullman were focused on the general topic of protection without any market domain in mind. Although largely ignored, indications came as early as 1982 that the emerging TCSEC's approach to authorization might not be as generically applicable to the needs of the commercial sector as the DoD might have liked [10]. But in 1987, a highly publicized paper published in the prestigious IEEE Symposium of Security and Privacy by Clark and Wilson of MIT documented stark differences between commercial and military security requirements [11], arguing that the primary concern for most commercial applications is integrity rather than secrecy. Their approach mapped conventional accounting practice to information security and proposed two primary principles for ensuring information integrity: well-formed transactions and the separation of duty (SoD). Well-formed transactions constrain the ways in which users can modify data, thus ensuring that all data that starts in a valid state will remain so after the execution of a transaction. SoD stipulates that the completion of certain business processes requires the independent actions of two or more individuals (e.g., request and approve) to deter the likelihood of fraud. Unlike the Bell-LaPadula Security Model, which relied on access mediation in the operating system kernel, Clark and Wilson's approach relied on application-level controls. This difference in design addressed the goals of the two models. The military's multi-level security model seeks to control information flow, which can be defined in terms of low-level read-and-write operations. The commercial integrity model, as defined by Clark and Wilson, must ensure that information is modified only in authorized ways by authorized people, a requirement that is impossible to meet using only control over kernel-level operations.

Clark and Wilson were among the first to suggest that multilevel security is only one kind of MAC. Further corroboration for this perspective was published by Brewer and Nash [12], which introduced yet another MAC policy referred to as the Chinese wall policy. The stated objective of the Chinese wall policy is to prevent illicit flows of

information that can result in a conflict-of-interest. Consultants are naturally given access to proprietary information to provide a service for their clients, however, when a consultant gains access to the competitive practices of two banks, the consultant gains knowledge—amounting to insider information—that can undermine the competitive advantage of one or both institutions, or can be used for personal profit. The purpose of the Chinese wall policy is to identify and prevent the flow of information that can give rise to such conflicts. Included in the Chinese wall policy is a rule similar to the *-property, but with a different objective—preventing the leakage of data between controlled data sets—thereby providing the basis for its wall of protection.

1.2.6 Role-based Access Control

Like the Bell and LaPadula Model that formalized the multilevel security policy, role-based access control (RBAC) has its roots in historical practices that predate the formal model. Also like multilevel security, RBAC is, at least on the surface, conceptually simple. Access to computer system resources is based on user assignments to organization roles, each with predefined permission associations. Since this notion of roles, reflecting permissions for different jobs, had been used in specialized applications such as banking dating back to the 1970s, it is difficult to identify its exact origin. In one early research paper from the 1980s, Landwehr proposed a security model for a military messaging system application [13] with features that can be mapped to present day RBAC. A role was defined as the job a user performs, and a user may be assigned one or more roles to indirectly associate permissions with user. The model further specified that a user was always associated with at least one role in a session, and a user could change roles during the life of the session. To serve as a basis for authorization in ANSI SQL databases, Baldwin [14] alluded to the notion of roles but also addressed inheritance with named protection domains (NPDs) that could be related and organized into hierarchies based on NPD permission subsets.

Ferraiolo and Kuhn were the first to argue that RBAC would be broadly applicable. Although directives and regulations pertaining to the handling of classified information were available to serve as a rationale for TCSEC DAC and MAC, no analogous body of requirements had existed to drive the formulation of access control in the commercial

sector. This motivated Ferraiolo et al. to survey [15] the security needs of civilian government agencies and commercial enterprises. The responses indicated that these non-military institutions required a means by which to associate permissions with roles, and such functionality was not being accommodated with existing systems. In particular, end users did not necessarily own and control the information to which they were granted access. Rather, users were given access in a non-discretionary fashion based on their job functions or roles in the organization. In attempting to implement policy, organizations at the time often resorted to the cumbersome practice of central administration of ACL. A logical conclusion was that a form of RBAC would better meet the policy and policy-management needs of those surveyed rather than TCESC DAC.

To address these needs, Ferraiolo and Kuhn wrote a paper in 1992 titled *Role-Based Access Control*, which proposed a generalization and integration of existing application-specific approaches and concepts [16]. It included the sets, relations, and mappings used in defining roles and role hierarchies, user-role activation, and user-to-resource authorization as well as the option for constraints and more precise control. The model captured a method of delegating permissions to users based on roles that mirrored job functions, competencies, and authority as specified by the survey. It made a distinction between the roles that users are assigned to and the roles that users are using, as did Landwehr [13], it accommodated inheritance of privileges for users assigned to different roles, as did Baldwin [14], and it introduced two forms of separation of duties (SoD): static SoD, in support of objectives of Clark-Wilson [11]; and dynamic SoD, in support of the (by then) time-honored principle of least privilege.

Shortly after its publication, Sandhu addressed the Ferraiolo-Kuhn Model and RBAC's future significance:

...an important innovation, which makes RBAC a service to be used by applications... Instead of scattering security in application code, RBAC will consolidate security in a unified service which can be better managed while providing the flexibility and customization required by individual applications [17].

In 1996, Sandhu and colleagues introduced a framework for a family of RBAC models [18], famously referred to as RBAC96, that categorized RBAC into four separate but related models. The

framework, as shown in [Figure 1.4](#), included a base model with minimal features of an RBAC system, two advanced models that included the features of the base model with added support of role hierarchies and constraints for, among other purposes, accommodating SoD policies, and a fourth model that included combined features of the first three. In addition to providing a modular approach to RBAC, RBAC96 introduced the notion of administering RBAC relations through RBAC. Equally important, RBAC96's delineation of RBAC features provided a solid foundation for RBAC's future standardization.

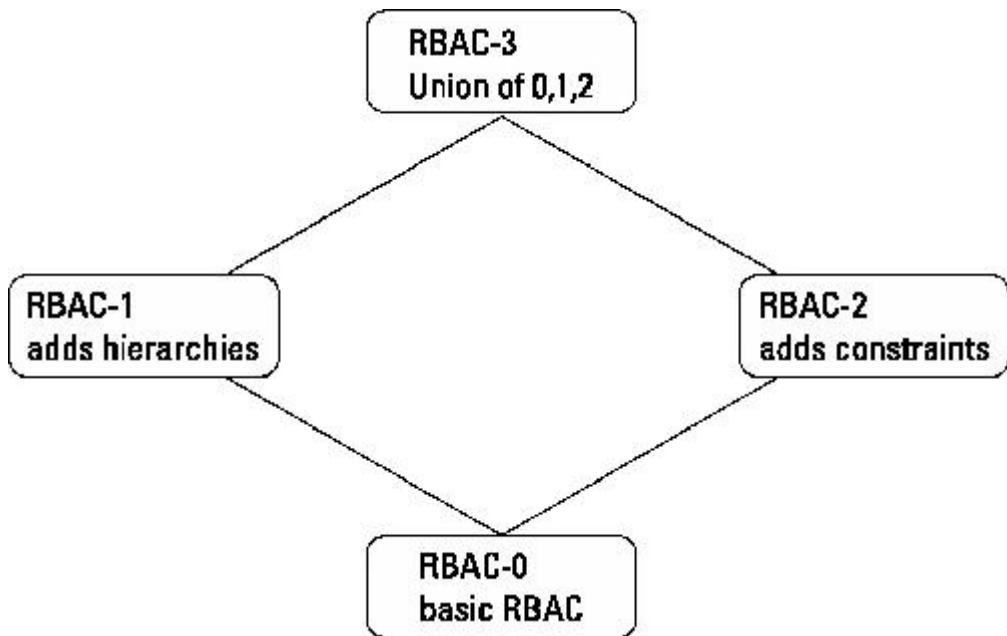


Figure 1.4 RBAC96 family of RBAC models.

The relevance of RBAC to the security needs of a wide variety of application domains, infrastructure technologies, and business processes quickly led to the establishment of a robust RBAC research community producing hundreds of papers and dozens of patent applications.

Part of RBAC's appeal is its dual use. RBAC has shown the capability to enforce a wide variety of enterprise-tailored access control policies through configuration of its relations, and it allows for efficient provisioning and deprovisioning of user permissions through creating and deleting user-to-role assignments.

In 2000, NIST initiated an effort to establish an international standard for RBAC by publishing a proposal [\[19\]](#) in the 5th ACM RBAC workshop. The proposed standard followed the RBAC96 structure and was updated [\[20\]](#) based on workshop discussions and rebuttals [\[21\]](#). In

2004, the standard was approved as INCITS 359-2004 by the ANSI International Committee for Information Technology Standards [22]. RBAC has since been included in the OASIS XACML Profile for RBAC, which has facilitated application of RBAC for access control in Web services [23].

Today, RBAC features appear at virtually all levels of computing including operating systems, database management systems, networks, and enterprise management levels. RBAC has also been incorporated into infrastructure technologies such as public key infrastructure, workflow management, and directory services.

1.2.7 Emergence of ABAC

Although ABAC concepts have historically paralleled that of RBAC, ABAC lacked a formal basis in support of its analysis and potential deployment throughout much of this period. That began to change with RBAC's noted limitations as well as the prevalence of internet and distributed systems where centralized management is difficult.

Critics cite three primary issues. First, RBAC can be cumbersome to set up and manage given the need for and difficulty of directly associating permissions with roles. Second, differentiating roles in different contexts (e.g., teller in branch A and teller in branch B) can result in large quantities of role definitions. In extreme cases, the number of defined roles can approach or even exceed the number of users. Third, role qualifiers of a user alone are often insufficient for expressing real-world access control policies.

Moreover, although RBAC and other access control standards existed, access control was often implemented in different ways. In modern day distributed computing environments, the management of access control information (also known as privilege management) can be a problematic. Enterprise administrators are forced to contend with a multitude of security domains when managing access policies and attributes. Even if properly coordinated across operating environments, global controls are hard to visualize and implement in a piecemeal fashion. Furthermore, because operating environments implement access control in different ways, it is difficult to exchange and share access control data across operating environments.

ABAC has gained the attention of businesses, academia, and standard bodies because of its potential benefits, particularly the advantage of simplified authorization management. Policies are expressed in terms of attributes without prior knowledge of potentially numerous users and resources that are or will be governed under those policies, and users and resources are independently assigned attribute values without knowledge of policy details. Furthermore, ABAC enables the instantiation and enforcement of combinations of multiple diverse policies using just one mechanism.

ABAC is often prescribed as an authorization framework, rather than a just a policy model. At its center, and conceptually like Anderson's reference monitor, is an ABAC engine (illustrated in [Figure 1.5](#)) that computes decisions based on assigned attributes of a requesting user, attributes of the object for which access is sought, optional environmental attributes (or conditions), and policies expressed in terms of those attributes.

The ABAC authorization framework includes a functional architecture for issuing access requests, computing, and enforcing access decisions in support of distributed and heterogeneous applications and services. Standardization will therefore inevitably play an important part in its success.

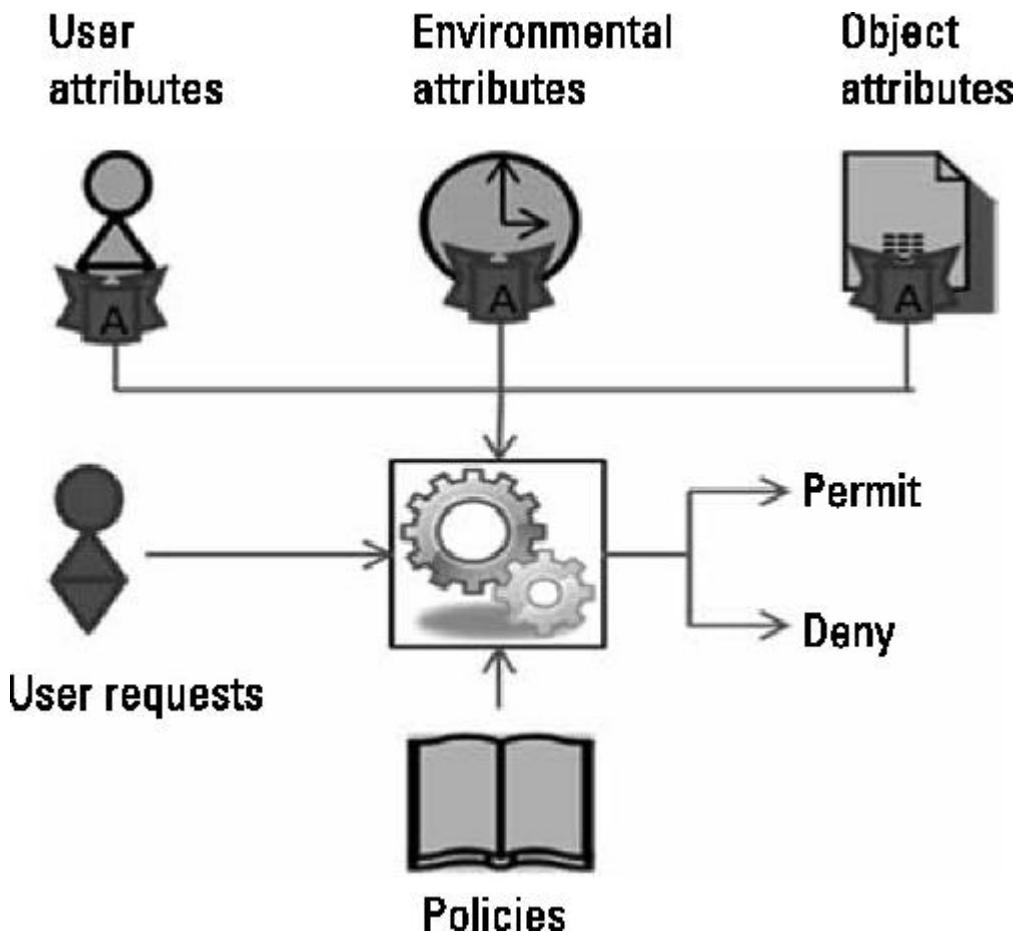


Figure 1.5 An ABAC engine computes decisions for user access requests based on attributes and policies expressed in terms of attributes.

Today there are two standards that comprehensively address the ABAC framework: Extensible Access Control Markup Language (XACML) [23] and Next Generation Access Control (NGAC) [24, 25]. These standards provide a single generic access control facility for applications, resulting in a dramatic alleviation of many administrative, interoperability, and usability challenges otherwise faced by enterprises. This is achieved by removing application reliance on the access control logic implemented in underlying, and often proprietary, operating environments. The necessary access control logic is implemented using a common set of access control modules that provide access decision functionality using a centralized policy and attribute repository. The XACML and NGAC standards are presented in detail respectively in [Chapters 4](#) and [5](#).

As of this writing, ABAC's market penetration and adoption continues to grow but still trails that of RBAC. However, that can easily

change given ABAC's administrative and interoperability advantages as well as its potential ability to accommodate the objectives of past models and ad hoc policies.

References

- [1] Denning, P. J., et al., *An Undergraduate Course on Operating Systems Principles*, Computer, January/February 1972, pp. 40-58.
- [2] Lampson, B. W., "Dynamic Protection Structures," *Proc. of the Fall Joint Computer Conference*, AFIPS '69, New York, NY; Association for Computing Machinery, November 1969, pp. 27-38.
- [3] Lampson, B. W., "Protection," *Proc. 5th Princeton Conference on Information Sciences and Systems*, pp. 437, 1971. Reprinted in Association for Computing Machinery Operating Systems Review Vol. 8, No. 1, January 1974, p. 18.
- [4] Graham, G. S., and P. J. Denning, "Protection: Principles and Practice," *Proc. Spring Joint Computer Conference*, AFIPS '72, Atlantic City, New Jersey; May 16-18, 1972, pp. 417-429.
- [5] Harrison, M. A., W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," Comm. Association for Computing Machinery, Vol. 19, No. 8, August 1976, pp. 461-471.
- [6] Ware, W. H., *Security Control for Computer Systems: Defense Science Board Task Force on Computer Security*, Technical Report R-609-1, Rand Corporation, Santa Monica, CA, February 1970.
- [7] Anderson, J. P., *Computer Security Technology Planning Study Volume II*, ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA, October 1972.
- [8] Bell, D. E., and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, Bedford, MA: The Mitre Corporation, 1973. See also D. E. Bell and L. J. LaPadula, *Secure Computer System: Unified Exposition and MULTICS Interpretation*, MTR-2997 Rev. 1, Bedford, MA: The MITRE Corporation, March 1976; also ESD-TR-75-306, rev. 1, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA.
- [9] Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, Supercedes CSC-STD-001-83 dated August 15, 1984.
- [10] Lipner, S. B., "Non-Discretionary Controls for Commercial Applications," *Proc. IEEE Symp. Secur Priv*, IEEE Computer Society Press, 1982, pp. 2-10.
- [11] Clark, D. D., and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies," *Proc. IEEE Symp. Secur Priv*, IEEE Computer

Society Press, 1987, pp. 184–194.

- [12] Brewer, D. F. C., and M. J. Nash, “The Chinese Wall Security Policy,” *Proc. IEEE Symp. Secur Priv*, April 1989, pp. 215–228.
- [13] Landwehr, C. E., C. L. Heitmeyer, and J. D. Mclean, “A Security Model for Military Message Systems,” *Association for Computing Machinery Transactions on Computer Systems*, Vol. 2 No. 3, pp. 198-222, August 1984.
- [14] Baldwin, R. W., “Naming and Grouping Privileges to Simplify Security Management in Large Database,” *Proc. IEEE Symp. Secur Priv*, April 1990, pp. 187-194.
- [15] Ferraiolo, D. F., D. M. Gilbert, and N. Lynch, *Assessing Federal and Commercial Information Security Needs*, Technical Report NISTIR 4976, National Institute of Standards and Technology, Computer Systems Laboratory, Gaithersburg, Maryland, November 1992.
- [16] Ferraiolo, D. F., and D. R. Kuhn, “Role-based Access Controls,” *Proc. 15th National Computer Security Conference*, National Institute of Standards and Technology, National Computer Security Center, October 1992, pp. 554–593.
- [17] Sandhu, R. S., et al., “Role-Based Access Control: A Multidimensional View,” *Proc. 10th Annual Computer Security Applications Conference*, December 1994, pp. 54-62.
- [18] Sandhu, R. S., et al., *Role-based Access Control Models*, Computer, Vol. 29, No. 2, February 1996, pp. 38-47.
- [19] Sandhu, R., D. Ferraiolo, and R. Kuhn, “The NIST Model for Role-Based Access Control: Towards a Unified Standard,” *Proc. 5th ACM Workshop on Role-Based Access Control*, Association for Computing Machinery, July 26-27, 2000, pp. 47-60.
- [20] Ferraiolo, D. F., et al., “Proposed NIST Standard for Role-based Access Control,” *Association for Computing Machinery Transactions on Information Systems Security*, Vol. 4, No. 3, 2001, pp. 224-274.
- [21] Jaeger, T., and J. E. Tidswell, “Rebuttal to the NIST RBAC Model Proposal,” *Proc. 5th Association for Computing Machinery Workshop on Role-Based Access Control*, July 26-27, 2000, pp. 65-66.
- [22] *Information technology – Role-Based Access Control (RBAC)*, INCITS 359-2004, American National Standard for Information Technology, American National Standards Institute, 2004.
- [23] eXtensible Access Control Markup Language (XACML) Version 3.0, OASIS Standard, January 22, 2013, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf>

- [24] American National Standards Institute, “Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA),” INCITS 499-2013, American National Standard for Information Technology, March 2013.
- [25] American National Standards Institute, “Information technology – Next Generation Access Control – Generic Operations and Data Structures,” INCITS 526, American National Standard for Information Technology, January 2016.

2

Access Control Models and Approaches

2.1 Introduction

Background and knowledge of prior access control policy models and approaches are necessary in understanding how ABAC fits into the field of access control and authorization management. In addition to these topics, this chapter introduces some basic terminology that is applied throughout the remainder of this book.

2.2 Terminology

For the purposes of access control, users can be human or non-human entities (e.g., programs or devices) that interact with the computer system. A user has an identity that is unique and maps to only one entity. User authorization and authentication are fundamental to access control. They are distinct concepts but are often confused. Part of the confusion stems from the close relationship between the two; proper authorization in fact is dependent on authentication. Authentication can be defined as verifying the identity of a user (for example, by using a password, or a biometric like a fingerprint), often as a prerequisite to allowing access to resources in a system, while authorization is the granting or denying of access rights to a user.

Classical access control models and mechanisms are defined in terms of subjects, access rights, and named objects. A user is unable to access objects directly, and instead must perform access through a subject. A subject represents a user and any system process or entity that may act on behalf of a user. Subjects are the active entities of a system that can act on an object, cause a flow of information between objects, or change the access state of the system

Objects are references to system entities that must be protected. Each object has a unique system-wide identifier. The set of objects may pertain to processes, files, ports, and other system abstractions, such as exhaustible system resources like printers. Subjects may also be included in the set of objects. In effect, this allows them to be governed by another subject. That is, the governing subject can administer the access of such subjects to objects under its control. The selection of entities included in the set of objects is determined by the protection requirements of the system.

Subjects operate autonomously and may interact with other subjects. Subjects may be permitted modes of access to objects that are different from those of other subjects. When a subject attempts to access an object, a reference mediation function determines whether the subject's assigned permissions adequately satisfy policy before allowing the access to take place. In addition to carrying out user accesses, a subject may maliciously (e.g., through a Trojan horse) or inadvertently (e.g., through a coding error) make requests that are unknown to and unwanted by its user.

Some access control models recognize operations as somewhat independent to access rights. Operations denote actions that can be performed on the contents of objects that represent resources or on access control data that embody policies and attributes. The entire set of operations (Op), is partitioned into two distinct, finite sets of operations: resource operations (ROp), and administrative operations (AOp). Common resource operations for an operating system include read, write, and execute for example. Resource operations can also be defined specifically for the system in which the model is implemented. Administrative operations on the other hand pertain only to the creation and deletion of policy and attribute elements pertinent to the model at hand, and, if included, are a stable part of the model, regardless of the implementation environment.

To be able to carry out an operation, the appropriate access rights are required. As with operations, the entire set of access rights (AR) are partitioned into two distinct, finite sets of access rights: resource access rights (RAR), and administrative access rights (AAR). Normally, a one-to-one mapping exists between ROp and RAR , but not necessarily between AOp and AAR . For instance, to assign an object to an object

attribute (an administrative operation) might require two access rights – “assign-from” a set of objects and “assign-to” a set of object attributes.

Central to access control is the notion of privileges. An individual privilege is a triple of the form (u, ar, o) [1], and has the meaning that user u has the access right ar on object o . With respect to a privilege (u, ar, o) , (ar, o) is a capability of u , but (ar, o) is also commonly referred to as a permission. For practical systems, privileges are not directly stored in their primitive form, and instead are indirectly represented in some alternate form (as access control data). Well-known representations include access control and capability lists, and roles and their relations as discussed below.

2.3 Access Control Models and Policies

Although data resources may be protected under a wide variety of different access policies, these policies can be generally categorized as either discretionary or mandatory controls. Discretionary access control (DAC) is a narrow administrative policy that permits system users to allow or disallow other users’ access to objects that are placed under their control [2]. Discretionary models form a broad class of access control models. Discretionary in this context means that subjects that represent users are allowed some freedom to manipulate the authorizations of other subjects to access objects [3]. Mandatory access control (MAC) models are the complement of discretionary models, insofar as they require that access control policy decisions are regulated by a central authority, not by the individual owner of an object. That is, authorizations to objects can be changed only through the actions of subjects representing administrators, and not by those representing users [3]. With MAC models, subjects and objects are typically classified into or labeled with distinct categories. Category-sensitive access rules that are established through administration completely govern the access of a subject to an object and are not modifiable at the discretion of the subject.

Models are usually written to bridge the rather wide gap in abstraction between policies and mechanisms. Many different access control models, both discretionary and mandatory, have been developed to suit a variety of purposes. Models are often developed or influenced by well-conceived organizational policies for controlling access to

information, whose key properties are generalized, abstracted, and described in some formal or semi-formal notation. Therefore, models typically differ from organizational policy in several ways. As mentioned, models deal with abstractions that involve a formal or semi-formal definition, from which the presence or lack of certain properties may be demonstrated. Organizational policy on the other hand is usually a more informally stated set of high-level guidelines that provide a rationale for the way accesses are to be controlled, and may also give decision rules about permitting or denying certain types of access. Policies may also be incomplete, include statements at variable levels of discourse, and contain self-contradictions, while models typically involve only essential conceptual artifacts, are composed at a uniform level of discourse, and provide a consistent set of logical rules for access control.

Organizational objectives and policy for access control may not align well with those of a particular access control model. For example, some models may enforce a policy that is too restrictive for some organizations to carry out their mission, but essential for others. Even if alignment between the policy and model is strong, in general, the organizational access control policy may not be satisfied fully by the model. For example, different federal agencies can have different conformance directives regarding privacy that must be met, which affect the access control policy. Nevertheless, access control models can provide a strong baseline from which organizational policy can be expressed and enforced.

Well-known models include discretionary access control, multilevel security, role-based access, Chinese Wall, originator control, Clark-Wilson, and n-person control. Some of these models are discussed below to give an idea of the scope and variability between models.

It is important to keep in mind that models are written at a high conceptual level, which stipulates concisely the scope of policy and the desired behavior between defined entities, but not the security mechanisms needed to reify the model for a specific computational environment, such as an operating system or database management system. While certain implementation aspects may be inferred from an access control model, such models are normally implementation-agnostic, insofar as they do not dictate how an implementation and its security mechanisms should be organized or constructed. These aspects

of security are addressed through information assurance processes (e.g., compliance of the mechanism to the model and policies).

2.4 Policy Enforcement

Authentication and authorization mechanisms are the building blocks that must be integrated into a coherent access control policy. The ability to enforce access control policies is a critical capability of most modern day enterprises. Policies are enterprise requirements that specify how access is managed and who, under what circumstances, may access what information. Among other issues, security policy enforcement is instrumental in preventing the unauthorized disclosure of sensitive data, protecting the integrity of vital data, mitigating the likelihood of fraud, and ultimately enabling the secure sharing of information.

Access control policies are enforced through a mechanism consisting of a fixed system of functions and a collection of access control data (reflecting the configuration of the mechanism) that together map a user's access request to a decision whether to grant or deny access. Included in the access control data is the set of permissions—each indicating a user's potential to perform an operation (e.g., read or write) on object or resource. Regarding a specific mechanism, permissions are not individually specified, but instead permissions are organized in terms of, and mapped (through administrative operations or a predefined set of rules) onto a set of subjects (processes) and object attributes, pertaining to a specific type or class of policy. Also common to access control mechanisms is a requirement to store and authenticate user identities. From an authenticated identity, an access control mechanism is able to establish a security context (activate a specific identity, groups, or other attributes) as a basis for granting or denying user and process access requests to objects managed under the control of the mechanism at hand. Operationally, access control mechanisms compute a series of decisions based on the specifics of the access control data, and ultimately enforce policy based on those decisions.

2.5 Discretionary Access Control

The access matrix discussed in the previous chapter was originally envisioned as a discretionary access control (DAC) model [1, 4]. Many other DAC models have been derived from the access matrix and share common characteristics. The access matrix was later formalized as the now well-known Harrison, Ruzzo, Ullman (HRU) model and used to analyze the complexity of computing the safety properties of the model, which was found to be undecidable [5, 6]. DAC policies can be expressed in the HRU model, but DAC should not be equated to it, since the HRU model is policy neutral and can also be used to express access control policies that are non-discretionary [7].

In addition to an administrator's ability to manipulate a subject's authorization to access objects, a DAC access matrix model leaves a certain amount of control to the discretion of the object's owner. Ownership of an object is typically conferred to the subject that created the object, along with the capabilities to read and write the object. For example, it is the owner of the file who can control other subjects' accesses to the file. Control then implies possession of administrative capabilities to create and modify access control entries associated with a set of other subjects, which pertain to the owned objects. Control may also involve the transfer of ownership to other subjects. Only those subjects specified by the owner may have some combination of permissions to the owner's files.

One approach to implementing DAC is through capability lists. In a capability system, access to an object is allowed if the subject that is requesting access possesses a capability for the object. A capability is a protected identifier that identifies both the object and an access right to be allowed to the subject that possesses the capability. Each subject is associated with a capability list, which stores its approved rights to all concerned objects. The right side of [Figure 2.1](#) provides an example of a capability list for the subject User A. A subject possessing a capability is proof of the subject having the access permissions. The principle advantage of capabilities is that it is easy to review all accesses that are authorized for a given subject. On the other hand, it is difficult to review the subjects that can access a particular object. To do so would entail an examination of each and every capability list. It is also difficult to revoke access to an object given the need for a similar examination. For this reason, capability lists have been criticized by some in their support of DAC policies and are therefore not commercially popular.

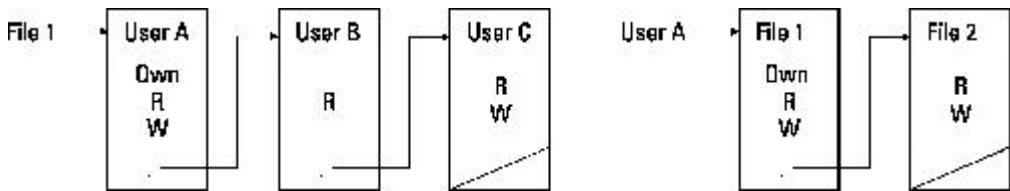


Figure 2.1 An access control list for object File 1 and a capability list for User A.

Perhaps the most common approach to representing and administering DAC policies is through the use of access control lists (ACLs). Each object is associated with an ACL (see left side of [Figure 2.1](#)) that stores the users and the users' approved access rights for the object. The list is checked by the access control system to determine if access is granted or denied.

The principle advantage of ACLs is that they make it easy to review the users that have access to an object as well as the rights that users have to the object. In addition, it is easy to revoke access to an object by simply deleting an ACL entry. These advantages make ACLs ideal for implementing policies that are object-oriented, such as the policy of DAC. Another advantage is that the lists need not be excessively long, if groups of users with common accesses to the object are assigned to a named group, and the group name is specified in the ACL for the object instead of the group's individual members.

DAC policy tends to be very flexible and is widely used in the commercial and government sectors. However, DAC potentially has two inherent weaknesses [3]. The first is the inability for an owner to control access to an object once permissions are passed on to another subject. For example, when one user grants another user read access to a file, nothing stops the recipient user from copying the contents of the file to an object under its exclusive control. The recipient user may now grant any other user access to the copy of the original file without the knowledge of the original file owner. Some DAC models have the ability to control the propagation of permissions. The second weakness is vulnerability to Trojan horse attacks, which is common weakness for all DAC models. In a Trojan horse attack, a process operating on behalf a user may contain malware that surreptitiously performs other actions unbeknownst to the user.

2.6 Mandatory Access Control Models

In contrast to DAC, mandatory access control (MAC) enables ordinary users' capabilities to execute resource operations on data, but not administrative operations that may influence those capabilities. MAC can be further characterized as controls that accommodate confinement properties to prevent indirect leakage of data to unauthorized users, and those that do not. Of the models discussed below, multilevel security and Chinese Wall address confinement, and RBAC does not.

2.6.1 Multilevel Security

The security objective of the multilevel security (MLS) as defined by the Bell-Lapadula Model [8] is to restrict the flow of information from an entity at one security level to an entity at a lesser security level. With regard to this policy, security levels are assigned to users, with subjects acting on behalf of users and also to objects. Security levels have a hierarchical and a nonhierarchical component, where the nonhierarchical component is represented by a set of categories. For instance, the hierarchical components might include unclassified (U), confidential (C), secret (S), and top-secret (TS) while the categories may include {}, {NATO}, {nuclear}, {NATO, nuclear}. The security levels are partially ordered under a dominance relation, often written as \geq , that takes into consideration both components. A security level $s/1$ dominates $\geq s/2$, if and only if the hierarchical component of $s/1$ is greater than or equal to that of $s/2$ and the category set of $s/2$ is a subset of that of $s/1$. The partial ordering under the dominance relation for hierarchical component S and TS, and categories nuclear and NATO, is shown in [Figure 2.2](#) where, for example, $(TS, \{\}) \geq (S, \{\})$, $(S, \{nuclear, NATO\}) \geq (S, \{nuclear\}) \geq (S, \{\})$.

The security level of the user, often referred to as the user's clearance level, reflects the level of trust bestowed to the user and must always dominate the security levels that are assigned to the user's subjects. The security levels that are assigned to an object, often referred to as the object's classification level, reflect the sensitivity of the contents of the objects. The security objective of MLS is to restrict the flow of information from an entity at one security level to an entity at a lesser security level. Two properties accomplish this:

- *Simple security property*: A subject is permitted read access to an object if the subject's security level dominates the security level of the object.
- **-property*: A subject is permitted write access to an object if the object's security level dominates the security level of the subject.

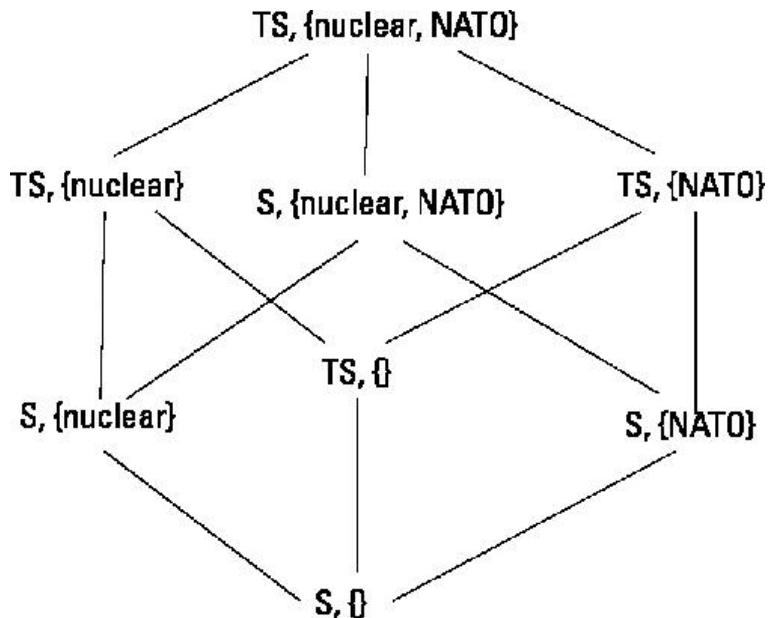


Figure 2.2 A partial ordering of security levels under the dominance relation.

Satisfaction of these properties prevents users from being able to read information that dominates their clearance level. The simple security property directly supports this policy, never allowing a subject to read information that dominates the invoking user's clearance level. Indirectly, the **-property*, also referred to as the confinement property, prevents the transfer of data from an object of a higher security level to an object of a lower security level, and is required to maintain system security in an automated environment. [Figure 2.3](#) illustrates a system of subjects and objects where subjects are permitted read and write access to objects in accordance with the simple security and **-property*, and as a consequence information is only permitted to flow in the opposite direction of object dominance.

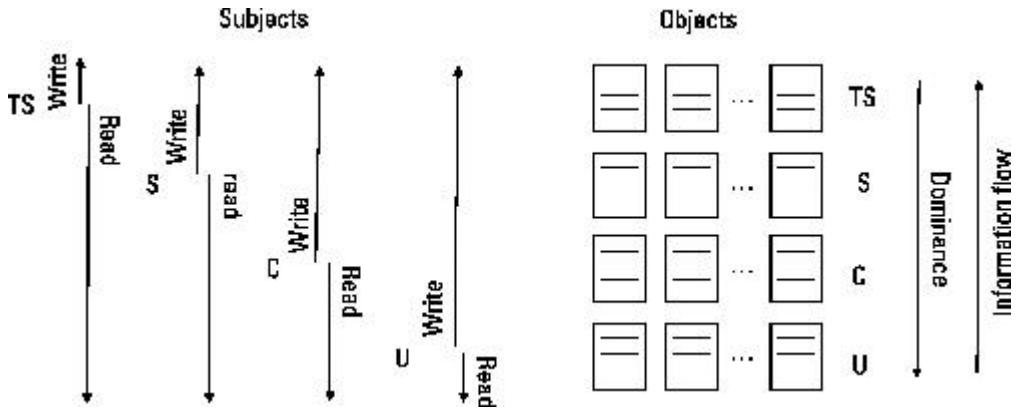


Figure 2.3 A system of subjects and objects where subjects are only permitted to perform read and write operations in accordance with the simple security and *-property.

These two properties are supplemented by the tranquility property, which can take one of either two forms: strong, and weak. Under the strong tranquility property, the security level of a subject or object does not change while the object is being referenced. The strong tranquility property serves two purposes. First, it associates a subject with a security level. Second, it prevents a subject from reading data with a high security level, storing the data in memory, switching its level to a low security level, and writing the contents of its memory to an object at lower level.

Under the weak tranquility property labels are allowed to change, but never in a way that can violate the defined security policy. It allows a session to begin in the lowest security level, regardless of the user's security level, and increases that level only if objects at higher security levels are accessed. Once increased, the user's security level for this session can never be reduced, and all objects created or modified take on the security level held by the session at the time when the object was created or modified, regardless of its initial security level. This is known as the high water mark principle.

Because of the constraints placed on the flow of information, MLS models prevent software infected with a Trojan horse from violating policy. Information can flow within the same security level or higher, preventing leakage to a lower level. However, information can pass through a covert channel in MLS, where information at a higher security level is deduced by inference, such as assembling and intelligently combining information of a lower security level.

2.6.2 Chinese Wall Policy and Model

The Chinese Wall policy evolved to address conflict-of-interest issues related to consulting activities within banking and other financial disciplines [9]. The stated objective of the Chinese Wall policy and its associated model is to prevent illicit flows of information that can result in conflicts of interest. The Chinese Wall model is based on several key entities: subjects, objects, and security labels. A security label designates the conflict-of-interest (COI) class and the company dataset (CD) of each object.

The Chinese Wall policy is application-specific in that it applies to a narrow set of activities that are tied to specific business transactions. Consultants or advisors are naturally given access to proprietary information to provide a service for their clients. However, when a consultant gains access to the competitive practices of two banks, the consultant essentially obtains insider information that could be used to profit personally or to undermine the competitive advantage of one or both of the institutions.

The Chinese Wall model establishes a set of access rules that comprises a firewall or barrier, which prevents a subject from accessing objects on the wrong side of the barrier. It relies on the consultant's dataset to be logically organized such that each CD belongs to exactly one COI class, and each object belongs to exactly one CD (shown in [Figure 2.4](#)). A subject can have access to at most one CD in each COI class. However, the choice of dataset is at the subject's discretion. Once a subject accesses (i.e., reads or writes) an object in a CD, the only other objects accessible by that subject lie within the same dataset or within the datasets of a different COI class. In addition, a subject can write to a dataset only if it does not have read access to an object that contains unsanitized information (i.e., information not treated to prevent discovery of a corporation's identity) and is in a CD different from the one for which write access is requested.

The Chinese wall model is summarized using two rules defined by Brewer and Nash, one for reading, and one for writing:

- Read Rule—Subject S can read object O only if:
 - O is in the same CD as some object previously read by S , or
 - O belongs to a COI class for which S has yet to read an object.
- Write Rule—Subject S can write object O only if
 - S can read O under the read rule, and

- No object can be read within a different CD than the one for which write access is requested.

It is important to recognize that the Brewer-Nash rules consider subjects to include both users and the processes that are acting on behalf of the users, and the rule for writing takes into consideration the possibility of a Trojan horse that can leak sensitive data outside a given company dataset.

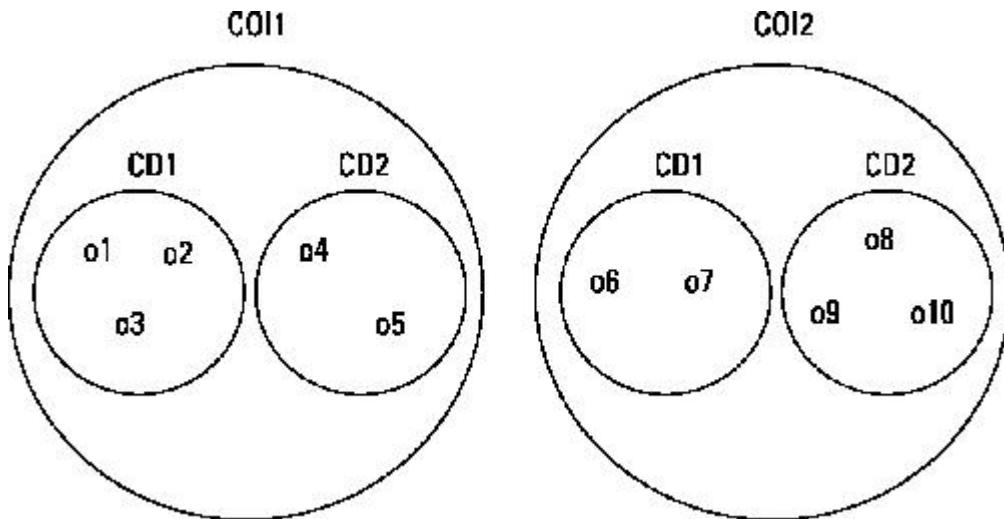


Figure 2.4 Organization of objects in company datasets (CDs), and CDs in conflict of interest (COI) classes.

2.6.3 Role-Based Access Control

The standard role-based access control (RBAC) model [10] governs the access of a user to information through roles for which the user is authorized to perform. The RBAC model (as illustrated in [Figure 2.5](#)) is based on several entities: users, roles, permissions, sessions, operations, and objects. A user represents an individual or an autonomous entity of the system. A role represents a job function or job title that carries with it some connotation of the authority held by a member of the role. Access authorizations on objects are specified for roles, instead of users. A role is fundamentally a collection of permissions to use objects appropriate to conduct a particular job function, while a permission represents an operation or mode of access to one or more objects of a system. Objects represent the protected resources of a system.

Users are given authorization to operate in one or more roles, but must utilize a session to gain access to a role. A user may invoke one or

more sessions, and each session relates a user to one or more roles. The concept of a session within the RBAC model is equivalent to the more traditional notion of a subject discussed earlier. When a user operates within a role, it acquires the permissions assigned to the role. Other roles authorized for the user, which have not been activated in a session, remain dormant and the user does not acquire their associated permissions. Through this role activation function, the RBAC model supports the principle of least privilege, which requires that a user be given no more permission than necessary to perform a job.

Another important feature of RBAC is role hierarchies, whereby one role at a higher level can acquire the permissions of another role at a lower level, through an explicit inheritance relation. A user assigned to a role at the top of a hierarchy is also indirectly associated with the permissions of roles lower in the hierarchy and acquires those permissions as well as those assigned directly to the role. Standard RBAC also provides features to express policy constraints involving separation of duty (SoD) and cardinality. SoD is a security principle used to formulate multiperson control policies in which two or more roles are assigned responsibility for the completion of a sensitive transaction, but a single user is allowed to serve only in some distinct subset of those roles (e.g., not allowed to serve in more than one of two transaction-sensitive roles). Cardinality constraints that limit a role's capacity to a fixed number of users have been incorporated into SoD relations in standard RBAC.

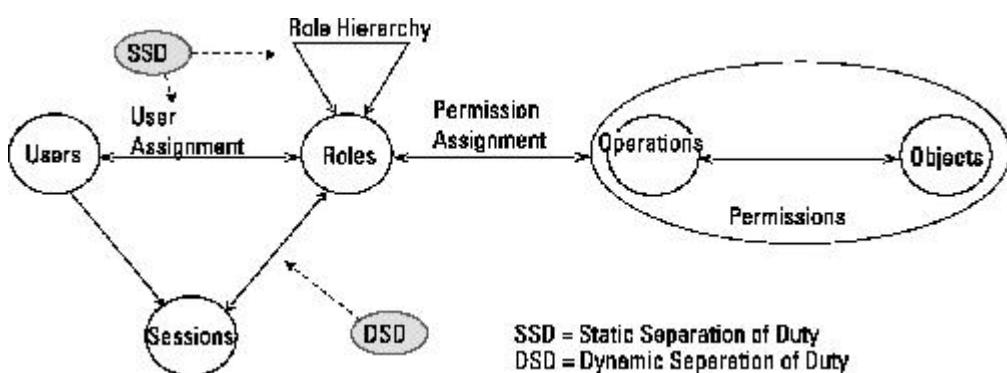


Figure 2.5 The RBAC model of entity datasets and relations.

Two types of SoD relations exist: static separation of duty (SSD) and dynamic separation of duty (DSD). SSD relations place constraints on the assignments of users to roles, whereby membership in one role may prevent the user from being a member of another role, and thereby

presumably forcing the involvement of two or more users in performing a sensitive transaction that would involve the permissions of both roles. DSD relations, like SSD relations, limit the permissions that are available to a user, while adding operational flexibility, by placing constraints on roles that can be activated within a user's sessions. As such, a user may be a member of two roles in DSD, but unable to execute the permissions that span both roles within a single session.

Certain access control models may be simulated or represented by another. For example, MLS can simulate RBAC if the role hierarchy graph is restricted to a tree structure rather than a partially ordered set [11]. RBAC is also policy neutral, and sufficiently flexible and powerful enough to simulate both DAC and MLS [12], although not as efficiently as their native models.

References

- [1] Lampson, B. W., "Protection," in *Proceedings 5th Princeton Conference on Information Sciences and Systems*, pp. 437, 1971. Reprinted in *Association for Computing Machinery Operating Systems Review*, Vol. 8, No. 1, January 1974, p. 18.
- [2] *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, Version-1, National Computer Security Center, Fort George G. Meade, Maryland, USA, September 30, 1987, p. 29.
- [3] Hu, V. C., D. F. Ferraiolo, and D. R. Kuhn, "Assessment of Access Control Systems," *National Institute of Standards and Technology (NIST) Interagency Report (IR) 7316*, September 2006, p. 60.
- [4] Graham, G. S., and P. J. Denning, "Protection: Principles and Practice," *Spring Joint Computer Conference*, AFIPS '72, Atlantic City, New Jersey, May 16-18, 1972, pp. 417-429.
- [5] Harrison, M. A., W. L. Ruzzo, and J. D. Ullman, "Protection in Operating Systems," *Comm. Association for Computing Machinery*, Vol. 19, No. 8, August 1976, pp. 461-471.
- [6] Tripunitara, M. V., and N. Li, *The Foundational Work of Harrison-Ruzzo-Ullman Revisited*, CERIAS Tech Report 2006-33, Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette, Indiana, 2006, p. 17.
- [7] Li, N., and M. V. Tripunitara, "On Safety in Discretionary Access Control," *Proc. IEEE Symp. Secur Priv*, Oakland, California, May 8-11, 2005, pp. 96-109.

- [8] Bell, D. E., and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations and Model*, Bedford, MA: The Mitre Corporation, 1973. See also D. E. Bell and L. J. LaPadula, *Secure Computer System: Unified Exposition and MULTICS Interpretation*, MTR-2997 Rev. 1, Bedford, MA: The MITRE Corporation, March 1976; also ESD-TR-75-306, rev. 1, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA.
- [9] Brewer, D. F. C., and M. J. Nash, “The Chinese Wall Security Policy,” *Proc. IEEE Symp. Secur Priv*, April 1989, pp. 215–228.
- [10] *Information technology – Role-Based Access Control (RBAC)*, INCITS 359-2004, American National Standard for Information Technology, American National Standards Institute, 2004.
- [11] Kuhn, D. R., “Role Based Access Control on MLS Systems Without Kernel Changes,” *3rd Association for Computing Machinery Workshop on Role Based Access Control (RBAC '98)*, Fairfax, Virginia, October 22-23, 1998, pp. 25-32.
- [12] Osborn, S., R. Sandhu, and Q. Munawer, “Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies,” *Association for Computing Machinery Transactions on Information and System Security (TISSEC)*, Vol. 3, No. 2 (May 2000), pp. 85-106.

3

ABAC Models and Approaches

3.1 Introduction

Attribute-based access control (ABAC) has gained significant attention from businesses, academia, and standards bodies in recent years. ABAC uses attributes on users, objects, and optional entities such as environmental conditions, and specifies access policies in terms of those attributes to assert who can perform which operations on which objects. As a result, it is easy to set up and modify ABAC policies as needs change. ABAC enables increased precision in specifying policy over prior models by allowing for more inputs into an access control decision, providing a bigger set of possible combinations of those variables to reflect a larger and more definitive set of possible rules to express policies. The access control policies that can be implemented in ABAC are limited only by the computational language and the richness of the available attributes.

This flexibility enables the greater breadth of subject access to a wider range of objects without specifying individual relationships between each subject and each object. For example, a subject is assigned a set of subject attributes upon employment (e.g., Nancy Smith is a *nurse practitioner* in the *cardiology department*). An object is assigned its object attributes upon creation (e.g., a folder with *medical records* of *heart patients*). Objects may receive their attributes either directly from the creator or as a result of some automated scheme. The administrator or owner of an object creates an access control rule using attributes of subjects and objects to govern the set of allowable capabilities (e.g., all *nurse practitioners* in the *cardiology department* can view the *medical records* of *heart patients*). Under ABAC, access decisions can change between requests by simply changing attribute values, without the need

to change the subject/ object relationships defining underlying rule sets. This provides a more dynamic access control management capability and limits long-term maintenance requirements of object protections.

For traditional information systems, a resource-system provider will only deal with a set of known subjects, and only those subjects are permitted to access a set of known system resources. This works well in a centralized system where subjects and resources are known and relatively static, but it can be difficult to apply in distributed systems. ABAC allows for a more ad hoc and dynamic approach to accessing system resources. Under ABAC the relationship between resource consumers and resource providers is often not established up-front, but rather consumers can discover and make use of dynamically appearing and disappearing resources based on the authorization state of the ABAC system.

Further, ABAC enables object owners or administrators to apply access control policy without prior knowledge of the specific subject and for an unlimited number of subjects that might require access. As new subjects join the organization, rules and objects do not need to be modified. As long as the subject is assigned the attributes necessary for access to the required objects (e.g., all *nurse practitioners* in the *cardiology department* are assigned that as the value for their departmental affiliation or department attribute), no modifications to existing rules or object attributes are required. This benefit is often referred to as accommodating the external (unanticipated) user and is one of the primary benefits of employing ABAC.

While there is currently no single agreed upon definition of ABAC, there are commonly accepted high-level definitions and descriptions of its function. One such high-level description is given in National Institute of Standards and Technology (NIST)'s publication, a “Guide to Attribute Based Access Control (ABAC) Definition and Considerations” [1]:

Attribute Based Access Control: An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, (optionally) environmental conditions, and a set of policies that are specified in terms of those attributes and conditions.

ABAC models define authorized accesses based on characteristics of a wide variety of system entities, known as *attributes*. The entire set of attributes is referenced by a unique name. Common to all models are two types of attributes:

- *Subject Attributes*: Each subject is assigned a set of attributes that can represent subject identities, ages, roles, affiliations, or other common characteristics pertinent to policy, such as security clearances.
- *Object Attributes*: Each object is assigned a set of attributes. Object attributes characterize data and other resources by identifying collections of objects, such as those associated with certain projects, applications, or security classifications.

Subject and object attributes can be assigned to their entities either through administrative actions, or though properties or metadata maintained by the system. In some models, attributes also have an associated type such as an integer or string that can be used directly for comparison (e.g., age of the subject ≥ 21).

In addition to subject and object attributes are environmental attributes (also known as environmental conditions) that are common to several, but not all models.

- *Environmental attributes*: Environmental attributes that depend on the availability of system sensors that can detect and report values are somewhat different from subject and resource attributes. Environmental attributes are not properties of the subject or resources, but are measurable characteristics that pertain to the operational or situational context in which access requests occur. These environmental characteristics are subject and object independent, and may include the current time, day of the week, or threat level.

Several models, standards, research prototypes, and products exist today that embody ABAC concepts. Collectively these concepts define ABAC as an access control framework that comprises (1) access control data for the expression of attributes and policies, (2) a set of administrative operations (often in the form of a policy language) for configuring access control data, and (3) a set of functions for enforcing

policy on requests to execute operations on objects and for computing access decisions to accommodate or reject those requests based on the current state of the access control data. This ABAC framework encompasses four layers of functional decomposition: enforcement, decision, administration, and access control data, and involves several components that work together.

The remainder of this chapter touches on many of these ABAC concepts—delineating between ABAC policy models and ABAC architectures. Also included is an overview of ABAC’s most salient functional components and architecture. The chapter introduces, characterizes, and provides representative examples of various ABAC policy models, then describes approaches to combining RBAC and ABAC features. The material presented in this chapter provides a solid background to aid in understanding and appreciating the XACML and NGAC standards presented in [Chapters 4](#) and [5](#).

3.2 ABAC Architectures and Functional Components

Although there are several proposed ABAC policy models, ABAC implementations generally follow two architectural approaches. The first approach is based on XACML’s reference architecture and the second is based on NGAC’s functional architecture [2]. Both architectures encompass four layers of functional and information decomposition—enforcement, decision, access control data, and administration—Involving several components that work together to bring about policy-preserving access control.

XACML-style architecture (illustrated in [Figure 3.1](#)) involves a multistep authorization process [3]. At its core, the XACML architecture includes a policy decision point (PDP) that computes decisions to permit or deny subject requests (to perform actions on resources). Requests are issued from, and PDP decisions are returned to, a policy enforcement point (PEP) using a standardized request and response language. The PEP is typically implemented as a component of an operating environment that is tightly coupled with its application. The PEP may not generate requests in XACML syntax nor process XACML syntax-compliant responses. In order to convert access requests in native format (of the operating environment) to XACML access requests (or convert a PDP response in XACML to a native format), the XACML architecture

may optionally include a context handler. The context handler is a component that manages the XACML authorization workflow. It is not explicitly shown in Figure 3.1 since it is an integral part of the PEP or PDP.

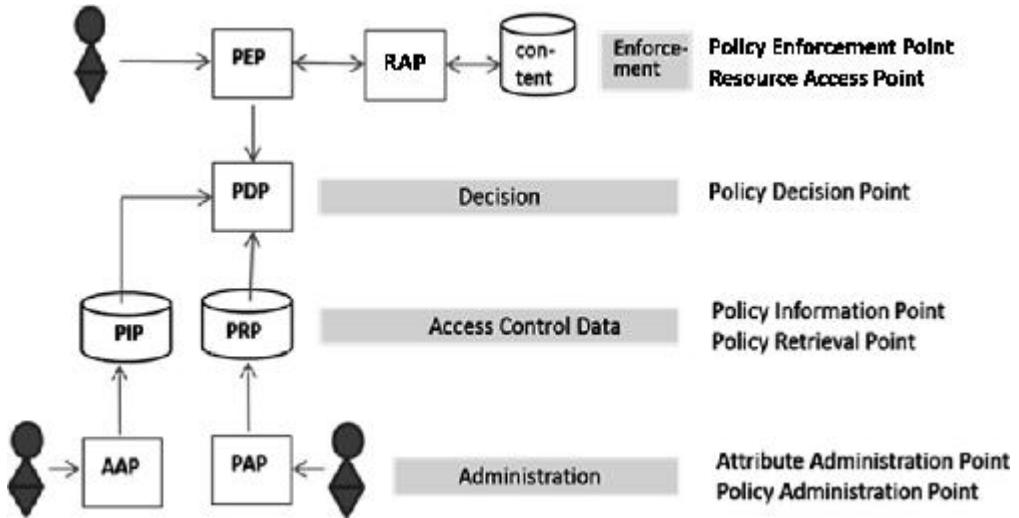


Figure 3.1 XACML style ABAC architecture.

A request is comprised of attributes extracted from the policy information point (PIP) minimally sufficient for target matching. The PIP is shown as one logical store, but in fact may comprise multiple physical stores. In computing a decision, the PDP queries policies stored in a policy retrieval point (PRP). If the attributes of the request are not sufficient for rule and policy evaluation, the PDP may request the context handler to search the PIP for additional attributes. Information and data stored in the PIP and PRP comprise the access control data and collectively define the current authorization state.

A policy administration point (PAP) using the XACML policy language creates the access control data stored in the PRP in terms of rules for specifying policies, policy sets as containers of policies, and rule and policy-combining algorithms. Although not included in the XACML reference architecture, an attribute administration point (AAP) is shown for creating and managing the access control data stored in the PIP. AAP implements administrative routines necessary for the creation and management of attribute names and values for users and resources. The resource access point (RAP) implements routines for performing operations on a resource that is appropriate for the resource type. In the event that the PDP returns a permit decision, the PEP issues a command to the RAP for execution of an operation on resource content. The RAP,

in addition to the PEP, runs in an application's operating environment, independent of the PDP and its supporting components. The PDP and its supporting components are typically implemented as modules of a centralized authorization server that provides authorization services for multiple types of operations.

NGAC's functional architecture is shown in [Figure 3.2](#). Among its components is a PEP that traps application requests. An access request includes a process ID, user ID, operation, and a sequence of one or more operands mandated by the operation that pertain to either a data resource or an access control data element or relation. Administrative operational routines are implemented in the PAP and resource operational routines (e.g., read and write) are implemented in the RAP.

To determine whether to grant or deny, the PEP submits the request to a PDP. The PDP computes a decision based on the current configuration of data elements and relations stored in the PIP, via the PAP. Unlike the XACML architecture, the access request information from an NGAC PEP together with the NGAC relations (retrieved by the PDP) provide the full context for arriving at a decision. The PDP returns a decision of grant or deny to the PEP. If access is granted and the operation was read/write, the PDP also returns the physical location where the object's content resides, the PEP issues a command to the appropriate RAP to execute the operation on the content, and the RAP returns the status. In the case of a read operation, the RAP also returns the data type of the content (e.g., PowerPoint) and the PEP invokes the correct data service application for its consumption. If the request pertained to an administrative operation and the decision was *grant*, the PDP issues a command to the PAP for execution of the operation on the data element or relation stored in the PIP, and the PAP returns the status to the PDP, which in turn relays the status to the PEP. If the returned status by either the RAP or PAP is successful, the PEP submits the context of the access to the event processing point (EPP). If the context matches an event pattern, the EPP automatically executes an associated set of administrative operations, potentially dynamically changing the access state.

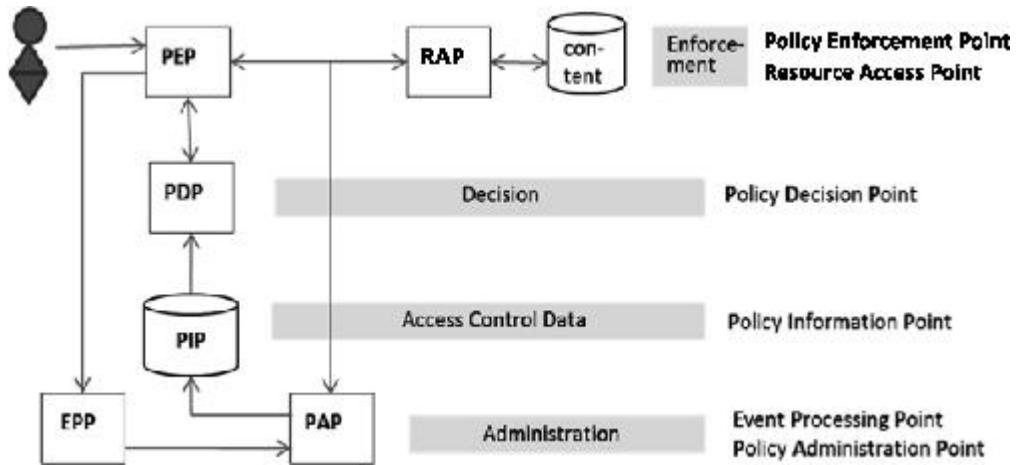


Figure 3.2 NGAC style ABAC architecture.

3.3 Logical-Formula and Enumerated ABAC Policy Models

A number of models have been proposed in the literature for the expression of ABAC policies. Common to these models are user and object attributes, and a computational language for expressing policies in terms of those attributes. In general, there are two techniques for specifying ABAC policies [4]. The most common approach is to define authorization policies by using logical-formula involving predicates expressed using logical operators (e.g., AND, OR, \geq , \neq) on attribute values of varying types (e.g., strings and integers). For example, $\text{can_access}(s, a, o) \rightarrow \text{role}(s) = \text{"doctor"} \text{ AND } \text{ward}(s) = \text{ward}(o) \text{ AND } (a = \text{read} \text{ OR } a = \text{write})$ specifies that any subject with a role of doctor can read or write any object where the ward of the subject is the same as the ward of the object. For purposes of evaluation, both subjects and objects need to be individually assigned to their attribute values in advance. XACML includes a policy specification language that falls into this category-based. References [5–7] are examples of other logical-formula ABAC policy models. Logical-formula authorization policies are powerful and conveniently specify even complicated business requirements.

The other technique for expressing policy is by enumeration or configuration of relations involving attribute names. NGAC [2] and label-based access control [4] fall under this category. For instance, for accessing system resources (as opposed to access control information) NGAC specifies policies in part through configuration of containment relations, association relations often denoted by (uai, opsi, and oai),

where ua_i and oa_i are user and object attributes and ops_i is an operation set and policy classes ($pc_1, pc_2 \dots pc_n$). In particular, $\text{can_access}(s, a, o) \rightarrow$ for all pc_i that contain o , there exists an association (ua_i, ops_i, oa_i) where o is contained in oa_i , oa_i is contained in pc_i , s is contained in ua_i , and a is contained in ops_i .

These techniques have relative advantages and disadvantages with respect to efficiency of policy expression and policy review. Both approaches have the potential for substantial expressive power and can express many of the same policies. For example, the graph in [Figure 3.3](#), where the arrows represent containment and the dotted lines represent associations, illustrates a policy similar to the above logical-formula policy example.

Note that the enumerated form of the policy expression uses three association relations, while the logical-formula approach uses just one rule. In general, as the number of data instances within the scope of policy increases, so will the need for corresponding association relations, but the number of rules will always remain at one. Although the number of association relations may exceed the number of logical rules in expression of the same policy, both enumerated and logical-formula approaches require the same number of attribute value assignments.

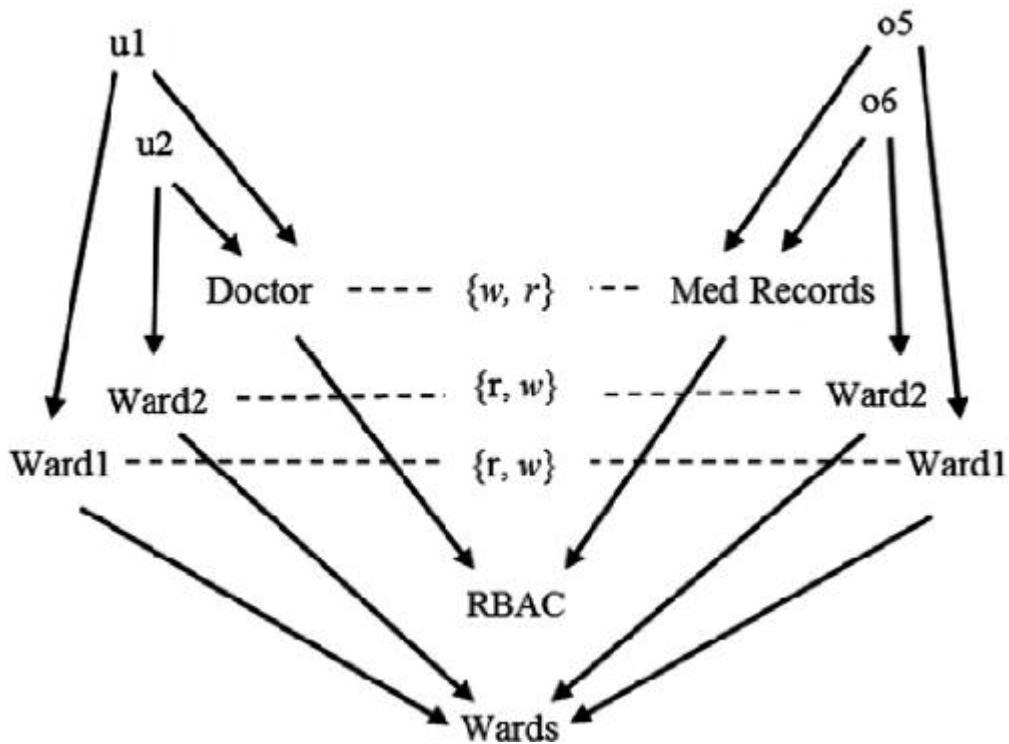


Figure 3.3 Enumeration graph.

A desired feature of access control is the review of capabilities (operation, object) of subjects and review of access control entries (subject, operation) of objects. As discussed in [Chapter 2](#), these policy reviews are considered key advantages of capability lists and access control list mechanisms. Being able to review capabilities of a subject is also referred to as a before the fact audit and enables resource discovery. Before the fact audit has been noted as a key motivating feature behind the deployment of RBAC and includes being able to review the access consequences of assigning a user to a role. Resource discovery includes the capability for a user or administrator to discover or see accessible objects. Being able to review the access control entries of an object is equally important. Who are the subjects that can access this object and what are the consequences of assigning an object to an attribute or deleting an assignment?

Enumerated policy models support efficient algorithms for both per-user and per-object review. Biswas, Sandhu, and Krishnan report in [\[4\]](#) that conducting a policy review using enumerated policies is inherently simple. Logical-formula-based mechanisms cannot do review efficiently. Conducting a policy review under such mechanisms is equivalent to the satisfiability problem in propositional logic, which is NP complete. In other words, there exists no method of determining the authorization state without testing all possible decision outcomes.

3.4 ABAC Model—Applications Primatives

The emergence of Web services technologies and the evolution of distributed systems toward service oriented architectures (SOA) have helped promote collaboration and information sharing by breaking down stove-piped systems and connecting them through interoperable system-to-system interfaces.

By far the largest area of research in domain specific ABAC models is towards attribute and policy-based access control for Web services. ABAC provides a potential solution to furthering automated Web service discovery and use by allowing access control decisions to be made without prior knowledge of the subject or their relation to the service provider. Of the many ABAC models targeting Web services, most notable is the logical-formula-based ABAC policy model by Yuan and

Tong (ABAC for Web Services) [7], upon which several other ABAC models are based (e.g., [8, 9]).

In the Yuan and Tong policy model, attributes are specified as name-value pairs. An attribute name/ID denotes the property or characteristic associated with a subject, resource, or environment. For example, in a medical setting, the attribute name role associated with a subject may have doctor, intern, and admissions nurse values. Subject and resource instances are specified using a set of name-value pairs for their respective attributes. For example, the subject attributes used in a medical policy may include: role = “doctor”, role = “consultant”, ward = “pediatrics”, subjectname = “smith”; environmental attributes: time = “12:11”, threatlevel = “low”; and resource attributes: resource-id = “medical-records”, wardlocation = “pediatrics”, patient = “johnson”.

A functional notation is used for reporting on attribute values with the format ATTR(), where the parameter may be a subject, resource, action, or environment. A functional notation for reporting on the value assignment of individual attributes is also used. For example: role(s1) = “doctor”, wardlocation(r1) = “pediatrics”, and currenttime(e) = “12:11”. A policy rule decides on whether a subject s can access a resource r in a particular environment e , is provided as a Boolean function of the s , r , and e attributes:

Rule: can_access ($s, r \leftarrow f(\text{ATTR}(s), \text{ATTR}(r), \text{ATTR}(e))$)

If the function’s evaluation is true, then the access to the resource is granted; otherwise the access is denied. A policy may consist of one or more policy rules and the access control decision process is an evaluation of applicable policy rules. For example, a rule specifying that subjects with role of doctor can access resources on the same ward as that of the doctor can be written as follows:

Rule1: can_access ($s, r, e \leftarrow \text{role}(s) = \text{“doctor”} \wedge \text{ward}(s) = \text{wardlocation}(r)$)

Yuan and Tong point out the administrative efficiency of specifying policies using their ABAC approach in comparison to that of RBAC. As a case in point, consider Rule1 where wards(s) and wardlocation(r) can take on n values. In RBAC the roles and permissions that are needed to achieve the policy would be as follows in [Table 3.1](#).

In general, if there are k subject attributes ($SA_1, SA_2 \dots SA_k$) and m resource attributes ($RA_1, RA_2 \dots RA_m$), and if for each attribute, $\text{Range}()$ denotes the range of possible values it can take, then the respective number of roles and permissions needed to be created would be:

$$\bigcap_{i=1,k} \text{Range}(SA_i)$$

and

$$\bigcap_{j=1,m} \text{Range}(RA_j)$$

Table 3.1
RBAC Roles

| RBAC Roles |
|------------------------|
| Ward-1 Doctor |
| Ward-2 Doctor |
| ... |
| Ward-n Doctor |
| RBAC Permissions |
| Read, Ward-1 resources |
| Read, Ward-2 resources |
| ... |
| Read, Ward-n resources |

Therefore, as the number and range of values of attributes increase for the expression of an ABAC policy, the number of roles and assigned permissions needed to express an equivalent policy could grow exponentially.

3.5 Hierarchical Group and Attribute-Based Access Control

Servos and Osborn [10] present a formal model that adds additional capabilities to the set provided by ABAC, by including hierarchies for attributes. Attribute inheritance can be implemented for any attributes in the system, through user and object groups as well as environmental attributes. In addition, the model separates out two classes of attributes

not used in most other models—administrative and connection. Strong typing for attributes is also included to reduce errors in policy specification. Collectively, these features enable Hierarchical Group and Attribute-Based Access Control (HGABAC) to reduce the number of attribute and group assignments needed to implement security policies, and also allow it to emulate DAC, MAC, and most aspects of RBAC.

The basic elements of HGABAC include a collection of sets used in specifying access rules: users, objects, operations, policies, permissions, and sessions. Users, objects, and operations are essentially the same as defined in conventional access control models. Other concepts in HGABAC are central to its utility:

- Sessions are specified in terms of user ID, connections, and attributes, where a user may activate only a subset of their attributes for a particular session. Thus a session is represented as a triple $\langle \text{user}, \text{active attributes}, \text{connection attributes} \rangle$. Only the active attributes are used in evaluating policies to determine permissions available to the session.
- Policies are sets of policy strings in the rule notation defined in the paper, where only certain policy strings may be active for a session. For example, a policy may be the string “`user.id = object.owner`”.
- Permissions are tuples containing a policy string and operation, as $\langle \text{policy}, \text{op} \rangle$. Access is allowed only if there is a permission containing a policy string that is satisfied by some set of active attributes in the user’s session, along with the attributes of the object being accessed, current connection state, and the administrative and environment attributes. Thus for example a permission for write access to an object could be specified as $\langle \text{user.id} = \text{object.owner}, \text{write} \rangle$.

Attributes in HGABAC are triples of the form $\langle \text{attribute name}, \text{value}, \text{type} \rangle$. Attribute names are unique identities and values are atomic values from an unordered set, or `null`. Types for the values are from system defined data types. HGABAC uses a more fine-grained division of attributes than other ABAC models:

- User attributes (UA): $\langle \text{name}, \text{type} \rangle$ —Values are assigned to users directly or to groups.

- Object attributes (OA): $\langle name, type \rangle$ —Values are assigned to objects directly or to object groups.
- Environment attributes (EA): $\langle name, value, type \rangle$ —Record the current state of system environment properties. Each element of EA is an implementation-defined globally unique name.
- Connection attributes (CA): $\langle name, type \rangle$ —Provide information on connections within the system. Each connection name is an implementation-defined globally unique name. Properties for connection attributes are system-dependent, but are expected to at least include a session ID.
- Administrative attributes (AA): $\langle name, value, type \rangle$ —are defined by administrators. Each element is an implementation-defined globally unique name. Administrative attributes apply to all policies that reference them, and will change at runtime depending on actions of administrators and implementation characteristics.

HGABAC also designates user groups (UG) and object groups (OG) and three relations specified as triples of element name, attribute name, and value. There are two direct assignment relations: user attribute assignments (UAA) and object attribute assignments (OAA). A special relation for user group attribute assignment (UGAA) is a triple $\langle group\ name, attribute\ name, values \rangle$, where there is only one triple in UGAA for any group name/ attribute name pair. A similar relation exists for object group attribute assignment (OGAA). A group hierarchy is represented as a directed acyclic graph specifying inheritance relations.

The central component of HGABAC is a set of mapping functions used in checking attributes for policy rules:

- Direct—produces the $(name, value)$ pairs for any of the relations UAA, OAA, UGAA, OGAA.
- Consolidate—from a collection that may contain multiple instances of $(name, value)$ pairs with the same name occurring more than once, produces a set of $(name, value)$ pairs where each name occurs only once.
- Member—maps a user or object to the set of groups to which they belong.
- Inherited—maps a user, object, or group to the set of attributes assigned indirectly through the hierarchy.

- Effective—maps a user, object, or group to their effective attributes (those assigned directly or inherited).
- Name—produces the first element of a $(name, value)$ pair.
- Parents—maps a group to its set of parents.
- Authorized: $P, S, O \rightarrow \{true, false, undef\}$ —evaluates the policy rules, where P is the set of policies, S the set of sessions, and O the set of objects. The function returns $\{undef\}$ if it is not possible to evaluate the policy, for example where an attribute is not present in the set of attributes for the session.

HGABAC is shown to reduce the number of attribute assignments, compared with an ABAC system, without inheritance. It also allows simplified representation of DAC and MAC rules. The provision of inheritance in HGABAC provides for easy specification of RBAC hierarchies as well. It thus provides a great deal of flexibility in access control, but certain important functions have not been incorporated yet. These include separation of duty (a standard function of RBAC), delegation, and improved control of administrative function. These functions are being investigated in continuing research.

3.6 Label-Based ABAC Model with Enumerated Authorization Policy

As noted previously, one approach to implementing ABAC is to use enumerated relations, rather than rules based on logic predicates. This approach makes it possible to represent authorization relations as graphs, so graph algorithms may be applied, often resulting in dramatic reductions in processing time to determine access permissions. Most ABAC models that have appeared so far have been rule-based. NGAC, as discussed previously, is based on enumerated relations and provides great expressive power for policies. Another example of the enumerated approach is the label-based access control (LaBAC) of Biswas, Sandhu, and Krishnan [4]. LaBAC can be considered a very basic ABAC, or a simplified instantiation of NGAC, because it uses only two attributes: a user label ($uLabel$) and an object label ($oLabel$). LaBAC represents authorization policies using enumerations of these two attributes. Despite its simplicity, this model has considerable expressive power, and it has

been shown that traditional RBAC and lattice-based access control can be configured as LaBAC.

LaBAC is designed as a family of models, with a basic form that presents one of the simplest possible attribute-based designs, to a complete model that can implement constraints and hierarchies. The inclusion of hierarchies and constraints has a significant impact on the LaBAC model's ability to express policy.

- Basic—LaBAC₀ includes the two attribute types needed to define a LaBAC model.
- Hierarchical—LaBAC_H adds hierarchies.
- Constraints—LaBAC_C includes constraints.
- LaBAC₁—Includes all of the above features.

A policy in LaBAC is defined as a subset of the tuples from the universe of possible combinations of user and object labels, $UL \times OL$. Authorization is determined by a function—is_authorized(s, a, o)—such that a subject s is allowed action a if the following conditions hold:

- s is assigned a value ul ;
- o is assigned a value ol ;
- the policy for action a contains the tuple (ul, ol) .

Some interesting relationships with other models can be shown [4]. Traditional lattice-based access control is defined with levels such as confidential, secret, and top secret, where a user's clearance must dominate a resource's classification. It can be shown that LaBAC₁ can implement this traditional model. Similarly, conventional RBAC can be implemented if the LaBAC system includes hierarchies, that is either LaBAC_H or LaBAC₁. Finally, NGAC (policy machine), which is more general than LaBAC, can implement LaBAC_H, although the converse relationship does not hold (LaBAC cannot implement NGAC).

3.7 Hybrid Designs Combining Attributes with Roles

No access control approach is best in all applications—it is usually necessary to analyze characteristics of the application domain and consider design tradeoffs. There are also some applications where the

most effective approach will be to implement a hybrid architecture, combining the features of different schemes in a way that is suitable for meeting both security and performance requirements [11, 12]. Hybrid architectures can be designed in several different ways, with tradeoffs that affect ease of deployment, management, and analysis of security properties. In this section, we review the access control system structure taxonomy defined by Kuhn, Coyne, and Weil [11], with the implications and possible applications of each. To define taxonomy at an appropriate level of abstraction, access control structures are analyzed in terms of the mapping of user IDs, roles, and attributes to permissions. [Table 3.2](#) shows the possibilities. Structures 7, 8, and 9, which will be considered in following sections, are three possible types of full hybrid RBAC/ABAC designs, because they include user ID, roles, and attributes. Systems have been developed which fall into each of the types defined, and detailed formal models [Sandhu] for them are beginning to appear as well.

- Structure 0 is undefined because a system with no user IDs, roles, or attributes clearly has no access control.
- Structure 1 can be considered a pure ABAC system, and is probably what is most often referred to when discussing ABAC. Only attributes are used in determining permission sets.
- Structure 2 has roles but no user IDs or attributes. This design might be useful as a way of grouping roles, but it is listed as undefined from the standpoint of access control because any user or process, with any attributes, may access any permission. In other words, if all permissions are accessible, then no access control is present.

Table 3.2
Combination Strategies and Options for Integrating Attributes with RBAC [11]

| Option | U | R | A | Model | Permission Mapping |
|--------|---|---|---|---------------------------|---|
| 0 | 0 | 0 | 0 | undefined | — |
| 1 | 0 | 0 | 1 | ABAC-basic | $A_1, \dots, A_n \rightarrow \text{perm}$ |
| 2 | 0 | 1 | 0 | undefined | — |
| 3 | 0 | 1 | 1 | ABAC-RBAC hybrid | $R, A_1, \dots, A_n \rightarrow \text{perm}$ |
| 4 | 1 | 0 | 0 | ACLs, capabilities | $U \rightarrow \text{perm}$ |
| 5 | 1 | 0 | 1 | ABAC-ID | $U, A_1, \dots, A_n \rightarrow \text{perm}$ |
| 6 | 1 | 1 | 0 | RBAC-basic | $U \rightarrow R \rightarrow \text{perm}$ |
| 7 | 1 | 1 | 1 | RBAC-A, dynamic roles | $U, A_1, \dots, A_n \rightarrow R \rightarrow \text{perm}$ |
| 8 | 1 | 1 | 1 | RBAC-A, attribute-centric | $U, R, A_1, \dots, A_n \rightarrow \text{perm}$ |
| 9 | 1 | 1 | 1 | RBAC-A, role-centric | $U \rightarrow R \rightarrow A_1, \dots, A_n \rightarrow \text{perm}$ |

* U = user/subject ID; R = role; A = attributes

- Structure 3 is a weakly constructed hybrid model. Roles may be treated essentially the same as another attribute, but roles are not tied to a user ID.
- Structure 4 includes both a traditional access control list or capability design that both specify a mapping between users and permissions. ACLs attach user IDs to resources, while capabilities attach permissions to user IDs.
- Structure 5 simply treats the user ID as a special case among the attributes. This approach could be regarded as an attribute-based version of Option 4, and presents some interesting possibilities. ACLs might contain attributes in addition to user IDs, limiting the conditions under which each user on the ACL could access the resource. Alternatively, adding attributes to capability lists could provide additional restrictions under which the user could exercise access right provided by the capability.
- Structure 6 is a traditional RBAC system, in which access is determined based on role and user ID, where user ID determines possible roles and roles determine permissions.
- Structures 7, 8, and 9 define different ways in which ABAC and RBAC can be combined. Option 7, which we refer to as dynamic roles, adds roles to a basic RBAC structure. Instead of using only

the user ID to determine the user's possible roles, this option includes attributes in the decision. An advantage of this approach is that it can be layered on top of a conventional RBAC system. An early implementation of such a design was the U.S. Navy's Enterprise Dynamic Access Control (EDAC) system [13], described in more detail below.

- In structure 8, referred to as attribute-centric, roles are treated as attributes and handled in the same way as attributes, such as location or time of day. Clearly there must be some binding between user ID and role; this could be accomplished in different ways and no particular approach to this binding is presumed. In this case, the role may be more appropriately thought of as a position name so as not to confuse it with a traditional RBAC system. A major distinction is that the role is no longer a set of permissions, as in RBAC, because the position/role is only part of the permission set determination.
- The role-centric structure 9 evaluates ABAC rules after a user has been authorized for a particular role. Thus a user's permission set is constrained to the maximum set available to all of the user's roles. Permissions available to the user in this design are the intersection of P and R , where P is the set of permissions assigned to the subject's active roles and R is the set of permissions authorized by the ABAC rules. A user's set of possible roles therefore determines the complete set of available permissions. Note that assigning all users to a single role containing all permissions is equivalent to an ABAC-ID (structure 5).

The various structures defined above for ABAC and RBAC integration will have very different impacts on system implementation and usability. Thus it is important to understand their characteristics to recognize where they are appropriate solutions. Examples of these hybrid models can be found in the literature, and we highlight several of these to illustrate the features of hybrid structures.

3.8 ABAC and RBAC Hybrid Models

From the earliest days of role-based access control, it was recognized that a pure role-oriented structure for permissions would not be enough for many applications. Consequently, the earliest RBAC models contained provisions for additional constraints on roles that could be applied to deal with rapidly changing conditions that could matter for authorization. In particular, time of day is significant in many enterprises —an employee may have full access to their permissions during working hours, but may be limited to some degree outside of these hours. The basic RBAC model in [14] included provision for these and other constraints. This model used the authorization rule $\text{exec}(s, t) \Rightarrow t \in TA(AR(s))$, allowing access to a transaction t by a subject s only if t was contained in the transactions authorized, TA , for active role of subject s , $AR(s)$, noting that:

because the conditional $[\Rightarrow]$ is “only if”, this rule allows the possibility that additional restrictions may be placed on transaction execution. That is, the rule does not guarantee a transaction to be executable just because it is in $TA(AR(s))$, the set of transactions potentially executable by the subject’s active role. For example, a trainee for a supervisory role may be assigned the role of “Supervisor”, but have restrictions applied to his or her user role that limit accessible transactions to a subset of those normally allowed for the Supervisor role. [14]

“Temporal RBAC” models were also developed, to explicitly include time as a component of the access control structure. Other models included provisions for more easily including extensively varying attributes, such as geographic office location, among otherwise stable and consistent roles. This pattern of variable versus stable characteristics is common in role structures. The process of designing and structuring RBAC hierarchies, known as role engineering [15], developed to deal with the often challenging problem of producing an RBAC structure that could provide the required level of control without being too cumbersome to implement.

3.9 Complexities of RBAC Role Structures

Role-based access control became the most widely used form of advanced access control because it made security administration vastly easier than using access control lists. By adding a layer of abstraction,

the role as a set of permissions, it became possible for administrators to easily assign or remove the permissions associated with a particular job function or process. It also made auditing permissions much faster. Determining the set of permissions available to every user is essential for organizations to determine their risk exposure, that is, the maximum set of data or processes that could be potentially accessed improperly by a malicious user (or through a compromised user account). Instead of scanning ACLs of every resource and collecting user IDs, then sorting permissions by user, RBAC made it possible to simply list a user's roles, and then list the permissions of each role. Because a user has a small set of roles, and roles are also hierarchical, computing a user's available permissions is trivial.

The price for this simplicity of audit is the potential difficulty of establishing roles initially. Because a role is a set of permissions, with n permissions the number of potential roles is 2^n , or $2^n - 1$ if we exclude the “null” role that has no permissions. Establishing role hierarchies helps to reduce this size to some degree, but a large complex organization can have a huge number of distinct jobs with different sets of permissions. With the popularity of RBAC, a significant industry developed related to role engineering, the process of determining the role structure needed to implement a particular policy. This field includes role mining tools, which often use graph algorithms to convert a large number of sets of permissions into role hierarchies. A commonly heard term with such tools is the role explosion, or proliferation of roles needed to reflect sets of permissions for every position. Even with automated tools, a significant degree of adjustments and revisions may be needed to get the role structure correct.

The frequent difficulty of initial analysis and design of the role structure is one of the most common complaints against RBAC, and one of the motivations for ABAC. A secondary concern is that RBAC systems may not provide adequate support for dynamic environment characteristics such as time of day. Such dynamic values may be needed to fully determine permissions at a particular point in time. ABAC offers the potential to resolve both of these concerns. ABAC systems are easy to set up, since the only thing needed is to associate attributes with permissions. And since attributes can be either static or dynamic, ABAC policy rules can include continuously changing values such as time of day. The price paid for this flexibility is the difficulty of analyzing

permissions in ABAC—at any given time, which resources can a given user access, and the converse—for each resource, what users have access? These questions are critical in estimating the risk that some resources will be improperly accessed. One of the complaints with current ABAC implementations is that an overnight run may be needed to determine user–permission links. Worse, in systems where attribute values change rapidly, it may take longer to analyze permissions than it does for a large set of attributes to change their values. As a result, administrators may not be able to determine who has access to what resources at a given moment.

3.10 Complexities of ABAC Rule Sets

One of the more challenging aspects of ABAC is the problem of determining the current set of permissions available to all users. In existing ABAC implementations, this process may require significant time to run, but may be essential because of enterprise or government rules. For example, regulations may require that users have access only to a particular limited set of resources (the principle of least privilege). Thus it is necessary to run periodic audits, possibly daily, to determine the set of resources accessible to each user by virtue of the current settings of user, object, and environment attributes. There may be scalability issues for auditing basic ABAC systems with flat (i.e., non-hierarchical) sets of resource and user attributes. As such, these may be considered worst case calculations. Possible means of reducing the computational requirements to make ABAC systems more scalable are discussed also.

ABAC makes it easy to specify access rules, but to determine the permissions available to a particular user a potentially large set of rules might need to be executed in exactly the same order in which the system applies them. This can make it complex to determine risk exposure for a given employee position. Let *security state* refer to the linking of users and resources each user has permission to access at a specified time. If rules and attributes are unchanging, then we can determine the security state of a system by testing the rules or attribute sets for each protected resource with the attribute values for each user. That is, for each user instantiate each rule set with the current values of all attributes and

evaluate each rule. If there are U users and R rule sets (one set of rules for each resource), then this is $U \times R$ tests.

Consider now the task of determining a complete picture of possible security states of the system, for a flat (nonhierarchical) attribute set. This evaluation would be required to understand the maximum risk exposure of the organization, that is what permissions are possible for each user in the organization to obtain? This type of audit is often a concern for financial institutions, which may have very large monetary risk if users are able to exceed their expected authority with respect to managed funds. These firms must be able to show that any given user has a limited set of permissions, thus limiting the potential damage from an errant employee. To do this we must consider not just the current state of user and environment attributes, but possible off-nominal states. For example, a user may be currently working on project A, but may also be authorized to work on B or C, so the possible project attribute for this user has three possible values. Similarly, the time of day attribute may be used with two possible values, working hours and nonworking hours. If the user's access permissions depend on just these two attributes, we need to consider six possible attribute states to determine what the user may eventually be able to access or be prohibited from accessing, that is the security state for this user.

Computational requirements can be reduced by limiting the size of U or R , into groups or hierarchies such that users or resources can be treated as indistinguishable for the purpose of rule evaluations. In some cases, resources may be grouped, reducing the size of R in the calculations above. For example, where a particular set of files or database tables are always used together, a single rule set can apply for that set of files or tables. This may be difficult in many cases, since the sets of files or resources used in different tasks are usually not disjointed. In this situation it may be practical to establish meta-resources where a single rule set applies to the meta-resource, reducing R . Given the magnitude of the computational problem with flat sets of users and resources, it will also be necessary to reduce the size of U . This may be done on the basis of a variety of characteristics as dictated by the application such as geographic location, but in most organizations the relevant characteristic in determining user privileges is job position or role. For example, with 100,000 users separated into 100 roles, $U \times R$ is reduced by a factor of 1,000. The tradeoff in this case is in applying

effort initially to determine a user's role rather than basing decisions entirely on attributes to be evaluated at runtime. Using meta-resources in conjunction with roles may thus reduce the computational problem to a level that is tractable for many applications.

3.11 Dynamic Roles

As noted previously, the dynamic role scheme has the structure: $U, A_1, \dots, A_n \rightarrow R \rightarrow perm$. That is, the user ID and a set of attributes determine the role, which in turn determines the set of permissions. User ID may be null if specified in the rule set, so that only attributes are used to determine the role. Dynamic roles may be the best choice for rapidly changing environments which also need structured sets of permissions.

One of the earliest systems to merge RBAC and ABAC concepts is the Enterprise Dynamic Access Control (EDAC) System [13, 16–19], developed by Richard Fernandez at the U.S. Navy Space and Naval Warfare Systems Command. EDAC has been shown to be consistent with the ANSI 359 definition of RBAC, but it provides the addition of dynamic selection of roles. EDAC was designed to address the need for real-time variations in access control requirements, especially in response to changing security threats. It controls access to roles using the criteria attributes, environmental conditions, business rules, questionnaire, and workflow progress. EDAC was also based on the recognition of the fact that military and other large organizations may have exceedingly complex policies that can lead to the role explosion problem with conventional RBAC. By allowing dynamic changes in the way roles are made available, EDAC addresses the need for rapid changes in access rules as well as simplifying policy design for very large enterprises.

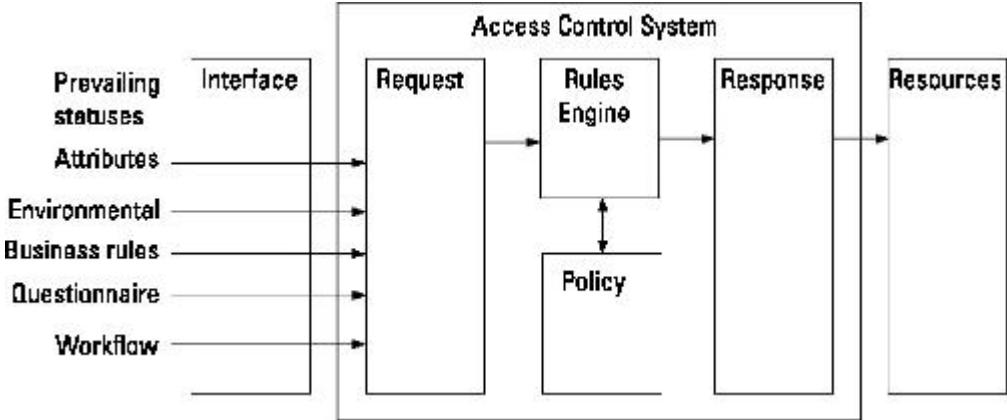


Figure 3.4 Enterprise dynamic access control [16].

EDAC is structured as shown in [Figure 3.4](#). In this model, users are assigned into roles based on any combination of the five criteria on the left of the diagram. Roles apply to one or more resources, or to the enterprise as a whole, which may be structured into a hierarchy as with conventional RBAC. The interface refers to a set of sensors or other components collecting values of the criteria used in decisions. These are then structured into a request for the rules engine, which determines the appropriate response based on a policy.

To see how this structure maps to RBAC, consider the resource diagram in [Figure 3.5](#). Three roles can apply to the door resource: admin, user, and guest. Permissions are composed of operations and objects, and roles are defined as a set of these permissions. The criteria on the left are thus used to determine roles. Although there are five criteria listed, they can be clearly mapped to the ABAC definition presented in [Section 3.1](#).

- *Attributes*: These correspond to user attributes in the ABAC model.
- *Environmental*: Environment conditions are represented as environment attributes in ABAC.
- *Business rules*: As defined in EDAC, business rules are a convenient way to compose complex parts of access control rules. An example given in [\[18\]](#) is a risk assessment, composed of attributes for user job position and Homeland Security advisory level, e.g. user job = program manager and advisory level = high → risk assessment = 8, with other combinations of user position and advisory level producing different risk assessments. These conditions could be encoded directly in

ABAC rules, but EDAC uses this structure to simplify administration.

- *Questionnaire*: This attribute relates to a user input for specific questions relevant to the decision, for example, whether a user has indicated that they have detected a particular problem to be investigated.
- *Workflow*: Multistage processes involving more than one user may be involved in determining access requirements, so this attribute indicates the current state of a workflow. For example, a workflow may be user problem report → manager check → maintenance request → maintenance completion. The workflow attribute would indicate the status of this series of events in determining access.

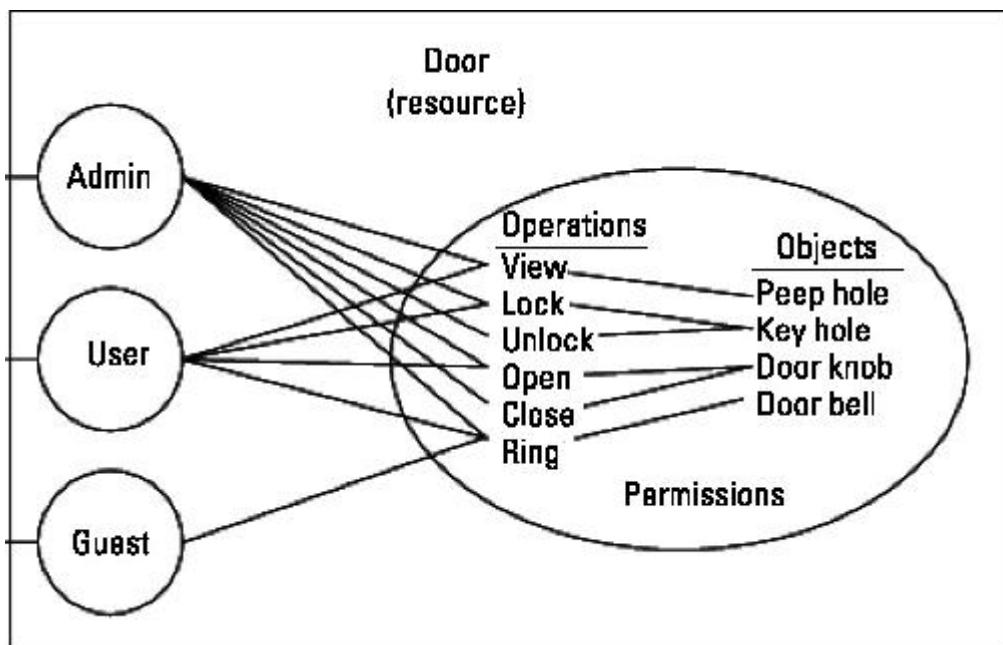


Figure 3.5 Resources in EDAC [16].

To see how these concepts work in EDAC, consider the policy shown in [Figure 3.6](#) (from EDAC version 2 overview). It is important to note the distinction between this and an attribute-centric approach where a role is just one of many attributes. In this case, position titles such as electronics technician are used as attributes, but a full role structure is present, with roles being made up of sets of permissions consistent with the RBAC standard definition.

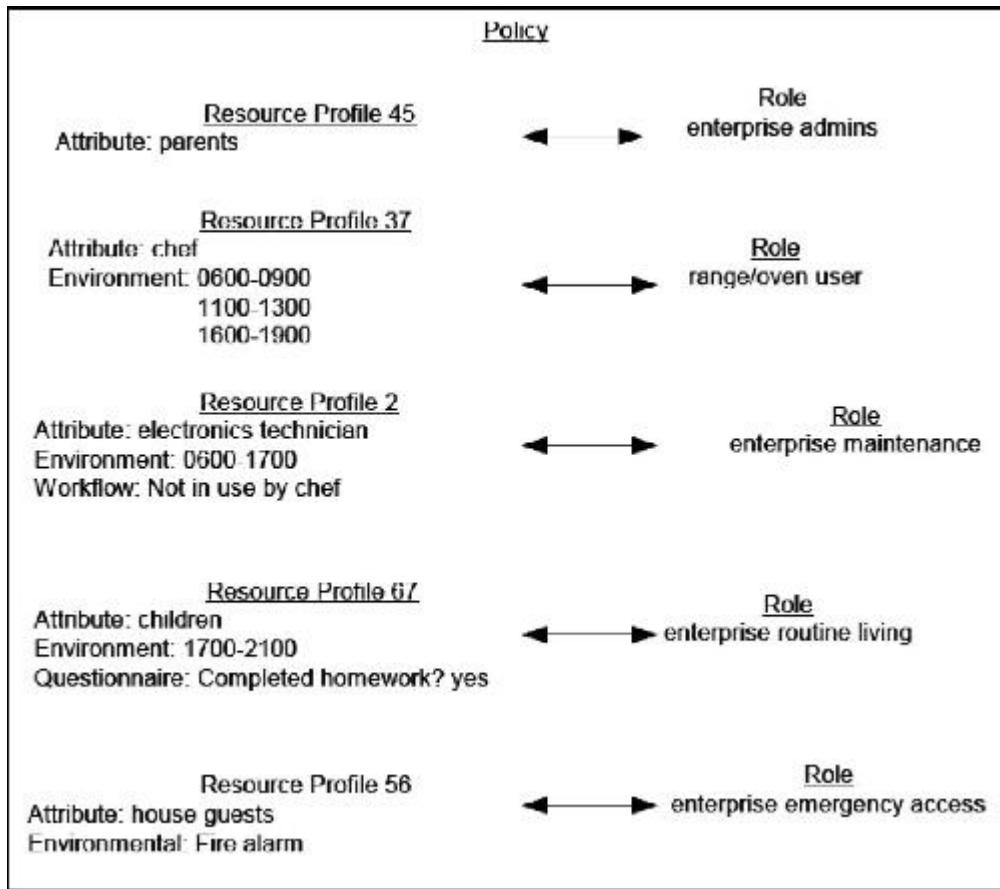


Figure 3.6 Example policy in EDAC [16].

Although not shown in Figure 3.6, the roles can be hierarchical. Thus the enterprise role admin can be expected to have a large set of roles below it in the hierarchy, while the resource role range/ oven user would have a smaller set. EDAC thus provides a fine-grained control of permissions, while simplifying administration. Roles in this example are likely to have a relatively stable set of permissions, so an RBAC structure makes sense, but there is also a need for dynamic changes in attributes such as time of day, and status of resources (such as the oven not being used by chef). Consequently, EDAC provides a very practical solution.

Another dynamic role implementation has been proposed by Al-Kahtani et al. [20], using a specified set of rules for an organization to dynamically assign roles. The design described uses user attributes and applies additional constraints as required by the enterprise. A seniority mechanism allows construction of hierarchical relationships among roles. Al-Kahtani et al. also show how to use such a dynamic role system to implement traditional mandatory access control (MAC) rules.

Kern et al. [21] also describe a rule-based approach to dynamic assignment of roles within an RBAC system. Kim, Joshi, and Kim use the dynamic role approach to support agent based cooperation systems [1, 22]. Dynamic role assignment is shown to be particularly useful in this context, because constant changes in a cooperative environment require appropriate permission changes.

Huang et al. [23] present a somewhat different approach to combining attributes with RBAC. This model uses two levels, an aboveground function that is a conventional RBAC system with some environment constraints, and an underground level that uses attributes to construct the roles at the aboveground level. That is, attributes are used in rules to infer a role structure, so this can be considered a form of dynamic role assignment through attributes. Although designed in the context of industrial control systems, the approach can be applied to a wide range of applications. This approach retains the simplicity of RBAC administration, and uses attributes to make role design easier to manage.

3.12 Role Centric Structure

The role-centric structure is $U \rightarrow R \rightarrow A_1, \dots, A_n \rightarrow perm$. That is, a user ID determines a set of possible roles, and then attributes are used in authorizing permissions available to the user's active roles. Permissions in this case are determined by the intersection of P and R , where P is the set of permissions assigned to the subject's active roles and R is the set of permissions specified by the applicable ABAC rules. Thus roles determine the maximum set of permissions a user may access, and attributes are used as constraints to limit this set further. As noted previously, a limited form of this design was actually introduced in the earliest days of RBAC [24]. A role-centric design extends the approach to include possibly many more attributes as part of the basic access decision rather than only a few exceptions or constraints. Because the user's role set determines the maximum set of available permissions, the principle of least privilege is maintained, limiting risk exposure and facilitating review of user permissions.

An extension of the TRON operating system provides an example of a role-centric approach to hybrid RBAC/ABAC capabilities. To address the privacy and security requirements of HIPAA and similar legislation

in other countries, the eTRON access control architecture described in [25, 26] implements a RBAC ontology based on the Health Level Seven catalog of roles and permissions for the health care field. RBAC elements are represented using the Web Ontology Language with RDF/XML syntax. The structure of the system is illustrated in Figure 3.7. Authentication uses a token providing a 128-bit unique identifier, enabling authorization by the access manager for roles available to the user. The context policy repository provides for additional constraints using attributes, consistent with the role-centric ABAC hybrid approach.

The eTRON architecture is able to easily meet privacy requirements for health records because the RBAC structure limits the range of analysis required to audit the links between users and permissions. By checking the roles of each user, it is possible to bind the maximum set of permissions available, and constraints are used to further limit permissions in daily use.

A different implementation of the role-centric approach is used in the physical-logical access control system defined by the Physical-Logical Security Interoperability Alliance (PSIA) [27]. The PSIA Physical-Logical Access Interoperability (PLAI) specification uses RBAC and attributes in access control for facilities management. The PLAI specification first requires the definition of roles that handle both logical (information) and physical access permissions. Two policy engines, one for logical and one for physical assets, then apply constraints using the personal attributes of the user and situational attributes, as shown in Figure 3.8. An interesting aspect of this design is that the policy engines for physical and logical access may coordinate rules in decisions, for example where a particular data access constraint implies some associated limit on physical access.

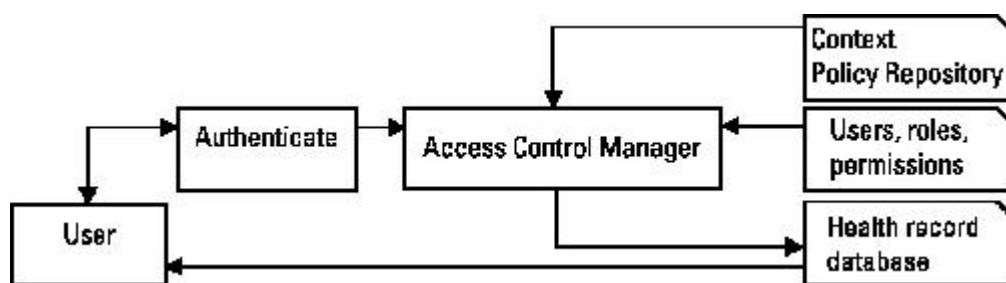


Figure 3.7 eTRON access control architecture.

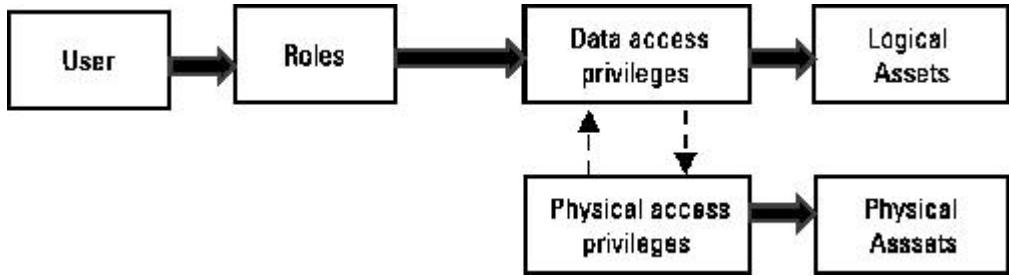


Figure 3.8 PSIA Physical-Logical Access Interoperability (PLAI) architecture.

It is also possible to provide for both role-centric and dynamic role designs in the same system. One example is provided by the Empower ID RBAC/ ABAC hybrid enterprise authorization architecture [28]. This hybrid essentially uses a dynamic role authorization engine as the first component facing the user, with an ABAC system applying possible additional constraints after the user's active role set has been determined, as shown in [Figure 3.9](#).

This design invokes two structures from those defined previously, with dynamic roles as the first component and the role-centric structure as the second:

$$U, A_1, \dots, A_n \rightarrow R \rightarrow A_1, \dots, A_n \rightarrow perm$$

In this case, policy rules involving attributes are applied on both sides of the role selection, to provide flexibility in permission assignment.

3.13 Attribute-Centric Structure

The attribute-centric structure is $U, R, A_1, \dots, A_n \rightarrow perm$. In essence, this is simply a pure ABAC system, where one of the attributes is referred to as a role. Strictly speaking, the role is not a role in the RBAC sense, because RBAC roles are sets of permissions. But the term role is widely used in access control designs and often used in a loose sense related to a user's position in an organization. Essentially any ABAC system can provide the attribute-centric structure, when an attribute for user role or position is included.

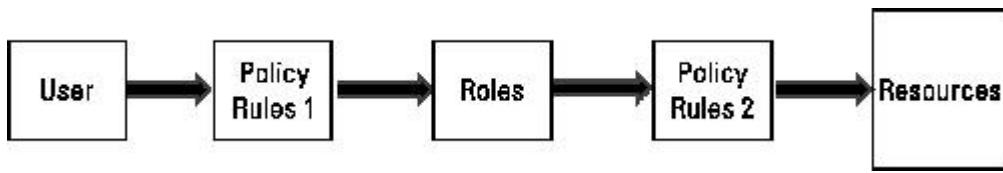


Figure 3.9 RBAC/ABAC hybrid architecture.

3.14 Conclusion

ABAC is an extremely flexible approach to access control, as seen in this chapter. Many different architectures are possible, with varying strengths and weaknesses, allowing organizations to select the design that works best in their environments. Some designs combine features of ABAC and RBAC into a hybrid, balancing features of both, and these hybrid designs can be structured in many different ways as well (see [Table 3.1](#)). Factors that may be considered in selecting an ABAC design include the (temporal) stability of various attributes, needs for integration with other models (such as multilevel security), frequency of changes in the environment, and needs for auditing and review.

References

- [1] Hu, V. C., et al., “Guide to Attribute-based Access Control (ABAC) Definition and Considerations,” (draft), *NIST Special Publication 800-162*, January 2014.
- [2] *Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA)*, INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.
- [3] Anderson, A., et al., *eXtensible Access Control Markup Language (XACML) Version 1.0*, OASIS, February 18, 2003.
- [4] Biswas, P., R. Sandhu, and R. Krishnan, “Label-Based Access Control: an ABAC Model with Enumerated Authorization Policy,” *Proc. 2016 Association for Computing Machinery International Workshop on Attribute Based Access Control*, March 2016, pp. 1-12.
- [5] Jin, X., R. Krishnan, and R. Sandhu, “A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC,” *IFIP Annual Conference on Data and Applications Security and Privacy*, July 2012, Springer Berlin Heidelberg, pp. 41-55.
- [6] Shen, H.-b., and F. Hong, “An Attribute-Based Access Control Model for Web Services,” *PDCAT’06*, IEEE, 2006, pp. 74-79.

- [7] Yuan, E., and J. Tong, “Attributed Based Access Control (ABAC) for Web Services,” *Proc. 2005 IEEE International Conference on Web Service*, IEEE, 2005.
- [8] Kerschbaum, F., “An Access Control Model for Mobile Physical Objects,” *Proc. 15th Association for Computing Machinery Symposium on Access Control Models and Technologies*, June 2010, pp. 193-202.
- [9] Liu, L., W. Xia, and L. Shen, “An Adaptive Multi-Channel MAC Protocol with Dynamic Interval Division in Vehicular Environment,” *1st International Conference on Information Science and Engineering (ICISE)*, IEEE, December 2009, pp. 2534-2537.
- [10] Servos, D., and S. L. Osborn, “HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control,” *International Symposium on Foundations and Practice of Security*, Springer International Publishing, November 2014, pp. 187-204..
- [11] Kuhn, D. R., E. J. Coyne, and T. R. Weil, “Adding Attributes to Role-Based Access Control,” *IEEE Computer*, Vol. 43 No. 6, pp. 79-81.
- [12] Coyne, E., and T. R. Weil, “ABAC and RBAC: Scalable, Flexible, and Auditable Access Management,” *IT Professional*, Vol. 15 No. 3, 2013, pp. 14-16.
- [13] Fernandez, R., *Enterprise Dynamic Access Control, Version 2 Overview*, Space and Naval Warfare Systems Center, 2006.
- [14] Ferraiolo, D. F., and D. R. Kuhn, “Role-Based Access Controls,” *Proc. 15th National Computer Security Conference*, National Institute of Standards and Technology, National Computer Security Center, October 1992, pp. 554-593.
- [15] Coyne, E. J., “Role engineering,” *Proc. 1st Association for Computing Machinery Workshop on Role-based Access Control*, December 1996, p. 4.
- [16] [http://www.public.navy.mil/spawar/Pacific/TechTransfer/ProductsServices/Documents/TIPSheets/Enterprise_Dynamic_Access_Control_\(EDAC\)_96217_et_al_TIP.pdf](http://www.public.navy.mil/spawar/Pacific/TechTransfer/ProductsServices/Documents/TIPSheets/Enterprise_Dynamic_Access_Control_(EDAC)_96217_et_al_TIP.pdf)
- [17] Fernandez, R., “Government Off-The-Shelf-Solution (GOTS) for Role-Based Access Control (RBAC),” *2004 Command and Control Research and Technology Symposium: The Power of Information Age Concepts and Technologies*, 2004.
- [18] Fernandez, R., *Enterprise Dynamic Access Control (EDAC) Case Study*, Space and Naval Warfare Systems Center, 2005.
- [19] R. Fernandez, *Enterprise Dynamic Access Control (EDAC) Compliance with the American National Standards Institute (ANSI) Role Based Access Control (RBAC)*, Space and Naval Warfare Systems Center, 2005.

- [20] Al-Kahtani, M., et al., “A Model for Attribute-Based User Role Assignment,” *Annual Computer Security Applications Conference*, IEEE, 2002, pp. 353–362.
- [21] Kern, A., and C. Walhorn, “Rule Support for Role-Based Access Control,” *Proc. 10th Association for Computing Machinery Symposium on Access Control Models and Technologies*, June 2005, pp. 130-138.
- [22] Kim, M., J. B. Joshi, and M. Kim, “Access Control for Cooperation Systems Based on Group Situation,” *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, Springer Berlin Heidelberg, November 2008, pp. 11-23.
- [23] Huang, J., et al., “A Framework Integrating Attribute-Based Policies into Role-Based Access Control,” *Proc. 17th Association for Computing Machinery Symposium on Access Control Models and Technologies*, June 2012, pp. 187-196.
- [24] Ferraiolo, D. F., D. R. Kuhn, and R. Sandhu, “RBAC Standard Rationale: Comments on a Critique of the ANSI Standard on Role-Based Access Control,” *Proc. IEEE Symp. Secur Priv*, November 2007, pp. 51-53.
- [25] Khan, M. F. F., and K. Sakamura, “Tamper-Resistant Security for Cyber-Physical Systems with eTRON Architecture,” In 2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS), pp. 196-203.
- [26] Khan, M. F. F., and K. Sakamura, “Context-Aware Access Control for Clinical Information Systems,” *2012 International Conference on Innovations in Information Technology (IIT)*, IEEE, March 2012, pp. 123-128.
- [27] Soleimani, M., T. R. Weil, and E. Coyne, “Behind Closed Doors: Let’s PLAI,” *IT Professional*, Vol. 17 No. 3, 2015, pp. 64-67.
- [28] <http://blog.empowerid.com/hs-fs/hub/174819/file-18506087-pdf/docs/empowerid-whitepaper-rbac-abac-hybrid-model.pdf>

4

ABAC Deployment Using XACML

4.1 Introduction

The Extensible Access Control Markup Language (XACML) standard provides a standardized framework for deployment of ABAC. The latest version of this OASIS [1] standard is XACML 3.0 [2]. XACML is by far the most widely used standard for the implementation/ deployment of ABAC, being used in quite a few commercial and open-source access control products. Our study of ABAC deployment using XACML will cover the following aspects:

- Business and technical drivers for XACML ([Section 4.2](#));
- XACML standard—components and their interactions ([Section 4.3](#));
 - XACML policy language model—components and elements;
 - XACML context (request and response)—obtaining authorization decision;
 - XACML framework (data flow model)—functional building blocks of deployment;
- ABAC deployment using XACML ([Section 4.4](#));
 - Access policy formulation and encoding;
 - Request/ response formulation;
 - Policy evaluation and access decision;
- Implementation of XACML framework ([Section 4.5](#));
 - Attribute support and management;
 - Advanced administration (delegation);
- Review & analysis ([Section 4.6](#));

4.2 Business and Technical Drivers for XACML

Access control in monolithic applications was specified and enforced using application-environment specific artifacts (e.g., access control lists (ACL) in file systems provided by operating systems or using hard-coded security constraints within the application code. When distributed systems using Web services technology were being built, the same approach was extended to each component of the distributed application. This approach resulted in three problems:

- It was getting expensive and unreliable to modify the security policy at each of the access enforcement points (because policies were specified close to or near each enforcement point)
- It was getting hard to keep a complete and consistent view of the global state of the entire distributed system. In other words, it was proving difficult to obtain a consolidated view of the safeguards in effect throughout the distributed system and by extension the enterprise information system.
- Having a consolidated view of the security policies and their enforcement assurance was not only needed for internal operational efficiency but also to cope with external pressures from consumers, shareholders and regulators who were demanding best practices in the protection of information assets of the enterprise.

To address the above problems, the strategy adopted was to completely externalize authorization from the application. This will enable policies to be reused across many applications, leading to greater consistency of access control rules and improved efficiency in maintaining them. The first generation of solutions driven by this access control strategy for distributed systems was based on general schemes and authorization frameworks (e.g., Ponder2 [3], and PERMIS [4]). The difficulty with these frameworks is that each of them specified their own policy language, enforcement technique and data format. The difference in access control methods (some based on MAC, some on DAC, etc) and the absence of a common access policy specification language made it difficult to exchange access control information between different security domains (e.g., conveying privileges between systems belonging to two different companies or even business units), thus inhibiting the goal of enforcing an enterprise-wide security policy across all systems.

XACML addressed the above problems through the following:

- XACML is an XML-based platform neutral language. It thus provides a simple, flexible way to express and enforce access control policies in a variety of environments, using a single, standardized common language.
- Its syntax and semantics can be extended to accommodate the unique authorization requirements in different application scenarios—that is support different types of access control methods including the most generic approach of providing access based on the different values of the attributes associated with all participating entities—subjects, resources, actions, and environmental variables.

Based on the above capabilities/features, XACML, in theory, can provide the definitive solution to access control problems in distributed, heterogeneous application environments. On the practical side, XACML has widespread support from the main platform and tool vendors, thus facilitating easy deployments.

4.3 XACML Standard—Components and Their Interactions

Before discussing the XACML standard and how it enables expression and enforcement of ABAC-based policies, it is necessary to provide a mapping of terms. The terminology mapping is needed since there is a slight difference between some of the terms used in the NIST ABAC publication *SP 800-162* [5] and in many parts of this book, and terms used in XACML Standard for referring to the same concept. This mapping is given in [Table 4.1](#) below:

The XACML model has three main components:

- *XACML Policy Language Model*—XML-syntax based language for specifying access control requirements in terms of attributes of subject, resource, action and environment.
- *XACML Context*—XML-syntax based formats for conveying access requests and access responses for authorization process.
- *Data Flow Model*—Management architecture describing the functional modules involved in authorization decision-making process and in creating policy/ attribute repositories. We refer to

the instance of a data flow model as XACML Framework in the rest of this chapter.

Table 4.1
ABAC Concepts Terminology Mapping

| Access Control Concept | ABAC Model Terminology (NIST SP 800-162 and This Book) | XACML Terminology |
|---|--|-------------------|
| 1. User making an access control request | Subject | Subject |
| 2. The operation to be performed on the requested IT entity | Action | Action |
| 3. The IT entity (database object or program) for which access is requested | Object | Resource |
| 4. The physical (e.g., time, date, etc) or the electronic (e.g., IP address) environment in which the requesting user is functioning. | Environment condition | Environment |
| Collective name to denote concepts 1-4 referred above | Element* | Category |

* Being a XML-based language, the building blocks that XACML uses to express an instance of an access control policy is called an element. This XACML element has no semantic relation to the ABAC model element referred here.

4.3.1 XACML Policy Language Model

The XACML Policy Language Model consists of three main components organized in a hierarchical structure. The three components are:

- Rule;
- Policy; and
- Policy Set.

The hierarchical structure linking these components is shown in [Figure 4.1](#).

From [Figure 4.1](#), it is clear that the lowest component in the hierarchy is the rule. Multiple (one or more) rules constitute a policy, or stated in another way, a policy is made up of multiple (one or more) rules. Multiple policies in turn constitute a policy set. In addition, a

policy set can also contain one or more policy sets within itself, thus allowing for nesting of policy sets. Any authorization system built using XACML standard makes use of the XACML policy language model components to build the authorization database called the XACML policy repository. The structure of any authorization database can have only two of the three components (i.e., policy set and policy) as the root of its hierarchical structure. This is due to the fact that a rule or a set of rules alone cannot constitute a policy repository. In other words, a rule is not a stand-alone entity within a policy repository but must always be encapsulated in a policy. This also implies that an XACML policy repository contains exactly either one policy or policy set as the root XML tag.

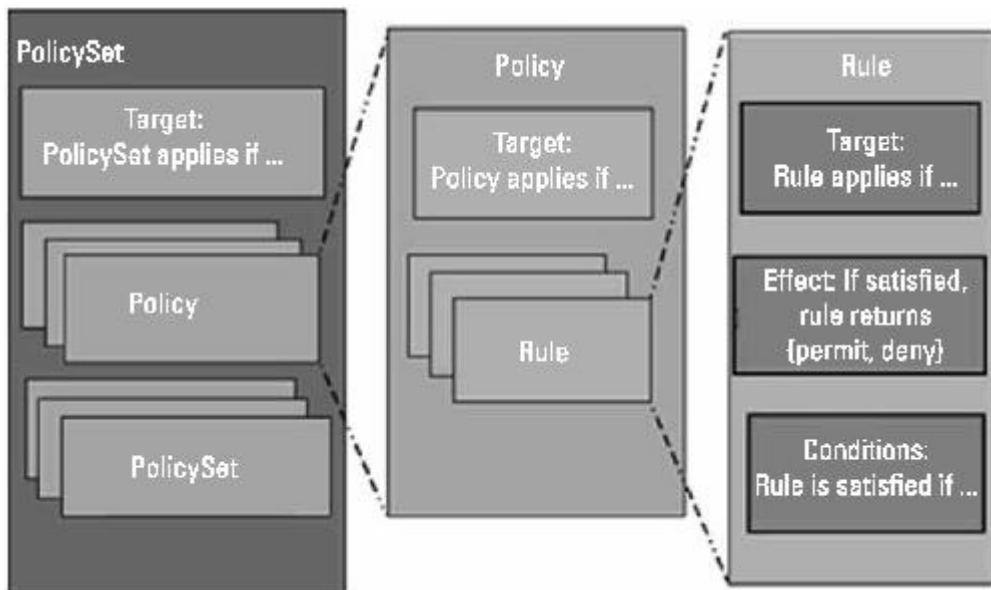


Figure 4.1 XACML policy language model components.

Since the purpose of [Figure 4.1](#) is to show the hierarchical structure of the components, only the major top-level elements within each component (rule, policy, and policy set) are shown. In addition, the policy and policy set components may optionally contain additional elements such as obligation expression and advice expression. [Figure 4.2](#) captures all the top-level elements of these three components and in addition shows the next-level elements (subelements) within the target element. This diagram will now form the basis for a brief description of each of the components along with their elements and subelements in XACML policy language model.

The rule component (or a rule) is composed of one target, one or more conditions, effect, one or more obligation expressions and one or more advice expressions. Every component except the effect is optional in a rule. The target contains the combination of attribute values associated with subject, resource and action for which the rule is intended to apply. The target element is also used in a policy or policy set and similar to its use in a rule expresses the attribute value combination the policy or policy set must meet in order to be applicable for a given access request. The target element is absent in a rule under two situations: (a) the rule is applicable to any access request—from any subject on any resource for any action under all environmental conditions (in this situation, the target in the parent policy is empty) or (b) the rule is applicable for the same situation for which its parent policy is applicable. In this scenario, the deemed target for the rule is the same as that of its parent policy element. The condition element in a rule is a Boolean expression used to further refine the applicability of the rule in addition to the predicates implied by its target. Being an optional element it may be absent in a rule instance. The next element in the rule is effect. The effect indicates the rule-writer’s intended consequence of a “TRUE” evaluation for the rule. The two allowed values for effect are permit and deny. The obligation expressions contain information for certain actions (called obligations) to be carried out either before or after fulfilling the access request by the access enforcement module PEP (explained in section 4.3.3). This information is also an outcome of the rule evaluation process, in addition to the access decision. Similarly, the advice expressions result in suggestions for PEP and unlike obligations may be safely ignored by PEP. The semantics for the target, the obligation expression and the advice expression elements is the same whether they are present in a rule, policy or policy set. Hence we omit discussion of these elements in our discussion of the policy and policy set.

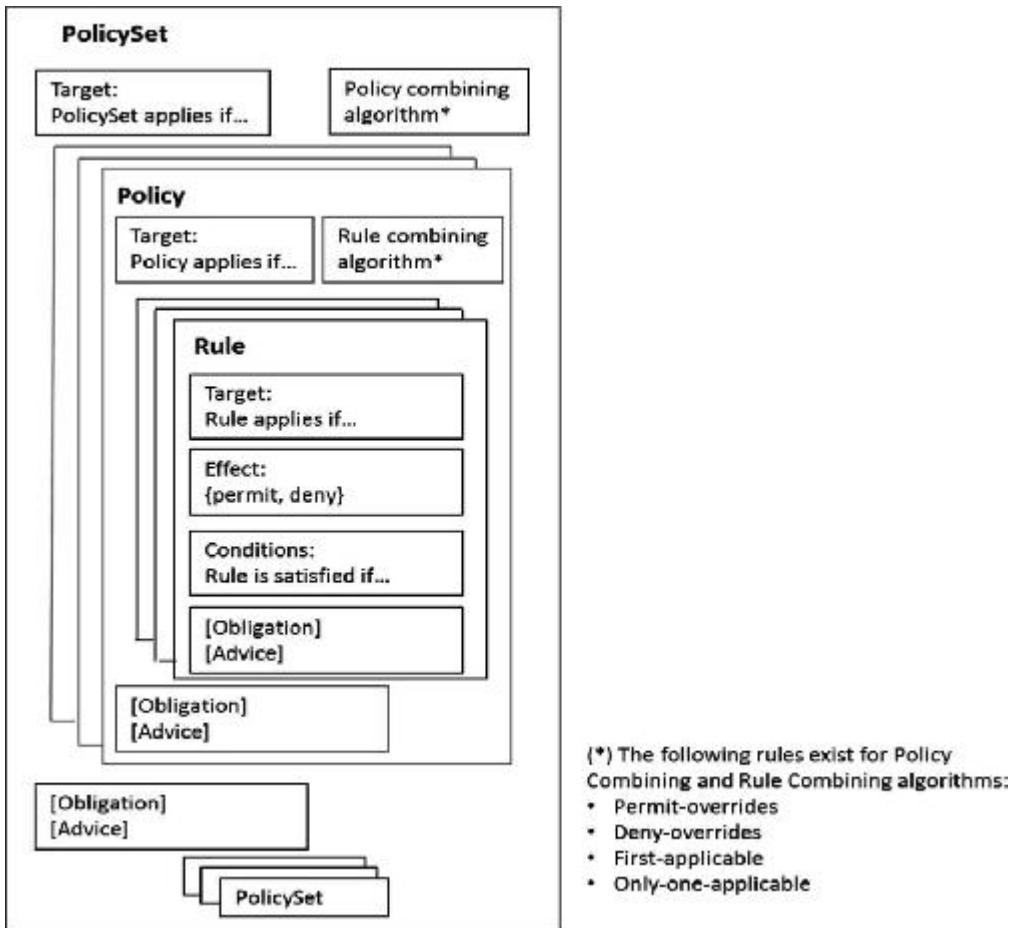


Figure 4.2 XACML policy components and elements.

A policy component comprises four elements: a target, set of rules, obligation expressions, advice expressions and a rule-combining algorithm-identifier. A policy may contain multiple rules, each of which may result in a different access control decision (the value for effect element). To reconcile these decisions, XACML uses a collection of conflict-resolution solutions called combining algorithms. A similar situation occurs in a policy set as well—each of the constituent policies may result in different decisions. Thus XACML has the provision to specify a combining algorithm for both a policy and a policy set. The combining algorithm specified in a policy is called a rule combining algorithm and the one specified in a policy set is called a policy combining algorithm.

The top most component in the XACML policy language model, the policy set may contain the following elements: a target, a set of policies, obligation expression, advice expression, a policy combining algorithm identifier, as well as a set of policy sets as elements. The semantics of the first four elements have already been covered in the discussions so far.

The inclusion of policy set element in a policy set means that XACML provides for nesting of policy sets in a policy repository.

4.3.2 XACML Context (Request and Response)

The second component of the XACML standard specifies the format used to convey an authorization request and the associated decision, i.e., the response. The format is called XACML context and is defined in a XML schema. The portion of the XML schema dealing with authorization request is called XACML request context, and the portion dealing with authorization response is called XACML response Context. The top-level elements in the XML schemata of both the request and response contexts are shown in [Figure 4.3](#). The designation of the name XACML context to the format is due to the fact that this XML-based format is an application environment-neutral canonical representation of the inputs to and outputs from the PDP (described in [Section 4.3.3](#)).

An instance of an XACML request context (or simply an XACML request context) consists of a set of attributes associated with requesting subjects, the resource acted upon, the action being performed and the environment ([Figure 4.3](#)). Each of these entities (i.e., subjects, resource, action and environment) can contain multiple attribute values.

An instance of an XACML response context (or simply an XACML response context) consists of one or more results ([Figure 4.3](#)). These results are obtained from the evaluation of the authorization decision request (encoded in XACML request context) against all applicable (relevant) policies. Each result consists of decision, optional status and an optional obligation as subelements. The decision can be one of the following values: permit, deny, not applicable (if no applicable policies or rules could be found), or indeterminate (if some error occurred during policy evaluation process). The status returns optional information to characterize the error. The last subelement, obligation, returns the actions to be performed before or after fulfilling the access request, if they are defined in any of the applicable policies or policy sets that were evaluated.

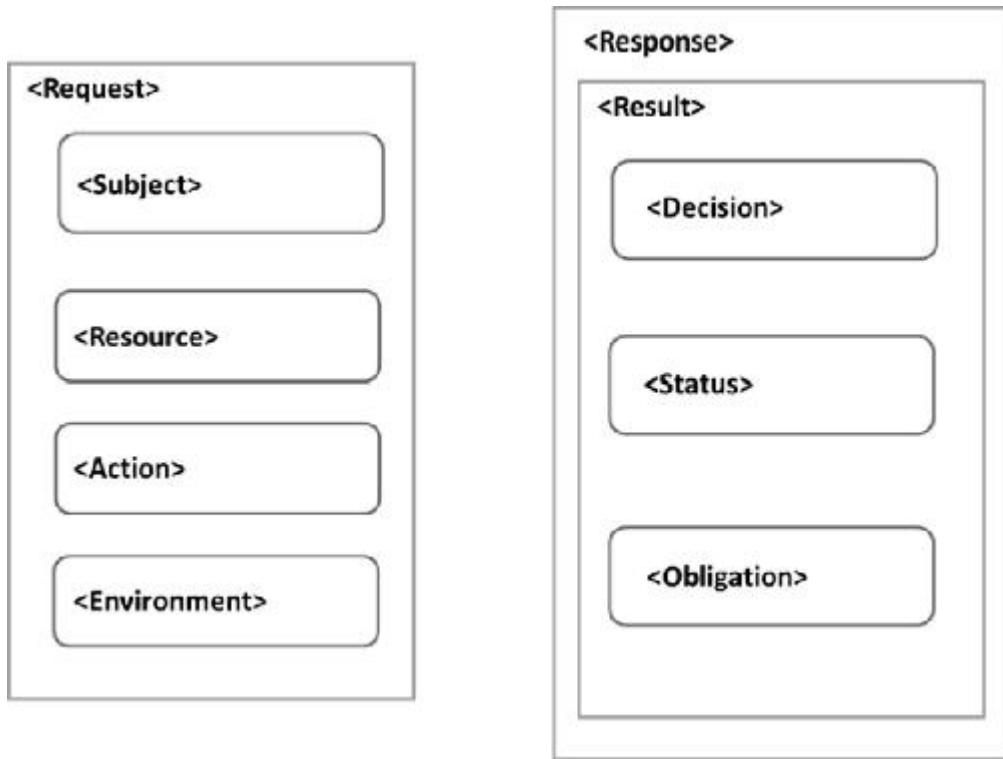


Figure 4.3 XACML request and response context.

4.3.3 XACML Framework (Data Flow Model)

The XACML standard provides a high-level management architecture (called the data flow model in the standard) with five key functional modules which collectively enable the authorization, decision-making, and policy/ attribute creation. The standard identifies the high-level function of each of the functional modules and data flows between each pair, again in terms of the type of data transmitted/ exchanged. It neither specifies the mechanics of implementing the functions in each of the modules nor does it specify the communication protocol for any of the data-flows. This component of the standard is not mandatory (non-normative).

Figure 4.4 is the schematic diagram of the XACML standard's data flow model. In the diagram the five key functional modules mentioned above are the following:

- Policy Enforcement Point (PEP);
- Context Handler (CH);
- Policy Decision Point (PDP);

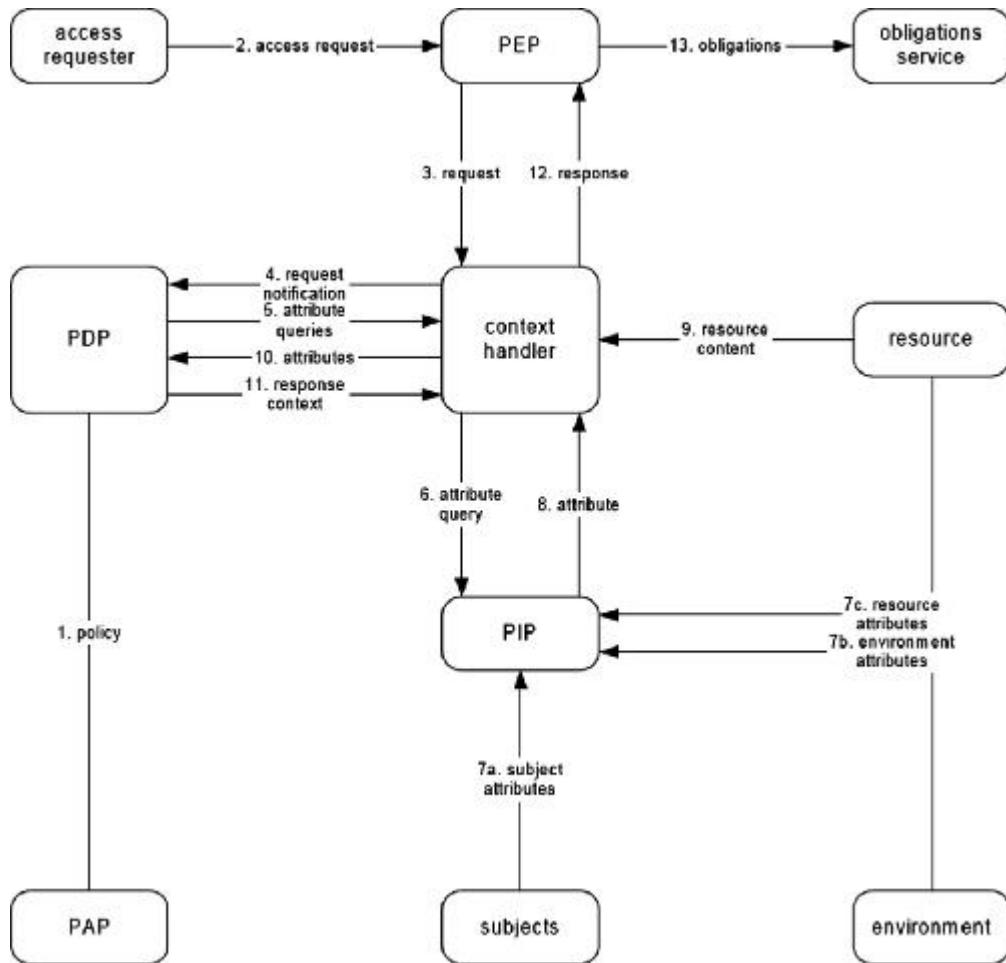


Figure 4.4 XACML data flow model.

- Policy Administration Point (PAP);
- Policy Information Point (PIP).

In addition, the diagram contains the following artifacts pertaining to the application environment for which access decision service is provided by the XACML framework. They are:

- Application environment entities (resource, subjects and environment);
- Application-specific functional modules (access requester and obligation service).

An overview of the data flows in the XACML data flow model is given below:

1. The PAP writes policies and policy sets and makes them available to PDP.
2. The access requester sends an access request to PEP.

3. The PEP sends the access request to the context handler in its native request format, optionally including attributes of the subjects, resource, action and environment.
4. The context handler constructs an XACML request context, optionally adds attributes, and sends it to PDP.
5. The PDP requests any additional subject, resource and environment attribute values from the context handler.
6. The context handler requests the attributes from PIP.
7. The PIP obtains the requested attributes from subjects, resource and environment.
8. The context handler forwards those attributes (and optionally the relevant resource content) to PDP.
9. The PDP requests the PAP (or the repository where PAP has stored the policies for PDP access) for policies matching the request's target.
10. The PAP (or the policy repository manager) returns the requested policies.
11. The PDP evaluates the related (applicable) policies and returns the authorization decision, encoded as an XACML response context to the context handler. It is thus the core of the access decision engine performing key operations such as parsing XACML access request, retrieving policies, obtaining additional attributes through PIP, evaluating policies and encoding the access decision response.
12. The context handler translates the XACML response context to the native response format of the PEP. The context handler returns the response to the PEP.
13. The PEP enforces the authorization decision (either allow or deny) and also fulfills the obligations (if any).

Based on the data flow described above, the function of each functional module in the XACML data-flow model can be summarized as follows.

1. The PEP enforces access control by making authorization decision requests to context handler (and in some environments directly to PDP) and enforcing the authorization decision that is returned in response to the request.

2. The context handler constructs the XACML request context from the native request format (received from PEP), obtains the attributes requested by PDP through PIP optionally adding the resource content to the XACML request context and also translates the XACML response context (received from PDP) to the native response format for transmitting to PEP.
3. The PDP retrieves the applicable policies from the policy repository (used by PAP to store the policies) evaluates the applicable policies and renders an authorization decision.
4. The PAP creates the security policies and stores them in an appropriate repository to be made available to PDP.
5. The PIP serves as the functional module for retrieving all attributes required for policy evaluation. It manages all the information (attributes and their associated values) related to subject, resource and environment.

In the XACML request and response context, as described so far, the access request and response messages respectively are encoded in XML syntax. However, using the REST profile for XACML, there are implementations of XACML framework with a PEP that can generate access requests and accept responses in either JSON or XML format with corresponding PDP implementations that can handle requests and responses in both formats. Further, the PDP implementations built using the REST profile for XACML expose a REST interface with HTTP as the communication protocol, as opposed to the SOAP protocol.

4.4 ABAC Deployment Using XACML

The deployment of ABAC using XACML is now illustrated with an example. The application is an electronic health record (EHR) application that provides access to electronic medical records of patients undergoing treatment in the wards of a hospital. The principal users (access subjects) accessing the medical records, based on their role attribute, are doctors and resident-interns. Another attribute associated with an access subject, besides his or her role, is the current ward to which he or she is assigned (ward-assignment). The access modes or the values for the action-id attribute are read and create (just two actions are considered for simplicity). Since the medical records contain information

about a patient, the value of the identifying attribute for the resource (resource-id) is patient-data. Please note that the collective name in the XACML policy language terminology for all entities participating in an access request instance is category (equivalent of ABAC model's element). The various attributes associated with each XACML category in our EHR application are summarized in [Table 4.2](#) below.

This is an example of an application environment where access restrictions (conditions for legitimate access) have to be governed by attributes associated with the subjects and resources. The alternate method of assigning access rights based on a particular subject and particular resource (e.g., patient-data record) is not scalable and flexible since it will require continuous permission re-assignments in the dynamic environments in which both access subjects and resources keep on changing. For example, the doctor on duty (access subject) in various wards will change depending upon on the shift time and day, and the patient-data record of the patient (resource) will be different based on the patients currently undergoing treatment in the ward during that shift/ day. In such an application environment, specifying access based on some fixed, invariant properties (attributes) associated with the subjects, resources and possibly the environment (e.g., the device from which patient-data records are accessed) is a natural, flexible, scalable approach. Hence ABAC model is a natural fit as an access control model in this application environment. One of the first steps in the deployment of ABAC is the formulation of access policies. The development of access policies based on attributes and specification of these policies in pseudo code (to facilitate coding in XACML syntax later on) are therefore the natural initial steps to be followed in the deployment sequence.

4.4.1 Access Policy Formulation and Encoding

The root element of the policy repository pertaining to our application is a policy set. The target of this policy set is any type of access to patient-data by any user. This policy set holds two access policies—policy 1 and policy 2. Policy 1 governs all required access (read and create) for all staff (doctors and resident-interns) under normal circumstances while policy 2 governs access under patient emergency situations.

Table 4.2
Attributes and Values for XACML Categories in EHR Application

| XACML Category | Attribute ID | Attribute Value(s) |
|----------------|-------------------|---------------------------|
| Access subject | role | {doctor, resident-intern} |
| Access subject | ward-assignment * | |
| Resource | resource-id | patient-data |
| Action | action-id | {read,create} |

* Values for this attribute are not listed since this attribute's value is only used in a rule condition to compare it with a value found in the resource content (e.g., ward-location value found in the patient-data record). Only those attribute values found in the target of a policy set, policy or rule are listed in this table.

Policy 1's target is all read or create accesses to medical records by a doctor or resident-intern and it includes two rules, rule 1 and rule 2. Hence, policy 1 is considered applicable whenever any subject submits a read or create access to patient-data. The explicit target for rule 1 under this policy is for situations where a subject with either the role of doctor or resident-intern issues an access request and by inheritance of the target from its parent policy (i.e., policy 1) covers either a read or create request to the patient-data resource. Rule 1 also specifies the condition under which read or create requests from doctors or resident-interns to patient-data can be allowed. In general, the condition can be expressed using the following three types of attribute comparison expressions (ATE).

- ATE 1—The value of an attribute from a category is compared to a given string (e.g., subject's role= "doctor").
- ATE-2—The attribute values belonging to two different attributes (belonging to the same category or different categories) are compared (e.g., subject's access-location attribute value {contained in} = resource's allowed-location attribute values.)
- ATE-3—The value of an attribute for a category is compared to the value retrieved from the resource's content through an XPath [6] expression: subject's ward-assignment attribute value is compared to patient's ward-location value in each of the patient-data records.

In our rule 1, the attribute comparison expression of type ATE-3 is used in the rule's condition. The condition is that a doctor or resident-intern is allowed access to a patient-data record provided the record's ward-location content is equal to the ward-assignment attribute value of the accessing subject. Rule 2 is meant to make some exceptions to the access permitted under rule 1. Specifically, it denies create access to all resident-interns. In other words, while rule 1 provides blanket read and create access to all doctors and resident-interns (under the conditions that they are accessing patient-data records of patients in their assigned wards), rule 2 explicitly restricts create access to resident-interns. Hence the logical way to combine the outcome of these two rules is to use the rule combining algorithm deny-overrides.

Policy 2's target is all doctor access to patient-data under patient emergency condition. This condition is captured in its constituent rule (i.e., rule 3). Specifically, the condition contains the Xpath expression for determining patient-status content from the patient-data record and if it equals the value critical, access to that patient-data record must be allowed. In order that this effect takes precedence over any other outcome, the encapsulating policy (policy 2) must override the effect of any other applicable policy in the parent policy set. Hence, the policy combining algorithm of permit-overrides is used.

Since access to resources of our EHR application (i.e., patient-data) is governed by more than one policy, the governing policies have to be encapsulated within a policy set. The entire set of the access policy elements that will collectively form the contents for the XACML policy repository for our EHR application are shown in [Table 4.3](#) below.

With the above information on access policy elements and their contents, we are now ready to encode the above policies and rules in XACML. Before we do that, it will be a good idea to develop a pseudo code containing the major elements and sub-elements that will form part of the XACML policy repository document. The top most (or the root) element of this document has to be the policy set since there are multiple policy elements. The major elements and subelements of the encapsulated policies, policy 1 and policy 2, together with the policy set element are shown in pseudo code below:

```

<Policy Set EHR Access <policy-combining-algorithm>="permit-
overrides">
  <Target>
    /* :Attribute-Category      :Attribute ID      :Attribute
Value */
    :resource      :resource id      :patient-data
  </Target>

```

Table 4.3
XACML Policy Repository for EHR Application

| Access Policy Element | Constituent Elements/Target/Conditions | Combining Algorithm |
|-----------------------|--|---|
| Policy set | Policy 1—(target)—access subjects (with roles doctors and resident-interns) can access the resource (patient-data) with actions (read or create) Policy 2—(target)—access subject (with role doctor) accessing the resource (patient-data)—meant for emergency access | Policy combining algorithm—permit-overrides |
| Policy 1 | Rule 1—(condition)—the ward-assignment of the doctor or resident-intern is the same as ward-location content in patient-data record. Rule 2—(condition)—create access for resident-intern is explicitly denied | Rule combining algorithm—deny-overrides |
| Policy 2 | Rule 3—(condition)—the patient status is critical (patient-status content in patient-data record) | N/A |

```

<Policy PolicyId = "Policy 1" rule-combining-
algorithm="deny overrides">
    // Doctor & Resident-Intern Access to Patient-Data
Records //
<Target>
/* :Attribute-Category :Attribute ID :Attribute Value */
    :action      :action-id      :read
    :action      :action-id      :create
</Target>

<Rule RuleId = "Rule 1" Effect="Permit">
<Target>
/* :Attribute-Category :Attribute ID :Attribute Value */
    :access-subject   :role       :doctor
    :access-subject   :role       :resident intern
</Target>
<Condition>
Function: string-equal
Function: string-one-and-only
/* :Attribute-Category :Attribute ID */
    :access-subject   :ward-assignment
Function: string-one-and-only
    XPath Expression for getting the value of
    "ward-location" content from "patient-data"
    records
</Condition>
</Rule>
<Rule RuleId = "Rule 2" Effect="Deny">
<Target>
/* :Attribute Category :Attribute ID :Attribute Value */
    :access-subject   :role       :resident-intern
    :action          :action-id      :create
</Target>
</Rule>
</Policy 1>
<Policy PolicyId = "Policy 2">
    // Doctor Access to Medical Records during patient criti-
cal situations/
<Target>
/* :Attribute-Category :Attribute ID :Attribute Value */
    :action      :action-id      :read
    :action      :action-id      :create
</Target>
<Rule RuleId = "Rule 3" Effect="Permit">
<Condition>
Function: string-equal
Function: string-one-and-only
/* :Attribute-Category :Attribute ID :Attribute Value */

```

```

:access-subject      :role      :doctor
    Function: string-one-and-only
XPath Expression for getting the "patient-status" content
from "patient-data" records and checking whether its value
    - "Critical"
    </Condition>
</Rule>
<obligationExpression>
    After enforcing the access decision, the PEP passes
the following
        Obligation to be executed by an Obligation service
- Send an email
            to the primary-physician of the patient notifying
that
            his/her patient's record was accessed by the iden-
tified doctor
            (identified by the subject-id of accessing subject.)
        </ObligationExpression>
    </Policy>
</Policy Set>

```

4.4.2 Request/Response Formulation

XACML specifies a format for conveying an access request and the consequent decision response. The format is referred to as XACML context. The request and response formats represent a standard interface between a PDP, with standard behavior, and a standard PEP that issues requests and deals with the response. If the PEP does not generate requests in XACML context, a context handler is required between the PEP and PDP. The handler converts application environment-specific requests (coming from PEP) to XACML request context before submitting them to PDP, and also converts XACML response context received from PDP to application environment-specific responses before forwarding them to PEP.

A request context consists of one or more attributes associated with categories: subject, resource, action, and environment. For example, if a doctor with a login ID of “jsmith” attempts to read the patient data of the patient by name “David Brown”, the XACML Request Context may initially carry the values “jsmith” and “doctor” for the subject-id and role attributes of the subject (access-subject), value read for action’s action-id attribute, the value patient-data for resource’s resource-id and the Xpath expression for getting the record in the patient-data where patient-name is David Brown. The source for the value of the role attribute of the subject can be the identity token generated as part of the successful login

process. An initial XACML request context carrying the attributes mentioned above is given below:

```
<Request
    /* Subject Attributes */
    <Attributes
        Category="subject-category:access-subject">
            <Attribute AttributeId="subject:subject-id">
                <AttributeValue>jsmith</AttributeValue>
            </Attribute>
            <Attribute AttributeId="attribute:role">
                <AttributeValue>doctor</AttributeValue>
            </Attribute>
        </Attributes>
    /* Action Attribute */
    <Attributes
        Category="attribute-category:action">
            <Attribute AttributeId="action:action-id">
                <AttributeValue>read</AttributeValue>
            </Attribute>
        </Attributes>
    /* Resource Attributes */
    <Attributes
        Category="attribute-category:resource">
            <Attribute AttributeId="resource:resource-id">
                <AttributeValue>patient-data</AttributeValue>
            </Attribute>
            <Attribute AttributeId="Content-Selector">
                Path="EHR:patient-data/patient/patient-name/
text()"
                <AttributeValue>David Brown</AttributeValue>
            </Attribute>
        </Attributes>
    </Request>
```

Subsequently, based on the attributes found in the logical expressions of the conditions found in the rules of the applicable policies, the PDP may either request the PIP directly or through the context handler for additional attribute values. In addition, it may request the relevant resource content as well. In our example, the ward-assignment of the subject may be an additional attribute that will be requested. Additional resource contents requested may include the ward-location and patient-status values in the patient-data record.

4.4.3 Policy Evaluation and Access Decision

Let us look at a possible policy evaluation logic in our scenario. The policy evaluation process can be broadly looked upon as having two

phases—policy finder phase and attribute retrieval and condition matching phase. In the policy finder phase, the attributes in the XACML request context are matched with the attributes in the target of a policy (or policy set). In our scenario, based on matching the attributes in the request context (role attribute of subject, resource-id attribute of resource, action-id attribute of action), it is clear that both policy 1 and policy 2 are applicable. Finding that there are rules under these policies with conditions and additional attributes (rule 1 under policy 1 and rule 3 under policy 2), the policy evaluation logic enters the attribute retrieval and condition matching phase. It is at this point that the PDP may request additional attributes and resource content. In our scenario, the ward-assignment attribute value for the requesting subject and ward-location and patient-status values in the patient-data record for the patient with patient-name David Brown may be obtained by PDP.

Let us now briefly look at the actual policy evaluation logic in our scenario. Taking up policy 1 (the first applicable policy), we find that rule 1 is applicable since the role attribute value (doctor) in the access request matches with that of the target of this rule. The condition in this rule is evaluated by comparing the ward-assignment attribute value of the accessing subject with the ward-location value in the accessed patient-data record. If the ward-assignment value for accessing subject (jsmith) is cardiology and the ward-location value in the David Brown patient-data record is also cardiology then the condition under rule 1 is satisfied. Rule 2 under policy 1 is not applicable since the accessing subject does not have the role resident-intern. Since the rule combining algorithm for policy 1 is deny-overrides and the only applicable rule (rule 1) has resulted in the permit effect, the overall effect of evaluating the access request against policy 1 is permit.

Moving over to policy 2, let us assume that the only rule (i.e., rule 3) turns out to be not applicable as the patient-status value in the David Brown patient-data record is not critical. Hence policy 2 evaluation does not result in any effect. Since the combined effect of evaluating policies policy 1 and policy 2 is permit-overrides, the permit effect produced by evaluating policy 1 stands and hence the overall access decision result is permit. An XACML response context in our scenario will therefore have the value permit under the decision subelement under result element as shown below:

```

<Response>
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="status:ok" />
    </Status>
  </Result>
</Response>

```

4.5 Implementation of XACML Framework

The first task in ABAC deployment using XACML is the implementation of the functional modules of the XACML framework (PEP, context handler, PDP, PIP and PAP) in a higher level language such as Java. Out of these modules, typically the PAP, and in some instances the PDP, will be offered as off the shelf modules in a commercial ABAC product. An enterprise wanting to deploy ABAC has to install this product and integrate it with all application environments by custom building other modules such as PEP and PIP. The context handler will rarely be found as a separate module since in most instances the PEP itself will have the functionality to generate an XACML request context and interpret the XACML response context. The integration task may involve the following main activities:

- Establishment of networking and messaging protocol parameters between any pair of functional modules (PEP to PDP, PDP to PIP, etc.). Please note that the XACML data flow model in the standard does not specify the communication mechanisms between the functional modules.
- Choose deployment architecture based on product features and the needs of the application environment in the enterprise. In many instances, the PEP is always tightly integrated with application environment. However, the PDP can be configured either as a common functional module or service that will provide access decision support for multiple applications, or it can be tightly integrated with PEP and customized for a specific business application. The PIP also needs to be customized for an enterprise since the sources from which various categories of

attributes are obtained vary widely across organizations. Finally, the customized PIP needs to be integrated with PDP.

- The main on-going task in XACML framework deployment is the encoding/updating of policies and rules using PAP and storing them either in the policy repository or in some other artifacts that are accessible to PDP module(s). The content of the XACML policy repository, together with those of all attribute stores, defines the ABAC-XACML model instance, while the interfaces and messaging protocols of various functional modules constitute the overall ABAC-XACML deployment architecture.

The expressiveness of a ABAC-XACML model instance and the resultant granularity of access privileges it can help to generate depends greatly on the attribute data types the deployment supports, as well as the functions available for retrieval and usage of these attribute values in the various logical conditions. Hence attribute support and management forms a foundation for a feature-rich ABAC-XACML deployment and is discussed in the next section.

4.5.1 Attribute Support and Management

Attributes are named properties of various categories of an XACML language policy model. Since subjects, resource, action and environment are these categories, the total set of attributes in a ABAC-XACML model can be classified as:

- Subject attributes;
- Resource attributes;
- Action attributes; and
- Environmental attributes.

Attributes are specified using the following:

- Category designation (subject, resource, action, or environment);
- Attribute ID (unique identifier for the attribute—e.g., subject-id of access-subject);
- Attribute type (e.g., string, integer, etc);
- (Optional) issuer identifier (e.g., HR-Admin);
- (Optional) issue date and time (e.g., 25-Oct-2016).

Subject attributes are usually provided by enterprise directory services using protocols such as LDAP. The environmental attributes are provided by enterprise authentication services and are carried in the identity tokens (e.g., SAML 2.0 token). The resource attributes are obtained from application/ database schemata. The action attributes are operations specific to an application such as send for email application and write for a file processing application.

XACML does not impose any requirements on attribute names. Hence the attribute name space can be arbitrary. As far as attribute types are concerned, XACML supports many common data types such as strings, integers, email addresses, dates, and URI. In addition, common implementation languages for XACML enable addition of new types. For example, in a JAVA implementation, where attribute types are subclasses of the abstract class attribute-value, a new attribute type can be created by creating a new class that extends the attributevalue class, supplying the identifier for the new attribute type and by implementing all the necessary methods for the newly created java class (and hence for the new attribute type).

In XACML, wherever attributes are used, their types are also specified. Hence when a new attribute type is defined, it is easy to include values associated with these new attribute types in both policies as well as in all XACML contexts (requests and responses). However, including support for these newly defined attribute types, in order for them to be used for target matching and condition evaluation functions in PDP, does involve some extra coding work.

XACML 3.0 provides some predefined attributes associated with each category. The attributes associated with each category, and a brief description of each, are given in [Tables 4.4](#) and [4.5](#) below.

The attributes associated with the subject, besides the subject-id attribute are given in [Table 4.5](#) below.

4.5.2 Delegation

The XACML policies discussed thus far have pertained to access policies that are created and may be modified by a single authority. This single authority (e.g., a centralized administrator) is trusted and hence these access policies carry no designation of an issuer and are considered trusted. Thus, based on the semantics of trust, the type or class to which

these policies belong can then be called trusted access policies. But in the absence of any other type of policy, every policy in the XACML policy repository is simply called an access policy. The advantage of a trusted policy is that it can be directly used by the PDP in rendering a decision.

Table 4.4
Predefined Attributes in XACML

| Category | Attribute ID | Description |
|-----------------|---------------------|---|
| Subject* | subject-id | Unique identifier for the subject* |
| Resource | resource-id | Unique identifier for the resource |
| | target-namespace | Namespace of the top element(s) of the resource content |
| Action | action-id | Unique identifier for the action |
| | implied-action | Value of action-id attribute when the action is implicit |
| | action-namespace | Namespace in which the action-id attribute is defined |
| Environment | current-time | Time at which the request context was created |
| | current-date | Date at which the request context was created |
| | current-datetime | Combination of the attribute values of the above two attributes |

* There are many more attributes associated with the subject. For the sake of not cluttering this table they are provided in a separate table (Table 4.5) below. Please note that Table 4.5 contains all attributes associated with the subject except the subject-id attribute already identified in Table 4.4.

Table 4.5
Attributes Associated with Subject

| Attribute ID | Description |
|---------------------------|--|
| subject-id-qualifier | Security domain of the subject identified by the administrator and policy that manages the namespace in which the subject-id is administered |
| key-info | Public (cryptographic) key used to confirm the subject's identity |
| authentication-time | Time at which the subject was authenticated |
| authentication-method | Method used to authenticate the subject |
| request-time | The time at which the subject initiated the access request, according to the PEP |
| session-start-time | The time at which the subject's current session began, according to the PEP |
| authn-locality:ip-address | Location where authentication credentials were activated expressed as an IP address |
| authn-locality:dns-name | Location where authentication credentials were activated expressed as a DNS name |

In recognition of the need to create policies through delegation from a centralized administrator or single authority to a set of subordinate administrators (or delegated authorities), XACML standardization effort includes specification of a delegation profile [7]. This profile has introduced a new type of policy called administrative policies. A feature enabled by this new type of policy is the ability to designate subordinate authorities either for: (a) creating access policies or (b) creating one or more administrative policies (thus extending the delegation chain). This designated authority is called a delegate and the domain over which the delegate has authority is called a situation. Collectively these capabilities result in the specification of a set of delegated policies and thus delegation chains—consisting of one or more administrative policies and terminating with an access policy.

The two new concepts of delegate and situation introduced by administrative policies are encapsulated within its target. Thus the target in an administrative policy is different in content and semantics from that

found in an access policy. The delegate is an attribute category of the same type as a subject, representing the entity(s) that has been given the authority to create either an access policy (thus completing the delegation chain) or an administrative policy (thus adding to the delegation chain). A situation provides the scope for the delegated authority by defining a privilege domain. This privilege domain is expressed in terms of a combination of subject, resource, and action attributes. If the delegate creates a policy granting access rights for resources within the scope of that privilege domain, then the type designation for such a policy is untrusted access policy. An untrusted access policy can be distinguished from an access policy (found in policy repositories without delegation) in that the former should always have an issuer. On the other hand, if the delegate decides to exercise the right of further delegation (instead of exercising the right to create access policies) within the scope of that privilege domain, the resulting administrative policy he or she creates will be of a type called untrusted administrative policy.

Thus, in an XACML policy repository that supports delegation, you can have the following four subtypes of policies:

- (Trusted) access policies;
- Untrusted access policies;
- Trusted administrative policies;
- Untrusted administrative policies.

The root of trust, or the starting point for creating a set of delegated policies (and hence a delegation chain), is a trusted administrative policy. Trusted administrative policies are created under the same authority used to create (trusted) access policies. Thus a trusted administrative policy does not have an issuer tag since it is always created by a trusted centralized administrator.

Recall that the purpose of administrative policies is the ability to delegate the rights to create access policies to multiple subordinate authorities either directly or indirectly through a series of further delegations. Thus a trusted administrative policy gives the delegate the authority to create untrusted administrative policies or untrusted access policies. Hence a delegation chain has to naturally start from a trusted administrative policy and end with an untrusted access policy. Consequently, a delegation chain can optionally include one or more

intermediate untrusted administrative policies. The situation for a newly created untrusted administrative policy or untrusted access policy is a subset (the same or narrower in scope) of that specified in the trusted administrative policy. In addition, an untrusted administrative policy or untrusted access policy includes an issuer tag with a value that is the same as that of the delegate in the administrative policy (trusted or untrusted) under which it was created. Both of these policies have at least one rule with a permit or deny effect.

4.5.2.1 Delegation Chain—An Example

An example of a delegation chain (going from bottom to top) is shown in [Figure 4.5](#). As expected, the starting node of this delegation chain is a type of administrative policy called a trusted administrative policy. Based on the description, this should have a delegate (the subordinate administrator) and a situation (the privilege domain over which the subordinate administrator can grant access (or in turn delegate further)) in its target. In our example, the trusted administrative policy designates Smith as the subordinate administrator and situation 1 as the privilege domain over which Smith can grant access rights to resources or do further delegation.

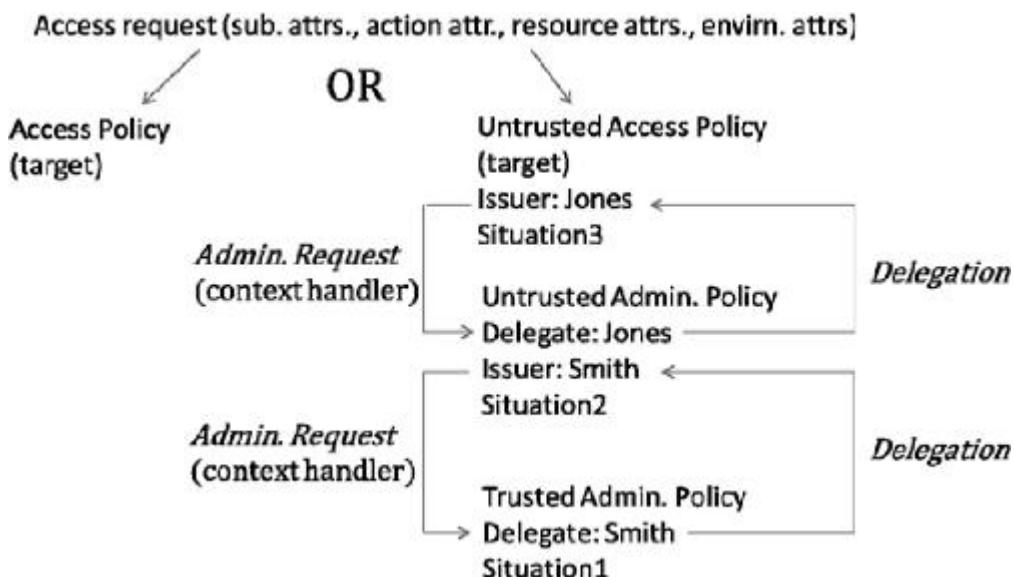


Figure 4.5 An example of a delegation chain.

Recall that an untrusted administrative policy should also have a delegate (the designated subordinate) and an associated privilege domain

(specified in situation). In addition, an untrusted administrative policy should have an issuer tag as well. The value of this issuer tag must match the value of delegate tag in some administrative policy. Then only the authority under which this untrusted administrative policy was created can be established. In our [Figure 4.5](#), there is an untrusted administrative policy issued by Smith (under the authority of the trusted administrative policy discussed above) designating Jones as the delegate (subordinate administrator) with the associated privilege domain being situation 2. The scope of this privilege domain situation 2 must be a subset of the privilege domain situation 1 that was defined in its authority-granting trusted administrative policy discussed above.

Just like the subordinate administrator Smith, his delegate Jones can exercise the right of further delegation or grant access rights to resources specified within her privilege domain. If Jones exercises the right of granting access rights to resources within her privilege domain (unlike Smith), the resulting access policy that she will create will be an untrusted access policy. In [Figure 4.5](#), there is an untrusted access policy issued by Jones (under the authority granted to her by Smith) with the scope of privilege domain named situation 3. This scope must be a subset of the privilege domain situation 2 that was delegated to her by Smith.

4.5.2.2 Access Request Processing in Delegation Chains

In the absence of any delegation chain, all access policies in the system are (trusted) access policies. In these situations, if the PDP is able to match the attributes in the request with the target of an access policy, then that access policy is treated as an applicable policy and automatically included as a participating policy in the relevant policy-combining algorithm (since the policy is trusted). In the presence of one or more delegation chains, the PDP may discover that the matching target (for a request) is found in an untrusted access policy. Since it is designated as an untrusted access policy, it can be considered for inclusion in the policy-combining algorithm by the PDP only if the authority of the policy issuer is first verified (recall that an untrusted access policy always has an issuer tag). Authority is verified by finding a delegation chain that leads to a trusted administrative policy. This chain may involve zero or more untrusted administrative policies in the path to

a trusted administrative policy. When policies are considered as nodes of a graph, the process of finding a delegation chain translates to finding/constructing a path in that graph from a node representing an untrusted access policy to a node representing a trusted administrative policy. To construct each edge of the graph, the PDP (using the XACML context handler) formulates an administrative request (as opposed to an access request) as shown in [Figure 4.5](#).

An administrative request has the same structure as an access request except that in addition to attribute categories—access-subject, resource, and action—it also uses two additional attribute categories, delegate and decision-info. If a policy, Px, happens to be one of the applicable (matched) untrusted access policies, the administrative request is generated using policy Px to construct an edge to policy Py using the following:

- Convert all attributes (and attribute values) used in the original access request to attributes of category delegated.
- Include the value under the issuer tag of Px as the value for the subject-id attribute of the delegate attribute category.
- Include the effect of evaluating policy Px as attribute value (permit, deny, etc.) for the decision attribute of the decision-info attribute category.

The administrative request constructed using the above attributes is evaluated against the target for policy Py. If the result of the evaluation is permit, an edge is constructed between policies Px and Py. The overall logic involved is to verify the authority for issuance of policy Px. For this there should exist a policy with its delegate value set to the policy issuer of Px with either its situation covering the attribute values in the target of policy Px. If that policy is Py, then it means policy Px has been issued under the delegate authority set in policy Py. The edge construction then proceeds from policy Py until an edge to a trusted administrative policy is found.

The process of selecting applicable policies for inclusion in the combining algorithm in delegation chains is illustrated in [Figure 4.6](#). Assume that for an access request, the matching targets are found in three untrusted access policies, P31, P32, and P33. Each of these policies can become an applicable policy (and hence can be included in an associated combining algorithm) if a path can be constructed (through an

edge or a series of edges) from each of them to a trusted administrative policy (thus verifying the authority under which each was issued). In Figure 4.6, such paths can be constructed/established from policies P31 and P32. From policy P31, this path goes through an untrusted administrative policy, P21 to a trusted administrative policy, P11. Similarly, originating from policy P32, there is a path that goes through an untrusted administrative policy P22 to a trusted administrative policy P12. However, no such path exists from the third untrusted access policy P33. Hence, only policies P31 and P32 will be used in the combining algorithm for evaluating the final access decision, and policy P33 will be discarded since its authority could not be verified.

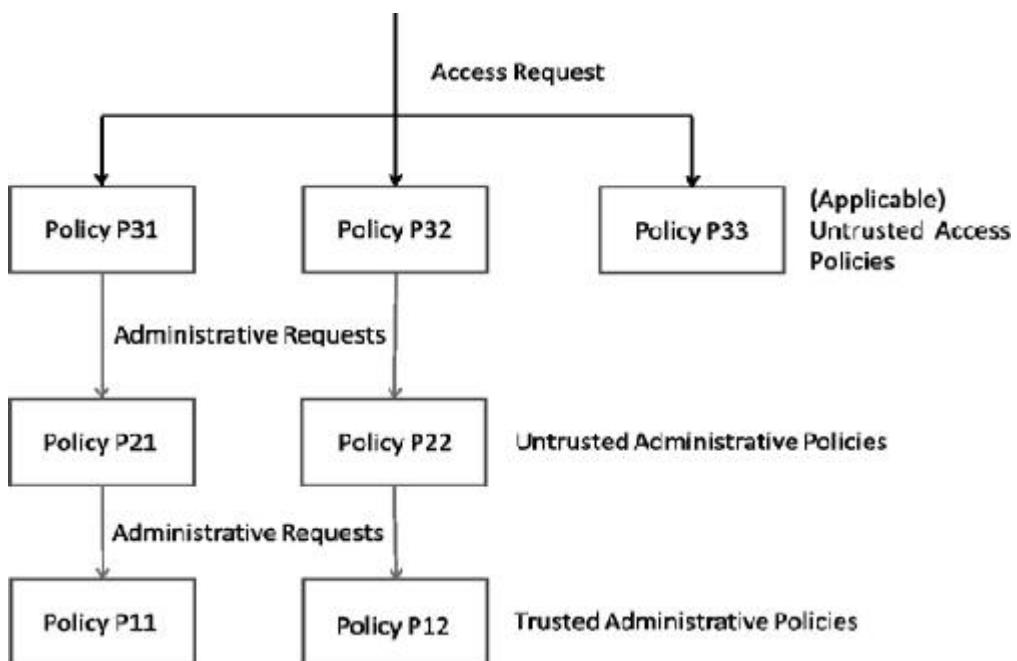


Figure 4.6 Utilizing delegation chains for policy evaluation.

Below is a more concrete example that illustrates the use of delegation chains to select applicable policies that are used in combining algorithms for arriving at final access decisions. The example gives a policy set that consists of four policies:

- *Policy P1*: A trusted administrative policy that gives John (the delegate) the authority to create policies for a situation involving reading of medical records to any user who has the role of doctor.
- *Policy P2*: An untrusted administration policy that is issued by John, under the authority of P1, to give Jessica (the delegate) the authority to create policies for a situation involving reading of

medical records to any user who has the role of doctor. Because of the matching of delegate of P1 to policy issuer of P2 and the fact that the situations in both policies P1 and P2 are the same, it is obvious that the authority to issue policy P2 has come from policy P1. Thus P1 and P2 form a delegation chain.

- *Policy P3*: An untrusted access policy that is issued by Jeff to give Carol the capability to read medical records.
- *Policy P4*: An untrusted access policy that is issued by Jessica to give Carol the ability to read medical records. Because of the matching of delegate of P2 to policy issuer of P4 and the fact that the situations in both policies P2 and P4 are the same, it is obvious that the authority to issue policy P4 has come from policy P2. Thus P2 and P4 form a delegation chain.

The four policies described above are given in the form of pseudocode below:

```

<Policy Set>
    <Policy P1> /* Trusted Administrative Policy */
        <Target> /*:Attribute-Category :Attribute ID :Attribute Value*/
            :access-subject      :role          :doctor
            :resource             :resource-id   :medical-records
            :action               :action-id     :read
            :delegate             :subject-id   :john
        </target>
        <Rule R1>
            Effect: PERMIT
        </Rule R1>
    </Policy P1>

    <Policy P2> /* Untrusted Administrative Policy */
        <Policy Issuer> john </Policy Issuer>
        <Target> /*:Attribute-Category :Attribute ID :Attribute Value*/
            :access-subject      :role          :doctor
            :resource             :resource-id   :medical-records
            :action               :action-id     :read
            :delegate             :subject-id   :jessica
        </target>
        <Rule R2>
            Effect: PERMIT

```

```

        </Rule R2>
    </Policy P2>

    <Policy P3> /* Untrusted Access Policy */
        <Policy Issuer> Jeff </Policy Issuer>
        <Target> /*:Attribute-Category :Attribute ID :Attribute Value */
            :access-subject :subject-id :carol
            :resource       :resource-id :medical-records
            :action         :action-id  :read
        </Target>
        <Rule R3>
            Effect: PERMIT
        </Rule R3>
    </Policy P3>

    <Policy P4> /* Untrusted Access Policy */
        <Policy Issuer> Jessica </Policy Issuer>
        <Target> /*:Attribute-Category :Attribute ID :Attribute Value */
            :access-subject :subject-id :carol
            :resource       :resource-id :medical-records
            :action         :action-id  :read
        </Target>
        <Rule R4>
            Effect: PERMIT
        </Rule R4>
    </Policy P4>
<Policy Set>

```

By matching the situation and delegate in one policy to the target and policy issuer in another, one can see that P1, P2, and P4 form a delegation chain. P3 is not part of any delegation chain. Given the above delegation structure, let us now look at the way the following access request REQ1 will be resolved.

```

<Request REQ1>
    <Attributes>
        /* :Attribute-Category : Attribute ID : Attribute
Value */
        :access-subject :subject-id :carol
        :access-subject :role   :doctor
        :resource      :resource-id :medical-records
        :action         :action-id  :read
    </Attributes>
</Request REQ1>

```

By matching the attributes (and values) in the request REQ1 with the attributes (and values) in the target of the policies in the policy set, one finds that only policies P3 and P4 (which are both access policies) are candidates for applicable policies as opposed to policies P1 and P2

which are administrative policies that contain delegated attributes. Since both policies P3 and P4 are untrusted access policies, their respective authorities have to be verified by making administrative requests. Since policy P3 is not part of any delegation chain, its authority cannot be verified. However, the authority for policy P4 can be established by using the delegation chain P1, P2, P4.

The same PAP interface that is used to create access policies can be used to create the additional policies needed for supporting delegation—untrusted access policies, trusted administrative policies, and untrusted administrative policies. This requires at least two classes of policy administrators. The members of the first class are system administrators authorized to create (trusted) access policies. The second class is made up of delegated administrators authorized to create untrusted administrative policies or untrusted access policies conforming to the situation or a subset of the situation authorized in any trusted administrative policy currently in the policy repository.

4.6 Review and Analysis

Review of the content of this chapter would have now convinced the reader that ABAC deployment using XACML is realized by implementing the functional modules of the XACML framework (Section 4.2.3) with XACML policy language model (Section 4.2.1) and XACML context (Section 4.2.2) providing the syntax for encoding access policies and making access requests/ obtaining access decisions respectively. Since access decision and policy encoding are the two primary functions in an ABAC-XACML deployment architecture, and since the PDP and PAP are the functional modules providing these functions, the focus of many open source XACML product offerings has been the implementation of these two modules. Majority of these implementations have been written in Java programming language. Some open source XACML offerings are: AT&T XACML [8], WSO2 Balana [9] and Enterprise Java XACML [10].

XACML enables expression of attribute-based access control policies at a fine level of privilege (access rights) granularity. In spite of this capability, the ability to enforce the principle of least privilege or in some instances the basic access to needed resources will be very much hampered if the request context is not formulated with the right set of

attributes and attribute values. For example, if in our EHR patient-data access example, the initial request context for an access request consists of only the value for the role attribute of the access subject. Subsequently when PDP discovers that it needs the ward-assignment attribute value of the accessing subject in order to evaluate rule 1 of policy 1 and if the PIP is not able to supply this attribute, then access will fail (with the access decision response of indeterminate) for all access request instances except for access to patient records with patient-status value critical. This is due to the fact that there is no rule that will produce a permit effect without comparison of these attribute values. Similarly, when there is a type mismatch between attribute values used in a condition, then the resultant decision response will also be indeterminate resulting in access denial. A situation in the context of our example is when the ward-assignment for the doctor (subject) is specified in terms of a number (e.g., 123) while the ward-location in the patient-data record is specified as a string (e.g., cardiology). The above examples show the importance of proper encoding the XACML request context, the sufficiency of attributes in the various attribute stores as well as the proper integration of PIP to all relevant attribute stores to retrieve the attributes needed for evaluating all applicable rule conditions.

The advantages of XACML in realizing the goals of an ABAC model and associated policies can be summarized as follows:

- The arbitrary name space for attributes together with extensibility of data types associated with attributes enables definition of arbitrary attributes which in turn enables specification of expressive policies.
- The extensibility of functions used in attribute comparisons enables definition of sophisticated functions which in turn enables specification of granular policies.

The constraints of an ABAC-XACML deployment, either due to the inherent features in the XACML policy language or XACML framework are:

1. The standard does not specify the means for obtaining various categories of attributes (subject, environment) securely, the methods to verify their trustworthiness as well as the mechanics of extracting them from multiple sources. The approach to

ensure the integrity and trustworthiness of attributes are addressed in [Chapter 7](#) of this book.

2. Performance and implementation limitations may limit full externalization of authorization services for some application environments and hence the PEP, the obligation service and in some instances the PDP may become tightly coupled to the application.
3. The attribute stores together with the PIP functionality to retrieve attributes from these stores play a key role in getting the right access decision response. This tight dependency between the policy content and some of these functional modules makes the overall implementation of XACML framework a complex process when these constituent modules are under different security domains as in federated application systems.
4. The expressive and granular policies make integrity checking of policy content (e.g., policy conflicts, redundancies) a complex process especially in situations where there are nested policy sets and multiple rule conditions.
5. Generation of information for policy review which provides (a) a list of all access rights for a subject (called capability list) or (b) a list of subject authorizations for a particular resource (e.g., ACL) is also an inherently complex process as it involves complete evaluation of all rules/ policies in the repository. This is a common problem with any ABAC model where access rights are captured using logical expressions instead of flat relations.

Readers interested in specification and capabilities of other access policy specification languages for ABAC besides XACML can refer to the comprehensive listing provided by World Wide Web Consortium (W3C) in [\[11\]](#).

References

- [1] OASIS, *eXtensible Access Control Markup Language (XACML)* TC, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.
- [2] OASIS Standard, *eXtensible Access Control Markup Language (XACML) Version 3.0*, January 22, 2013, <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>.

- [3] Imperial College London, *Ponder2 Wiki*, <http://www.ponder2.net/cgi-bin/moin.cgi/PonderTalk>, February 8, 2008.
- [4] PERMIS, <http://sec.cs.kent.ac.uk/permis/index.shtml>, July 20, 2011.
- [5] NIST Special Publication 800-162, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, January 2014, <http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf>.
- [6] Robie, J., M. Dyck, J. Spiegel, (eds.), *XML Path Language (XPath) 3.1*, W3C Candidate Recommendation, December, 17, 2015, <https://www.w3.org/TR/xpath-31/>.
- [7] XACML v3.0 Delegation and Administration Profile Version 1.0, Committee Specification Draft 04, Nov 13, 2014, <http://docs.oasis-open.org/xacml/3.0/administration/v1.0/csd04/xacml-3.0-administration-v1.0-csd04.pdf>
- [8] AT&T XACML 3.0 Implementation, <https://github.com/att/XACML>, April 13, 2015. See also <http://about.att.com/innovationblog/41315apolicyengine>.
- [9] WSO2, *Balana, The Open source XACML implementation*, <https://svn.wso2.org/repos/wso2/trunk/commons/balana/>, <http://xacmlinfo.org/category/xacml-3-0/>, <http://xacmlinfo.org/category/xacml/>, and <http://wso2.com/products/identity-server/>
- [10] Google Code Archive, *Enterprise Java XACML*, <https://code.google.com/archive/p/enterprise-java-xacml/>, February, 2012.
- [11] W3C, *PolicyLangReview*, <https://www.w3.org/Policy/pling/wiki/PolicyLangReview>, May 20, 2009.

Appendix 4.A

```
<PolicySet
    PolicySetId="EHR Patient Data Access"
    PolicyCombiningAlgId = "permit-overrides">
    <Description>
        The policy set is applicable for EHR Patient Data
access
    </Description>
    <Target>
        <AnyOf>
            <AllOf>
                <Match
                    MatchId="string-equal">
                    <AttributeValue>patient-data</AttributeValue>
                    <AttributeDesignator
                        Category="resource" AttributeId="resource-id"/>
                </Match>
            </AllOf>
        </AnyOf>
    </Target>

    <Policy
        PolicyId="Policy 1"
        RuleCombiningAlgId = "deny-overrides">
        <Description>
            Policy 1 is applicable for all Read or Create ac-
cess to Patient data.
        </Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match
                        MatchId="string-equal">
                        <AttributeValue>read</AttributeValue>
                        <AttributeDesignator
                            Category="action" AttributeId="action-id"/>

```

```

        </Match>
    </AllOf>
    <AllOf>
        <Match
            MatchId="string-equal">
            <AttributeValue>create</AttributeValue>
            <AttributeDesignator
                Category="action" AttributeId="action-id"/>
        </Match>
    </AllOf>
    </AnyOf>
</Target>
<Rule
    RuleId="Rule 1" Effect="Permit"
    <Description>
        Rule 1 is applicable for doctor or resident-
        intern access.
        It stipulates the condition that they can only
        access data
        of patients in wards to which they are currently
        assigned
    </Description>
    <Target>
        <AnyOf>
            <AllOf>
                <Match
                    MatchId="string-equal">
                    <AttributeValue>doctor</AttributeValue>
                    <AttributeDesignator
                        Category="access-subject" AttributeId="role"/>
                </Match>
            </AllOf>
            <AllOf>
                <Match
                    MatchId="string-equal">
                    <AttributeValue>res/intern</AttributeValue>
                    <AttributeDesignator
                        Category="access-subject" AttributeId="role"/>
                </Match>
            </AllOf>
        </AnyOf>
    </Target>
    <Condition>
        <Apply FunctionId="string-equal">
            <Apply FunctionId="string-one-and-only">
                <AttributeDesignator
                    Category="access-subject"
                    AttributeId="ward-assignment"/>

```

```

    </Apply>
    <Apply FunctionId="string-one-and-only"
        <AttributeSelector
            Category="resource"
            Path="EHR:patient-data/patient/ward-location/
                /text()"/>
    </Apply>
    </Apply>
</Condition>
</Rule>
<Rule
    RuleId="Rule 2" Effect="Deny"
    <Description>
        Rule 2 denies access to resident-intern in cre-
        ate mode.
    </Description>
    <Target>
        <AnyOf>
            <AllOf>
                <Match
                    MatchId="string-equal">
                    <AttributeValue>resident-intern</AttributeValue>
                    <AttributeDesignator
                        Category="access-subject" AttributeId="role"/>
                </Match>
            </AllOf>
        </AnyOf>
        <AnyOf>
            <AllOf>
                <Match
                    MatchId="string-equal">
                    <AttributeValue>create</AttributeValue>
                    <AttributeDesignator
                        Category="action" AttributeId="action-id"/>
                </Match>
            </AllOf>
        </AnyOf>
    </Target>

<Policy
    PolicyId="Policy 2"
    <Description>
        Just Like Policy 1, Policy 2 is also applicable for
        all Read or Create
        access to Patient data.
    </Description>
    <Target>
        <AnyOf>

```

```

<AllOf>
  <Match
    MatchId="string-equal">
      <AttributeValue>read</AttributeValue>
      <AttributeDesignator
        Category="action" AttributeId="action-id"/>
    </Match>
  </AllOf>
<AllOf>
  <Match
    MatchId="string-equal">
      <AttributeValue>create</AttributeValue>
      <AttributeDesignator
        Category="action" AttributeId="action id"/>
    </Match>
  </AllOf>
</AnyOf>
</Target>
<Rule
  RuleId="Rule 3" Effect="Permit"

          <Description>
            This single rule under Policy 2 is meant exclusively
            for doctor
            access to patient data in situations where the patient
            status is
            critical - hence meant for accessing data during emer-
            gency situations
          </Description>
          <Target>
            <AnyOf>
              <AllOf>
                <Match
                  MatchId="string-equal">
                    <AttributeValue>doctor</AttributeValue>
                    <AttributeDesignator
                      Category="access-subject" AttributeId="role"/>
                  </Match>
                </AllOf>
              </AnyOf>
            </Target>
            <Condition>
              <Apply FunctionId="string-equal">
                <Apply FunctionId="string one and only">
                  <AttributeSelector
                    Category="resource"/>
                </Apply>
              </Apply>
            </Condition>
          </Target>
        <Effect>
          Permit
        </Effect>
      </Rule>
    </Policy>
  </Policy>
</Policies>

```

```
        Path="EHR:patient-data/patient/patient-status/
            /text()"/>
        </Apply>
        <AttributeValue>Critical</AttributeValue>
    </Apply>
</Condition>
</Rule>
<ObligationExpressions>
    <ObligationExpression ObligationId="email"
FulfillOn="Permit">
        <AttributeAssignmentExpression
AttributeId="attribute:mailto">
            <AttributeSelector
                Category="attribute-category:resource"
                Path="PHR:patient-data/patient/primaryphysician/
email"/>
            <AttributeAssignmentExpression>
                <AttributeAssignmentExpression
AttributeId="attribute:text">
                    <AttributeValue>Your Patient record was accessed
by the doctor on
                    Call Dr.
                </AttributeValue>
            </AttributeAssignmentExpression>
            <AttributeAssignmentExpression
AttributeId="attribute:text">
                <AttributeDesignator
                    Category="subject-category:access-subject"
                    AttributeId="subject:subject-id"/>
            </AttributeAssignmentExpression>
        </ObligationExpression>
    </ObligationExpressions>
</Policy>
```

5

Next Generation Access Control

5.1 Introduction

In 2005, NIST initiated a project in pursuit of a standardized ABAC mechanism referred to as the policy machine that allows changes to a fixed set of data elements and relations in the expression and enforcement of ABAC policies [1]. The policy machine has evolved from a concept to a formal specification [2] to a reference implementation and open source distribution [3]. The policy machine has served as a research component in support of a family of American National Standards Institute/ International Committee for Information Technology Standards (ANSI/INCITS) standardization efforts under the title of Next Generation Access Control (NGAC) [4, 5]. In addition to the expression and enforcement of a wide variety of access control policies [6], NGAC facilities can be used to effectuate security-critical portions of the program logic of arbitrary data services and enforce mission-tailored access control policies over data services [7]. Taken together, these NGAC standards define:

- A standard set of data and relations used to express access control policies and attributes, and deliver capabilities of data services to perform operations on data resources;
- A standard set of administrative operations for configuring the data and relations; and,
- A standard set of functions, interfaces, and protocols for trapping and enforcing policy on requests to execute operations on data resources, computing access decisions to permit or deny those requests, and dynamically altering access state in response to access events.

NGAC takes a fundamentally different approach from XACML for representing requests, expressing and administering policies, representing and administering attributes, and computing and enforcing decisions. NGAC is defined in terms of a standardized and generic set of relations and functions that are reusable in the expression and enforcement of policies.

For purposes of brevity and readability, the NGAC specification is presented as a summary that highlights NGAC’s salient features and should not be considered complete. In some instances, actual NGAC relational details and terms are substituted with others to accommodate a simpler presentation.

5.2 Policy and Attribute Elements

NGAC’s access control data is comprised of basic elements, containers, and configurable relations. While XACML uses the terms subject, action, and resource, NGAC uses the terms user, operation, and object with similar meanings. In addition to these, NGAC includes processes, administrative operations, and policy classes. Like XACML, NGAC recognizes user and object attributes; however, it treats attributes along with policy class entities as containers. These containers are instrumental in both formulating and administering policies and attributes. NGAC treats users and processes as independent but related entities. Processes through which a user attempts access take on the same attributes as the invoking user.

Although an XACML resource is like an NGAC object, NGAC uses the term object as an indirect reference to its data content. Every object is an object attribute. The reference to an object is the value of its name attribute. Thus, the value of the name attribute of an object is synonymous with the object. The set of objects reflects entities needing protection, such as files, clipboards, email messages, and record fields.

Like an XACML subject attribute value, NGAC user containers can represent roles, affiliations, or other common characteristics pertinent to policy, such as security clearances.

Object containers (attributes) characterize data and other resources by identifying collections of objects, such as those associated with certain projects, applications, or security classifications. Object containers can also represent compound objects, such as folders, inboxes,

table columns, or rows, to satisfy the requirements of different data services. Policy class containers are used to group and characterize collections of policy or data services at a broad level, with each container representing a distinct set of related policy elements. Every user, user attribute, and object attribute must be contained in at least one policy class. Policy classes can be mutually exclusive or overlap to various degrees to meet a range of policy requirements.

NGAC recognizes a generic set of operations that include basic input and output operations (i.e., read and write) that can be performed on the contents of objects that represent data service resources, and a standard set of administrative operations that can be performed on NGAC access control data that represent policies and attributes. In addition, an NGAC deployment may consider and provide control over other types of resource operations besides the basic input/ output operations. Administrative operations, on the other hand, pertain only to the creation and deletion of NGAC data elements and relations, and are a stable part of the NGAC framework.

5.3 Relations

NGAC does not express policies through rules, but instead through configurations of relations of four types: assignments (define membership in containers), associations (to derive privileges), prohibitions (to derive privilege exceptions), and obligations (to dynamically alter access state).

5.3.1 Assignments and Associations

NGAC uses a tuple (x, y) to specify the assignment of element x to element y . We use the notation $x \rightarrow y$ to denote the same assignment relation. The assignment relation always implies containment (x is contained in y). The set of entities used in assignments include users, user attributes, and object attributes (which include all objects), and policy classes.

To be able to carry out an operation, one or more access rights are required. As with operations, two types of access rights apply: non-administrative and administrative.

Access rights to perform operations are acquired through associations. An association is a triple, denoted by ua — ars — at , where ua is a user attribute, ars is a set of access rights, and at is an attribute, where at may comprise either a user attribute or an object attribute. The attribute at in an association is used as a referent for itself and the policy elements contained by the attribute. Similarly, the first term of the association, attribute ua , is treated as a referent for the users contained in ua . The meaning of the association ua — ars — at is that the users contained in ua can execute the access rights in ars on the policy elements referenced by at . The set of policy elements referenced by at is dependent on (and meaningful to) the access rights in ars .

Figure 5.1 illustrates assignment and association relations depicted as graphs with two policy classes—project access, and file management. Users and user attributes are on the left side of the graphs, and objects and object attributes are on the right. The arrows represent assignment or containment relations and the dashed lines denote associations.

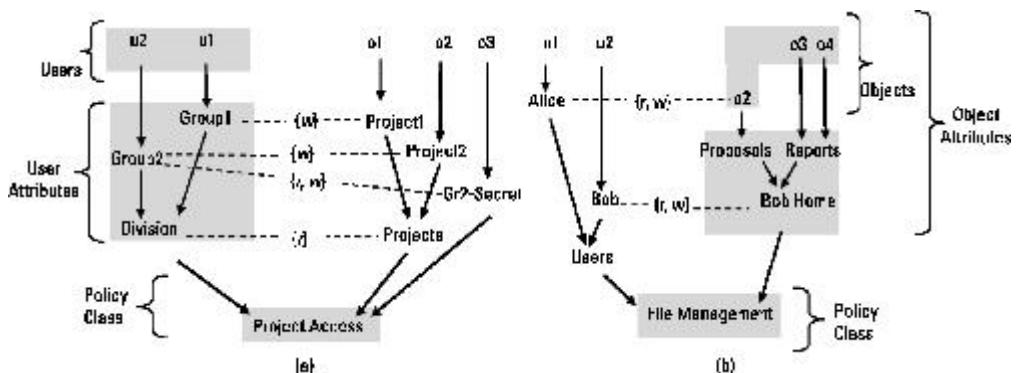


Figure 5.1 Two examples of assignment and association graphs.

Collectively associations and assignments indirectly specify privileges of the form (u, ar, e) , with the meaning that user u is permitted (or has a capability) to execute the access right ar on element e , where e can represent a user, user attribute, or object attribute. Determining the existence of a privilege (a derived relation) is a requirement of, but as we discuss later, not sufficient in computing an access decision.

NGAC includes an algorithm for determining privileges with respect to one or more policy classes and associations. Specifically, (u, ar, e) is a privilege, if and only if, for each policy class pc in which e is contained, the following is true:

- The user u is contained by the user attribute of an association;

- The element e is contained by the attribute at of that association;
- The attribute at of that association is contained by the policy class pc ; and,
- The access right ar is a member of the access right set of that association.

The left and right columns of [Table 5.1](#) respectively list derived privileges for [Figures 5.1\(a\)](#) and [5.1\(b\)](#), when considered independent of one another. [Table 5.2](#) lists the privileges for these graphs in combination.

Note that $(u1, r, o1)$ is a privilege in [Table 5.2](#) because $o1$ is only in policy class project access and there exists an association division— $\{r\}$ —projects, where $u1$ is in division, r is in $\{r\}$, and $o1$ is in projects. Note that $(u1, w, o2)$ is not a privilege in [Table 5.2](#) because $o2$ is in both project access and file management policy classes, and although there exist an association Alice— $\{r, w\}$ — $o2$, where $u1$ is in Alice, w is in $\{r, w\}$, and $o2$ is in $o2$ and file management, no such association exists with respect to project access.

Table 5.1
List of Derived Privileges for the Independent Configuration of Figures 5.1(a) and 5.1(b)

| | |
|---|---|
| $(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1),$ | $(u1, r, o2), (u1, w, o2), (u2, r, o2), (u2, w, o2),$ |
| $(u2, r, o2), (u2, w, o2), (u2, r, o3), (u2, w, o3)$ | $(u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)$ |

Table 5.2
List of Derived Privileges for the Combined Configurations of Figures 5.1(a) and 5.1(b)

| |
|--|
| $(u1, r, o1), (u1, w, o1), (u1, r, o2), (u2, r, o1), (u2, r, o2),$ |
| $(u2, w, o2), (u2, r, o3), (u2, w, o3), (u2, r, o4), (u2, w, o4)$ |

Just as access rights to perform read/write operations on resource objects are defined in terms of associations, so too are capabilities to perform administrative operations on policy elements and relations. In contrast to nonadministrative access rights, where resource operations are synonymous with the access rights needed to carry out those operations (e.g., a read operation corresponding to an r access right), the

authority stemming from one or more administrative access rights may be required for an administrative operation. Administrative access rights to perform an administrative operation maybe explicitly divided into two parts, as denoted by from and to suffixes.

For example, in the context of [Figure 5.1](#), we could create two associations Bob—{create ooa-from}—Bob Home and Division—{create ooa-to}—projects, meaning that the intersection of users in Bob and Division may create object-to-object attribute assignments (ooa) from objects in Bob Home to object attributes in projects. Remember that the set of referenced policy elements in the third term of an association (*at*) is dependent on the access rights in *ars*. As such, the absolute meaning of the two associations is that user *u2* can create assignments from *o2*, *o3*, or *o4* to projects, project 1, or project 2.

5.3.2 Prohibitions (Denials)

In addition to assignments and associations, NGAC includes three types of prohibition relations: user-deny, user attribute-deny, and process-deny. In general, deny relations specify privilege exceptions. We respectively denote a user-based deny, user attribute-based deny, and process-based deny relation by $u_deny(u, ars, pe)$, $ua_deny(ua, ars, pe)$, and $p_deny(p, ars, pe)$, where *u* is a user, *ua* is a user attribute, *p* is a process, *ars* is an access right set, and *pe* is a policy element used as a referent for itself and the policy elements contained by the policy element. The respective meanings of these relations are that user *u*, users in *ua*, and process *p* cannot execute access rights in *ars* on policy elements in *pe*. User-deny relations and user attribute-deny relations can be created directly by an administrator or dynamically through an obligation (see [Section 5.3.3](#)). An administrator, for example, could impose a condition where no user can alter their own tax return, although the user is assigned to an IRS auditor user attribute with capabilities to read/write all tax returns. When created through an obligation, user-deny and user attribute-deny relations can take on dynamic policy conditions. Such conditions can, for example, provide support for separation of duty policies (if a user executed capability x, that user would be immediately precluded from being able to perform capability y). In addition, the policy element component of each prohibition relation can be specified as its complement, denoted by \neg . The respective meaning of $u_deny(u, ars,$

$\neg pe)$, $ua_deny(ua, ars, \neg pe)$, and $p_deny(p, ars, \neg pe)$ is that the user u , and any user assigned to ua , and process p cannot execute the access rights in ars on policy elements not in pe .

Process-deny relations are exclusively created using obligations. Their primary use is in the enforcement of confinement conditions (e.g., if a process reads top secret data, preclude that process from writing to any object not in top secret).

5.3.3 Obligations

Obligations consist of a pair (ep, r) (usually expressed as when $[ep]$ do $[r]$) where ep is an event pattern and r is a sequence of administrative operations, called a response. The event pattern specifies conditions that if matched by the context surrounding a process's successful execution of an operation on an object (an event), cause the administrative operations of the associated response to be immediately executed. The context may pertain to and the event pattern may specify parameters like the user of the process, the operation executed, and the attribute(s) of the object.

Obligations can specify operational conditions in support of history-based policies and data services. Included among history-based policies are those that prevent leakage of data to unauthorized principals. Consider, for example the project access policy depicted in [Figure 5.1\(a\)](#). Although this policy suggests that only group 2 users can read Gr2-Secrets, data in Gr2-Secrets can indeed be leaked to group 1 users. Specifically, $u2$ or one of $u2$'s processes can read $o3$, and subsequently write its content to $o2$, thereby providing $u1$ the capability to read the content of $o3$. Such leakage can be prevented with the following obligation:

When any process p performs (r, o) where $o \rightarrow_{Gr2-Secret}$ do create $p_deny(p, \{w\}, \neg Gr2-Secret)$

The effect of this obligation will prevent a process (and its user) from reading an object in Gr2-Secret and subsequently writing its content to an object in a different container (not in Gr2-Secret).

Other history-based policies include conflict of interest (if a user reads information from a sensitive data set, that user is prohibited from reading data from a second data set) and work flow (approving [writing

to a field of]) a work item enables a second user to read and approve the work item.

5.4 NGAC Decision Function

The NGAC access decision function controls access in terms of processes. The user on whose behalf the process operates must hold sufficient authority over the policy elements involved. The function $\text{process_user}(p)$ denotes the user associated with process p .

Access requests are of the form $(p, op, argseq)$, where p is a process, op is an operation, and $argseq$ is a sequence of one or more arguments, which is compatible with the scope of the operation. The access decision function to determine whether an access request can be granted requires a mapping from an operation and argument sequence pair to a set of access rights and policy element pairs (i.e., $\{(ar, pe)\}$). The request is granted only if the process's user holds the access rights and policy element pairs.

When determining whether to grant or deny an access request, the authorization decision function considers all privileges and restrictions (denies) that apply to a user and its processes, which are derived from relevant associations and denies, giving restrictions precedence over privileges:

A process access request $(p, op, argseq)$ with mapping $(op, argseq) \rightarrow \{(ar, pe)\}$ is granted iff for each (ar_i, pe_i) in $\{(ar, pe)\}$, there exists a privilege (u, ar_i, pe_i) where $u = \text{process_user}(p)$, and (ar_i, pe_i) is not denied for either u or p .

In the context of [Figure 5.1](#), an access request may be $(p, \text{read}, o1)$ where p is $u1$'s process. The pair $(\text{read}, o1)$ maps to $(r, o1)$. Because there exists a privilege $(u1, r, o1)$ in [Table 5.2](#) and $(r, o1)$ is not denied for $u1$ or p , the access request would be granted. Assume the existence of associations Division—{create ooa-to}—projects, and Bob—{create ooa-from}—Bob Home in the context of [Figure 5.1](#), and an access request $(p, \text{assign}, \langle o4, \text{Project1} \rangle)$ where p is $u2$'s process. The pair $(\text{assign}, \langle o4, \text{Project1} \rangle)$ maps to $\{\text{(create ooa-from, } o4), (\text{create ooa-to, } \text{Project1})\}$. Because privileges $(u2, \text{create ooa-from, } o4)$ and $(u2, \text{create ooa-to, } \text{Project1})$ would exist under the assumption, and $(\text{create ooa-}$

from, $o4$) and (create ooa-to, Project1) are not denied for $u2$ or p , the request would be granted.

5.5 Delegation of Access Rights

The question remains, how are administrative capabilities created? The answer begins with a superuser with capabilities to perform all administrative operations on all access control data. The initial state consists of an NGAC configuration with empty data elements, attributes, and relations. A superuser either can directly create administrative capabilities or more practically can create administrators and delegate to them capabilities to create and delete administrative privileges. Delegation and rescinding of administrative capabilities is achieved through creating and deleting associations. The principle followed for allocating access rights via an association is that the creator of the association must have been allocated the access right over the attribute in question (as well as the necessary create-assoc-from and create-assoc-to rights) to delegate them. The strategy enables a systematic approach to the creation of administrative attributes and delegation of administrative capabilities, beginning with a superuser and ending with users with administrative and data service capabilities.

5.6 NGAC Administrative Commands and Routines

Access requests bearing administrative operations can create and destroy basic elements, containers and relations. Each administrative operation corresponds on a one-to-one basis to an administrative routine, which uses the sequence of arguments in the access request to perform the access. Each administrative operation is carried out through one or more primitive administrative commands. NGAC defines the complete set of administrative commands and their behavior in detail. The definitions specify the preconditions that need to exist for the effect of a command to occur, and the specific effect that the command has on the contents of NGAC's policy information point (policies and attributes store).

The access decision function grants the access request (and initiation of the respective administrative routine) only if the process holds all prohibition-free access rights over the items in the argument sequence

needed to carry out the access. The administrative routine, in turn, uses one or more administrative commands to perform the access. Administrative commands and routines are thus how policy specifications and attributes are formed.

Consider the administrative command CreateAssoc shown below, which specifies the creation of an association. The preconditions here stipulate membership of the x , y , and z parameters respectively to the user attributes (ua), access right sets (ars), and attributes (at) elements of the model. The body describes the addition of the tuple (x, y, z) to the set of associations (ASSOC) relation, which changes the state of the relation to ASSOC'.

```
createAssoc (x, y, z)
  x ∈ UA ∧ y ∈ ARS ∧ z ∈ AT ∧ (x, y, z) ∉ ASSOC
  {
    ASSOC' = ASSOC ∧ {(x, y, z)}
  }
```

An administrative routine consists mainly of a parameterized interface and a sequence of administrative command invocations. Each formal parameter of an administrative routine can serve as an argument in any of the administrative command invocations that make up the body of the routine. Administrative routines are used in a variety of ways. Although an administrative routine must be in place on a one-to-one basis to carry out an administrative operation, they can also be used to carry out more complex administrative tasks comprising of a sequence of administrative actions.

Consider the following administrative routine that creates a file management user in the context of [Figure 5.1 \(b\)](#). The routine assumes the pre-existence of the user attribute users assigned to the file management policy class shown in [Figure 5.1 \(b\)](#).

```

create-file-mgmt-user(user-id, user-name, user-home) {
    createNAInIA(user name, Users);
    createUinJA(user-id, user-name);
    createOainPC(user-home, File Management);
    createAssoc(user-name, {r, w}, user-home);
    createAssoc(user-name, {create o to, delete o from}, user-home);
    createAssoc(user-name, {create-coa-from, create-coa-to, delete-
coa-from, create-oaa-from,
        create oao to, delete oao from}, user-home);
    createAssoc(user-name, {create-assoc-to, delete-assoc-to, m a o
catc, w-allocatc}, user-home);
}

```

This routine with parameters ($u1$, *Bob* and *Bob Home*) could have been used to create file management data service capabilities for user $u1$ already in [Figure 5.1\(b\)](#). Through the routine the user attribute Bob is created and assigned to users, and user is created and assigned to Bob. In addition, the object attribute Bob Home is created and assigned to policy class file management. In addition, user $u1$ is delegated administrative capabilities to create, organize, and delete object attributes (presented folders) in Bob Home, and $u1$ is provided with capabilities to create, read, write, and delete objects that correspond to files and place those files into his folders. Finally, $u1$ is provided with discretionary capabilities to grant to other users in the users container capabilities to perform read/write operations on individual files or to all files in a folder in his home.

5.7 Arbitrary Data Service Operations

NGAC recognizes administrative operations for the creation and management of its data elements and relations that represent policies and attributes, and basic input and output operations (e.g., read and write) that can be performed on objects that represent data service resources. In accommodating data services, NGAC may establish and provide control over other types of operations, such as send, submit, approve, and create folder. However, it does not necessarily need to do so. This is because the basic data service capabilities to consume, manipulate, manage, and distribute access rights on data can be attained as combinations of read/write operations on data and administrative operations on data elements, attributes, and relations. For example, the create-file-mgmt-user routine specified above provides a user with capabilities to create

and manage files and folders, and control and share access to objects in the user’s home directory.

5.8 NGAC Functional Architecture

NGAC’s functional architecture (shown in [Figure 5.2](#)), involves several components that work together to bring about policy-preserving access and data services:

1. One or more policy enforcement points (PEPs);
2. One or more policy decision points (PDPs);
3. Zero or one event processing point (EPP);
4. One policy administration point (PAP);
5. One policy information point (PIP); and
6. One or more resource access points (RAPs).

NGAC supports two fundamental types of accesses within the functional architecture—resource access and administration access.

The descriptions of resource and administration accesses given below assume that an authenticated user has established a session with a PEP, and that all access requests can be securely identified with that session. A user can have only one session active at any time, but may have multiple processes operating on its behalf within that session.

Least privilege is an established administrative practice of assigning users and processes the minimal authorization necessary for the performance of their job function, and no more. NGAC supports the concept of least privilege for both resource and administration access, by allowing different authorizations for a user or its processes to become available at different times for the performance of different tasks. Processes acting for a user within the user’s session can be restricted to a subset of the user’s authorization, allowing them to be attenuated at a granularity below that of the user.

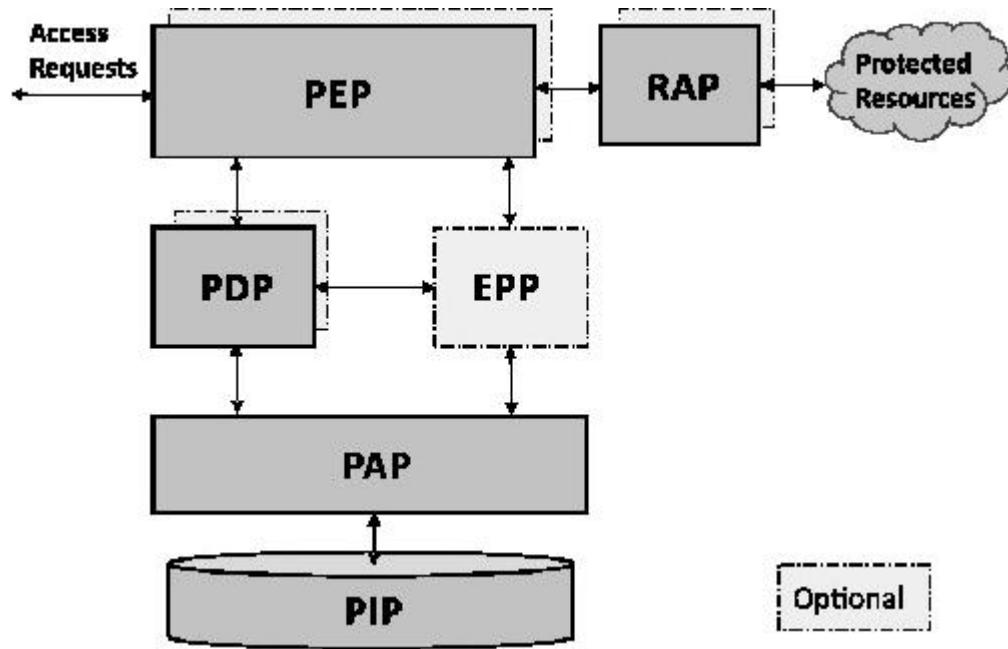


Figure 5.2 NGAC standard functional architecture.

5.8.1 Resource Access

A resource access is the only way in which users can gain access to protected resources in NGAC. A resource access begins when a user launches a client application, creating a process that attempts access to a resource via a PEP. Processes run on behalf of a specific user within a single session, and may instantiate other processes. Each process of a user uses the PEP to which the user has formed a session to request access to a protected resource. This is the only way a process is able to gain access to protected resources in NGAC.

A resource access references an object associated with the protected resource targeted. NGAC supports conceptual operations such as read/write or on/ off/reset, against protected resources. The behavior of resource operations is not defined by NGAC. The actual location and required routing to the resource are maintained within the PIP. Protected resources may have different sensitivities and other characteristics designated by attributes, for which policy can be formulated to protect against the leakage of information.

The function of the PEP is to ensure that only those requests for access that meet specific requirements are granted access to the protected resources. The PEP submits individual access requests to a PDP for adjudication. The PDP then obtains additional details necessary to

adjudicate an access request by retrieving authorization information related to the request from the PIP, via queries issued to the PAP.

The PDP renders an access decision based on:

1. The existence of all requisite privileges for the access request; and
2. The absence of any restrictions that countermand a requisite privilege.

An access request is granted by the PDP if (1) and (2) are both satisfied. The access request is denied for all other possibilities.

If an access request is granted, the PDP retrieves information for locating the resource identified in the request (i.e., via the identifier of the associated object, translated into a specific system resource at a specified location associated with a specified RAP) and conveys the location information obtained, along with the decision results, to the PEP from which it received the request. The PEP communicates with a specific RAP to perform the operation specified in the access request on the resource identified. Once the operation is complete, the PEP then returns the status information about the operation, and optionally data resulting from it, to the originating process.

If an EPP is present in the system, the PEP generates an event context for each successfully completed resource access, and forwards it to the EPP. The EPP uses the PAP to match the event context against the event patterns of obligations stored in the PIP. For each match, the EPP conveys the event response from the obligation to a PDP for validation and processing, for which the PDP uses the PAP. The resulting changes to the policy information in the PIP can affect the access decisions on future access requests.

5.8.2 Administrative Access

An administration access is the only way in which users gain access to policy information in NGAC. Like a resource access, an administration access begins when a user launches a client application, creating a process that attempts access to policy information by conveying an administrative operation and the required operands to a PEP. A user's processes use the PEP, to which the user has formed a session, to request access to policy information. This is the only way a process can gain

access to policy information in NGAC. NGAC administrative operations are used to create and destroy the basic elements, containers, and relations that are maintained by the PIP, and consequently, change the policies enforced by the NGAC framework.

Client applications launched by a user can administer the contents of the PIP, if sufficient authorization is held by the user. Policies governing administration responsibilities can be centralized to a single authority or decentralized to allocate responsibilities selectively among multiple authorities.

For each administrative access attempt, the PEP in turn submits an individual access request to a PDP for adjudication. Administration access requests are treated slightly differently than resource access requests by the NGAC architecture, however. The main difference is the way in which the request is processed by the PDP once the access decision is computed. If a grant decision is rendered, the PDP takes an additional step to carry out the access before returning the results to the PEP. The PDP communicates with the PAP to perform the administration operation given in the access request on the relevant policy information maintained by the PIP. For decisions, other than a grant decision, the results are returned to the PEP, like that done for a resource access.

If an EPP is present in the system, the PDP generates an event context for each successfully completed administration access, and forwards it to the EPP. The EPP uses the PAP to match the event context against the event patterns contained in obligations stored in the PIP, and processes the event responses using a PDP (not necessarily the same one that generated the event context) in the same way as done for a resource access.

5.9 Conclusion

NGAC controls access using a standard set of data and relations that express policies and attributes. Within NGAC, access control data is made up of basic elements, containers, and configurable relations. Instead of expressing policies through rules, NGAC uses configurations of four types of relations: assignments, which define membership in containers; associations, which derive privileges; prohibitions, which derive privilege exceptions; and obligations, to dynamically alter access

state. Together, these features provide great expressive power for policies, without sacrificing performance.

References

- [1] Ferraiolo, D. F., et al., “Composing and Combining Policies Under the Policy Machine,” *Tenth Association for Computing Machinery Symposium on Access Control Models and Technologies (SACMAT ‘05)*, Stockholm, Sweden, 2005, pp. 11-20.
- [2] Ferraiolo, D. F., S. Gavrila, and W. Jansen, “Policy Machine: Features, Architecture, and Specification,” National Institute of Standards and Technology (NIST) Internal Report (IR) 7987 Revision 1, October 2015.
- [3] NIST Policy Machine Versions 1.5 and 1.6, Harmonia [Website], <https://github.com/PM-Master>.
- [4] INCITS, *Information technology—Next Generation Access Control—Functional Architecture (NGAC-FA)*, INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.
- [5] INCITS, *Information Technology—Next Generation Access Control—Generic Operations and Data Structures (GOADS)*, INCITS 526-2016, American National Standard for Information Technology, January 2016.
- [6] Ferraiolo, D. F., V. Atluria, and S. I. Gavrila, “The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement,” *Journal of Systems Architecture*, April 2011, Vol. 57, No. 4, pp. 412-424.
- [7] Ferraiolo, D. F., S. Gavrila, and W. Jansen, “On the Unification of Access Control and Data Services,” *Proc IEEE Int Conf Inf Reuse Integr*, 2014, pp. 450 – 457.

6

ABAC Policy Verifications and Testing¹

6.1 Introduction

ABAC policies are specified to facilitate managing and maintaining ABAC systems, therefore faulty policies, misconfigurations, or flaws in software implementation can result in serious vulnerabilities. However, the correct implementations of ABAC policies by ABAC mechanisms are very challenging problems. It is common that a system's privacy and security are compromised due to the misconfiguration of access control policies instead of the failure of cryptographic primitives or protocols. This problem becomes increasingly severe as software systems become more and more complex, and are deployed to manage a large amount of sensitive information and resources that are organized into sophisticated structures. Therefore, identifying discrepancies between ABAC policy specifications and their intended function is crucial because correct implementation and enforcement of policies by applications is based on the premise that the policy specifications are correct.

ABAC models are usually written to bridge the rather wide gap in abstraction between ABAC policies and mechanisms to formally and precisely capture the safety requirements that ABAC systems should adhere to. As a result, policy specifications represented by models must undergo rigorous verification and validation through systematic verification and testing to ensure that the policy specifications truly encapsulate the desires of the policy authors. Verifying the conformance of ABAC policies and models is a nontrivial and critical task. One important aspect of such verification is to formally check the inconsistency and incompleteness of the model and policy safety requirements, because an ABAC model and its implementation do not necessarily explicitly express the policy, which can also be implicitly embedded by mixing with direct access constraints or other ABAC models.

In this chapter, we discuss general approaches for the verification for ABAC models and the testing of model implementations by first defining standardized structures of ABAC classes. We then demonstrate the expressions of ABAC models and safety requirements in formal specifications of model checkers for the use of *black box* and *white box* model verifications that verify the integrity, coverage, and confinement of the specified safety requirements against models. In addition, an efficient way of generating test cases for the implementation from a model is discussed.

6.2 ABAC Policy Classes

ABAC policy can be formally presented by ABAC models, and enforced by the mechanism. An ABAC model is useful for proving theoretical limitations of an ABAC system, so that a mechanism can be designed to adhere to the properties of the policy. Users see an ABAC model as an unambiguous and precise expression of requirements. Vendors and system developers see ABAC models as design and implementation requirements. On one extreme, an ABAC model may be rigid in its implementation of a single policy. On the other extreme, an ABAC model will allow for the expression a wide variety of policies combined. In general, all nondiscretionary ABAC policies can be modeled by finite state machine (FSM) models from one of the following policy classes: static, dynamic, and historical.

6.2.1 Static Policy Class

Static ABAC policy class regulates the access permissions by static system states or conditions such as rules, attributes, and environment conditions (times and locations for access). The properties of this policy class can be specified by asynchronous or direct specification expressions of an FSM model. The transition relation of authorization states is directly specified as a propositional formula in terms of the current and next values of the state variables. Any current state/ next state pair is in the transition relation if and only if it satisfies the formula, as demonstrated in Example 6.1:

Example 6.1: Static ABAC Model

```

VARIABLES
    access_state : boolean; /* 1 as grant, 0 as deny */

INITIAL
    access_state := 0;

TRANS /* transit to next access state */
    next (access_state) :=
        ((constraint_1 & constraint_2 & ... constraint_m) |
         (constraint_a & constraint_b & ... constraint_m)
        . . .
    ;

```

The system state of access authorization is initialized as the deny state and moved to the grant state for any access request that complies with the constraints of the rule corresponding with each constraint predicate (i.e., *constraint₁* and *constraint_n*) in a rule, and stays in the deny state otherwise.

6.2.2 Dynamic Policy Class

Dynamic ABAC policy class may include temporal constraints that regulate access permissions by dynamic system states or conditions. An ABAC model with this class specifies that accesses are permitted only by a certain subject to a certain object with certain limitations (e.g., object *x* can be accessed only no more than *i* times simultaneously by subject group *y*). For example, if a subject's role is a cashier, he or she cannot be an accountant at the same time when handling a customer's checks. This policy class can be specified with asynchronous or direct specification expressions of an FSM model, which uses a variable semaphore to express the dynamic properties of the authorization decision process. Another example of dynamic constraint states is enforcing a limited number of concurrent accesses to an object. The authorization process for a subject has four states: idle, entering, critical, and exiting. A subject is normally in the idle state. The subject is moved to the entering state when the subject wants to access the critical object. If the limited number of access times is not reached, the subject is moved to the critical state, and the number of the current access is increased by 1. When the subject finishes accessing the critical object, the subject is moved to the exiting state, and the number of the current access is decreased by 1. Then the subject is moved from the exiting state to the idle state. The authorization process can be modeled as the following asynchronous FSM specification; Example 6.2:

Example 6.2: Dynamic ABAC Model.

VARIABLES

```
    count, access_limit : INTEGER;

    request_1 : process_request (count);
    request_2 : process_request (count);
    .....
    request_n: process_request (count);
        /*max number of subject requests allowed by the
system*/
    access_limit := k; /*max number of concurrent
access*/
    court := 0; act {rd, wrt}; object {obj};
process_request (access_limit) {
    VARIABLES
        permission : {start, grant, deny};
        state : {idle, entering, critical, exiting};
    INITIAL_STATE (permission) := start;
    INITIAL_STATE (state) := idle;
    NEXT_STATE (state) := CASE {
        state == idle : (idle, entering);
        state == entering & ! (count > access_limit):
critical;
        state == critical : (critical, exiting);
        state == exiting : idle;
        OTHERWISE: state};
    NEXT_STATE (count) := CASE {
        state == entering : count + 1;
        state == exiting : count -1;
        OTHERWISE: DO NOTHING };
    NEXT_STATE (permission) := CASE {
        (state == entering) & (act == rd) & (object
== obj): grant;
        OTHERWISE: deny;
    }
}
```

6.2.3 Historical Policy Class

Historical ABAC policy classes regulate access permissions by historical access states or recorded and predefined series of events. The properties of this policy class can be best described by synchronous or direct specification expressions of an FSM model. For example, the following (Example 6.3) synchronous FSM specification specifies a Chinese Wall access control policy where there are two conflict of interest groups (*COI1*, *COI2*) of objects:

Example 6.3: Historical ABAC Model.

VARIABLES

```
access {grant, deny};
```

```
act (rd, wrt);
o_state {none, COI1, COI2};
u_state {1, 2, 3};
TNTTTAT_STATE(u_state) := 1;
TNTTTAT_STATE(o_state) := none;
NEXT_STATE(state) := CASE {
    u_state = 1 & act == rd & o_state == COI1: 2;
    u_state = 1 & act == rd & o_state == COI2: 3;
    u_state == 2 & act == rd & o_state == COI1: 2;
    u_state == 2 & act == rd & o_state == COI2: 2;
    u_state == 3 & act == rd & o_state == COI1: 3;
    u_state == 3 & act == rd & o_state == COI2: 3;
    OTHERWISE: 1; };
NEXT_STATE(access) := CASE {
    u_state == 2 & act == rd & o_state == COI1: grant;
    u_state == 3 & act == rd & o_state == COI2: grant;
    OTHERWISE: deny; };
NEXT_STATE(act) := act;
NEXT_STATE(o_state) := object;
```

Note that in practice, the same ABAC policies may be expressed by multiple different ABAC models or expressed by a single model in addition to extra constraint rules outside of the model.

6.3 Access Control Safety and Faults

Safety is the fundamental property of an access control system, which ensures that the access control system will not result in the leakage/blockage of permissions to an unauthorized/ authorized principal. Thus, an access control system is safe if no privilege can be escalated to unauthorized or unintended principals, but the correct privileges are always accessible to authorized principals. Safety is specified through the use of restricted access control models that can be proven in general for that model describing the safety requirements of any configuration [1].

Among all the safety features, separation of duties (SoD) [2] are more dynamic than others. SoD refers to the principle that no subject should be given enough privileges to misuse the system on their own. For example, the person authorizing paychecks should not also be the one who can prepare them. SoD can be enforced either statically (by defining conflicting roles, that is roles which cannot be executed by the same subject) or

dynamically (by enforcing the control at access time). Access control faults compromise the safety, at semantic level, access control faults are usually caused by erroneous or inefficient representation of access control properties or permission algorithms. At a syntactic level, access control faults are simply caused by implementation errors in access control mechanisms such as coding errors, or misconfigurations of access control systems. In general, access control faults can be categorized into the following classes.

Privilege Leakage

Privilege (i.e., action and resource pair) leakage refers to situations in which subject is able to access resources that are prohibited by the safety requirements. Such leakage may cause either the privilege escalation from one resource domain or class to prohibited ones such as leakage from lower to higher ranks of MLS policy [2], or privilege leak such as from one role to other prohibited ones of an RBAC policy [2]. Privilege leakage can be caused by mistaken privilege assignment directly or careless privilege inheritance indirectly.

Privilege Blocking

Opposite to privilege leaking, a privilege blocking fault blocks a legitimate access to rightful resources. Privilege blocking can also occur when the properties of access control policy cannot render a grant or deny decision, or there is no available logic in the access control policy algorithm for evaluating the access request. Privilege blocking can also be a result of the deadlock of access rules specification where a rule has a dependency on another rule or rules, which eventually depend back on the rule itself, such that a subject's request will never reach a decision because of the cyclic referencing.

Cyclic Inheritance

Cyclic inheritance fault refers to the problem of privileges inheritance from other subjects/ groups, which in a chain of inheritance relation inherit back to the subject/group's privilege. For example, subject x inherits privilege from subject y , which inherit privilege from subject z , which inherits privilege from subject x . Cyclic inheritance leads to undecidable or infinite access evaluation process.

Privilege Conflict

Unlike regular programming logic in which a later value assignment of a variable overwrites the previous assigned value of the same variable, the rules of an access control policy normally have no precedence consideration in permission evaluation. In other words, access control rules will not be overwritten by other rules unless specifically allowed to. Thus, privilege conflicts appear when the specifications of two or more access rules result in the conflicting decisions of permitting subjects' access requests by either direct or indirect (inherit) access assignments. In addition, when multiple policies are evoked for permission, conflicting decisions between policies may occur.

Multi-policies Considerations

In an enterprise environment, it may be required to have access control policies specified independently by different collaborative or networked systems in the enterprise. Thus, an intersystem access request may be evaluated by more than one policy that the requesting subject is governed under. Thus, access control policy autonomy should also be preserved for secure intersystem access. Maintaining the autonomy of all collaborative systems is a key requirement of the policy for interoperation. The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure intersystem access. The principle of security states that if an access is denied by an individual system, it must also be denied under secure intersystem access. In a collaborative system, violations of secure intersystem access can be caused by adding intersystem privilege inheritance relations. For example, Figure 6.1 shows that privilege k inherits privilege j through legal intersystem privilege inheritance (because both have the same privilege level j), which is granted in network x but denied in network y . These types of violations can be detected by checking for cyclic inheritance, privilege leakage, and SoD violation. Thus, both security and autonomy can be characterized as safety requirements of a multipolicies access control system, which should be preserved during collaborations. A metapolicy is a policy that is usually applied for reconciling policy autonomy difference or to handle priorities of access decisions rendered from more than one policy. Thus, in addition to autonomy requirements, an access control safety requirement may include priority model within the metapolicy [3].

6.4 Verification Approaches

The fundamental goal of ABAC policy and implementation verification is to detect conflicting or missing rules (i.e., policy statements) by verifying the ABAC policy model and testing outputs of the policy. To achieve this, semantic and syntactic methods with black box and/ or white box testing techniques may be used. Although the general safety computation is proven undecidable [4] for discretionary policies which are impossible to be described by static policy models, practical safety constraints such as confinements can be specified for discretionary policies. As a result, verifications can be performed upon the constraints.

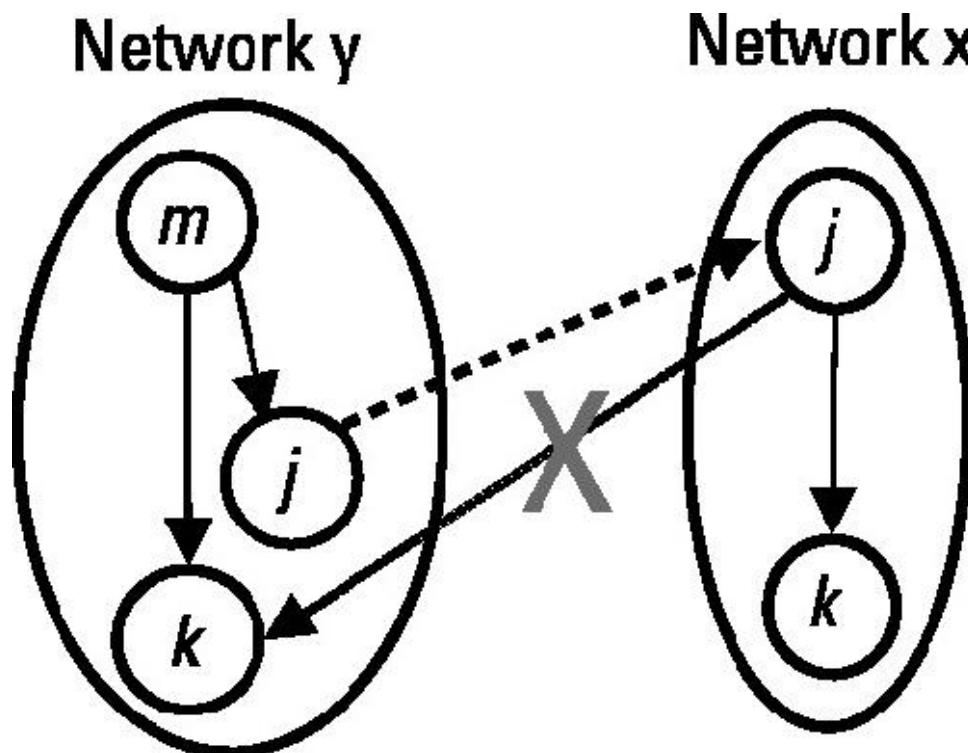


Figure 6.1 Privilege leaks through intersystem privilege inheritance.

In a nutshell, ABAC policy verification must test if the safety requirements of an ABAC policy are incorporated in the expressed model, which will be the blue print for implementing the ABAC system. The specification of safety requirements can be ABAC properties, business requirements, specifications of expected/ unexpected system security features, or direct translations of policy features. Safety requirements can also include privilege inheritance, for example to verify a SoD property. Safety requirements will specify that (1) subject x and y are mutually exclusive if neither one inherits the other's privilege directly or indirectly, (2) if subjects x and y are mutually exclusive, then no other subject inherits

privilege from both of them. Similar to SoD, dynamic SoD (DSoD) has the safety requirement (3) If SoD holds, then DSoD is maintained. Thus, (1) and (2) must be guaranteed [3].

Note that an ABAC policy is not necessarily explicitly expressed by a single model; it can also be implicitly embedded by mixing with direct access constraints or other ABAC models. Thus, an ABAC policy may be expressed by combining multiple ABAC models (e.g. for policy combinations) or additional constraints outside of the model into one combined model. The principle of ensuring the conformance of a model to the policy is to formally detect inconsistency and incompleteness faults as described in [Section 6.3](#). In the former case, for example, an access request can be both accepted and denied, while in the latter case the request is neither accepted nor denied according to the model.

6.4.1 Model Verification

The general approach for checking the correct specification of an ABAC model is to use black box methods to verify the ABAC model against safety requirements. And since the confidence of the model's correctness depends on the quality of the safety requirements, a white box property assessment method on entities in the model and safety requirements is required to assess the sufficiency of the safety, covering, and confinement of the model [5].

In terms of ABAC attributes, the formal definition of an ABAC model can be illustrated by a deterministic finite state transducer of a model corresponding to a finite state machine (FSM) with a five-tuple $M = (\Sigma, ST, s_0, \delta, F)$, where Σ is the input alphabet that represents the attributes associated with subjects, actions, objects, and environment conditions. ST is a finite, non-empty set of recorded ABAC system states and permissions, s_0 is the initial state, δ is the state-transition function, where $\delta : ST \times \Sigma \rightarrow ST$, F is the set of final states include *Grant*, *Deny* as the output.

For static ABAC models, as described in 6.2.1, the FSM M_{static} does not require intern states to reach the permission state, thus $F = ST = \{Grant, Deny\}$, that is, M_{static} is just a straightforward FSM model without state transitions. For dynamic ABAC models as described in 6.2.2, the input alphabets of FSM $M_{dynamic}$ are $\Sigma_{dynamic} = \{gCond_1, \dots, gCond_n\}$, where global condition $gCond_i$ is the threshold indicator of the access limitation, such as the number of persons that have to access at the same time in a n-person control policy [2], or the maximum number of accesses allowed for a

Limited_Number_of_Access policy. For historical ABAC models as described in 6.2.3, the input alphabets of the FSM $M_{historical}$ are $\Sigma_{historical} = \{ \dots sCond_i, aCond_i, oCond_i \dots \}$, where subject condition $sCond_i$, action condition $aCond_i$, and object condition $oCond_i$ contribute to a historical event that is used as determining factors for the next permission decision. Note that it is possible for different types of ABAC models to combine into one model such that $M_{combine} = \{M_{static} \cup M_{dynamic} \cup M_{historical}\}^2$.

An ABAC safety requirement p is expressed by the proposition $p: ST \times \Sigma^2 \rightarrow ST$ of FSM, which can be collectively translated in terms of logical formulae, such that $p = (sCond_1 * \dots * sCond_n * aCond_1 * \dots * aCond_n * oCond_1 * \dots * oCond_n * gCond_1 * \dots * gCond_n) \rightarrow d$, where $p \in P$ and d is the permission is a set of safety requirements, and $*$ is a Boolean operator. The purpose of model checking is to verify the set ST in M in which p is true according to an exhaustive state space search. In addition, by verifying the set of states in which the negation of p is true, we can obtain the set of counterexamples to make the assertion that p is true. The satisfaction of an ABAC model M to the safety requirement P by model checking is composed of two requirements:

1. Safety, where M satisfies P . That is, there is no violation of rules to the logic specified in P , and it is assured that M will eventually be in a desired state after it takes actions in compliance with a subject access request.
2. Liveness, where M will not have unexpected complexities. That is, there is neither a deadlock in which the system waits forever for system events, nor a livelock in which the model repeatedly executes the same operations forever.

Thus, the ABAC rules define the system behaviors that function as the transition relation δ in M . Then, when the safety requirement is represented by temporal logic formula p , we can represent the assertion that model M satisfies p by $M \models Ap \rightarrow AXd$, where temporal logic quantifier A represents “always”, and logic quantifier X represents “is true next state”. The purpose of safety and liveness verification using model checking is to determine whether these assertions are true, and to identify a state in which the assertions are not true as a counterexample for the assertions. Since the behavior of the ABAC mechanism can be represented by FSM M , and the safety requirements that M must satisfy can be represented by temporal logic formulas, we can define the correctness more precisely as that the

model can be led from every possible state that is reachable from initial states to the defined final state while complying with the safety requirements.

Even though checked by the black box testing as described above, the model is not fault proof because the temporal logic in the model might not be thorough in covering all possible values of all rules, or all conditions in rules. For example, two states determined by opposite assignments of the same Boolean variable are embedded in different substate modules, where a third state is triggered only when the constraints of the two states are satisfied. As demonstrated in [Figure 6.2](#), the two rules will never agree due to the self-negation to the same constraint. In this case, the third state will never be satisfied, but proven correct without counterexamples through the black box checking.

To detect this kind of semantic fault, the white box testing, based on code analysis, should be applied such that the resulting mutated versions are used to detect faults of the model. Testing for mutations makes sure all paths of a part of a model code are covered by setting the related target variables to all possible values as input, and checking to see if there are different outcomes from the changes. If there is none, then either the code that had been mutated was never executed or the variable was unable to locate the faults. As shown in [Figure 6.2](#), if we mutate the first case module to change x to $!x$, the resulting access state will be grant without being affected. That works the same for the second case module. This fault demonstrates that there is a redundancy in the model, which does not violate the temporal logic of the model. Further investigation to check the model that relates to the variable should reveal that the $(p == i \ \& \ q == j) \rightarrow access == grant$ safety requirement will never happen. Note that this fault can be caught if one more safety requirement $E! (p == i \ \& \ q == j)$ (which means there exists some path that eventually in the future will satisfy $!(p == i \ \& \ q == j)$ in CTL model checking) is specified for the black box checking. Hence, it is not expected that all safety requirements are perfectly specified in the beginning. Thus, white box checking can be used as a second line of defense against faults that will not be spotted by black box checking.

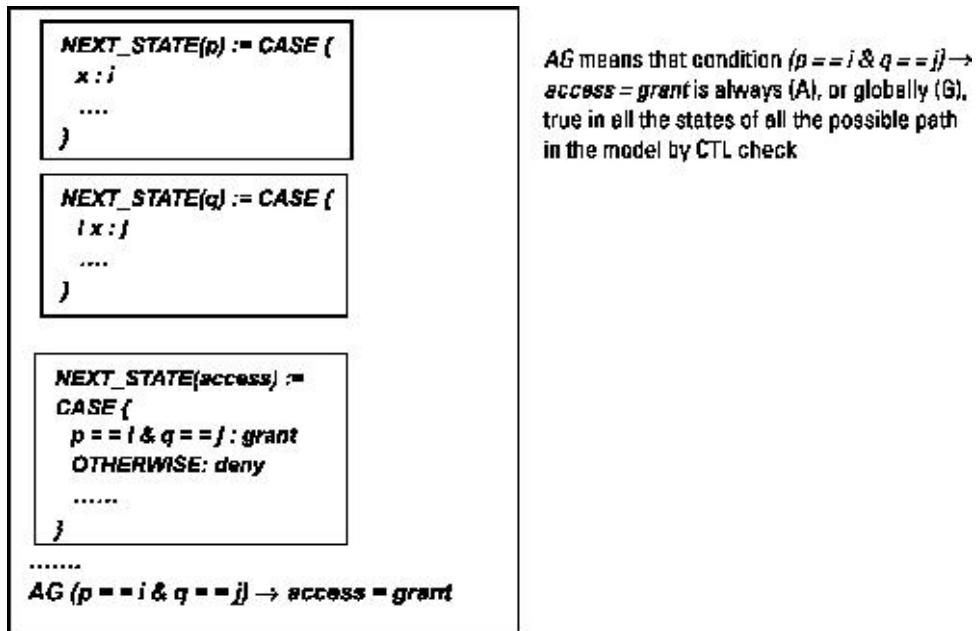


Figure 6.2 Example of never-achieved rules and the safety requirement in an ABAC model.

Most faults in an ABAC model result from the nondeterministic automata of FSM states, for example, in [Figure 6.3](#), white box checking will detect that the value x will result in a grant of access when it is either s or t . This does not violate the safety requirement, however, the safety property will not be maintained if a more stringent safety requirement requires that only one value of x attribute is desired from the policy.

Another example shows a transition to an unspecified state for a certain range of data values, such as in [Figure 6.4](#), there is no way for the black box checker to figure out the value of access when the x value is other than s , unless we check with the safety requirement $AG ! (p == i) \rightarrow access = deny$. This uncovered value can be detected by the white box checking when different values were assigned to x , which does not match any expected case condition, and results in the same grant of access. Thus, the safety requirement verification informs the users which rules are not covered by the existing safety requirement so that the users can add new properties to cover the uncovered rules.

```

NEXT_STATE(p) := CASE {
    x==s|t : i
    ....
}

```

```

NEXT_STATE(access) := CASE {
    p == i : grant
    OTHERWISE: deny
    ....
}

```

.....

AG (p ==i) → access = grant

```

NEXT_STATE(p) := CASE {
    x==s : i
    ....
}

```

```

NEXT_STATE(access):= CASE {
    p = i : grant
    ....
}

```

.....

AG (p ==i) → access = grant

Figure 6.3 Example of ambiguous value and the safety requirement in an ABAC model.

6.4.2 Coverage and Confinements Semantic Faults

The rules in the ABAC policy, model, and safety requirements may each describe their own space of permission conditions, and may not be congruent in one space as the initial relation illustrated examples in [Figure 6.5](#). The safety and liveness check can assure only the logical integrity of some rules against some safety requirements. The complete satisfaction of a model to its policy requires fixing of coverage and confinement faults if any violations are detected by additional coverage and confinement checks (CCC), which is the second line of defense against such semantic faults.

CCC requires mutant versions of the model, and extra modified properties for additional model checking. As illustrated in [Figure 6.5](#), the goal of CCC is to ensure that the rules in the safety requirement are completely covered by the model, and to confirm that no exceptional access permissions are granted unless intentionally allowed. The first step of CCC is to discover the rules, which are seeped through the specification of the safety requirement by applying white box checking on mutated versions of the model. The second step is to detect unexpected access permission that might not be the intention of the policy author by applying model checking on modified rules extracted from the original ones.

Rule Coverage Checking

The key notion of rule coverage checking is to synthesize a version of the given model in such a way that the permission of its rules is mutated, such that rule r is changed to $\sim r$. If safety requirements are satisfied by both mutated and original models through model checking, then some of the rules and their mutants would never be applied to the safety requirements; in other words, the safety requirements do not cover all the rules in the model.

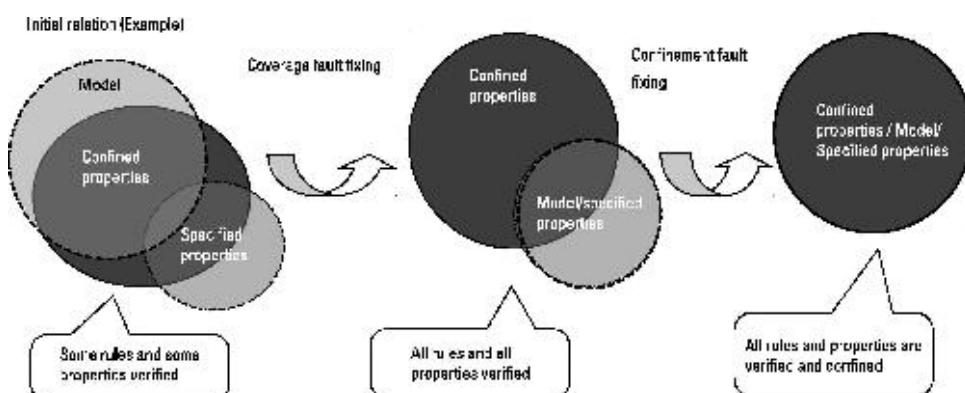


Figure 6.5 Relations of policy, model, and safety requirements.

As illustrated in [Figure 6.6](#), the safety and liveness checking verifies that the model conforms to the safety requirement $AG(q == i) \rightarrow access = grant$ without counterexamples; however, by applying the CCC by mutating the rule $u == j : grant$ to $u == j : deny$ for the coverage checking, the result shows that the safety requirement satisfies the mutated rules as well (without counterexamples), indicating that the variable u was never applied to the safety requirement $AG(q == i) \rightarrow access = grant$. This result shows that the rule $u == j : grant$ is not verified with the property $AG(q == i) \rightarrow access = grant$. One way of addressing this insufficiency is adding a new property that describes proper control of u . Note that it is necessary to check every rule in the model against all safety requirements to achieve thorough verification.

6.4.3 Property Confinement Checking

Property confinement checking ensures that there is no exceptional permission allowed in addition to the specified safety requirement; this checking requires a modified safety requirement to be added for the next run of model checking. Confinement checking should discover the discrepancy of the specified safety requirement and the safety requirement the ABAC policy author intends. The rationale is that if the model does not satisfy the modified safety requirement, then there are exceptional access permissions that leak through the safety requirement. [Figure 6.7](#) shows a transition to an unspecified state for a certain range of data values that allow exceptional permissions not covered by a specified safety requirement because the value of access when the u value is different than i (such as $u = j$) also grants access permission by the rule $otherwise: grant$. This fault can be caught by a counterexample $AG(u == j) \rightarrow access = grant$ when checking the model against the additional confinement property $AG(u == i) \rightarrow access = deny$ derived from original property $AG(u == i) \rightarrow access = grant$. The additional model checking for confinement verification informs the ABAC policy authors which safety requirement is not confined so that the ABAC policy author can add new rules to enforce the safety of the model. As in this case, changing the rule $otherwise: grant$ to $otherwise : deny$ and adding all granted rules in the state will correct the problem.

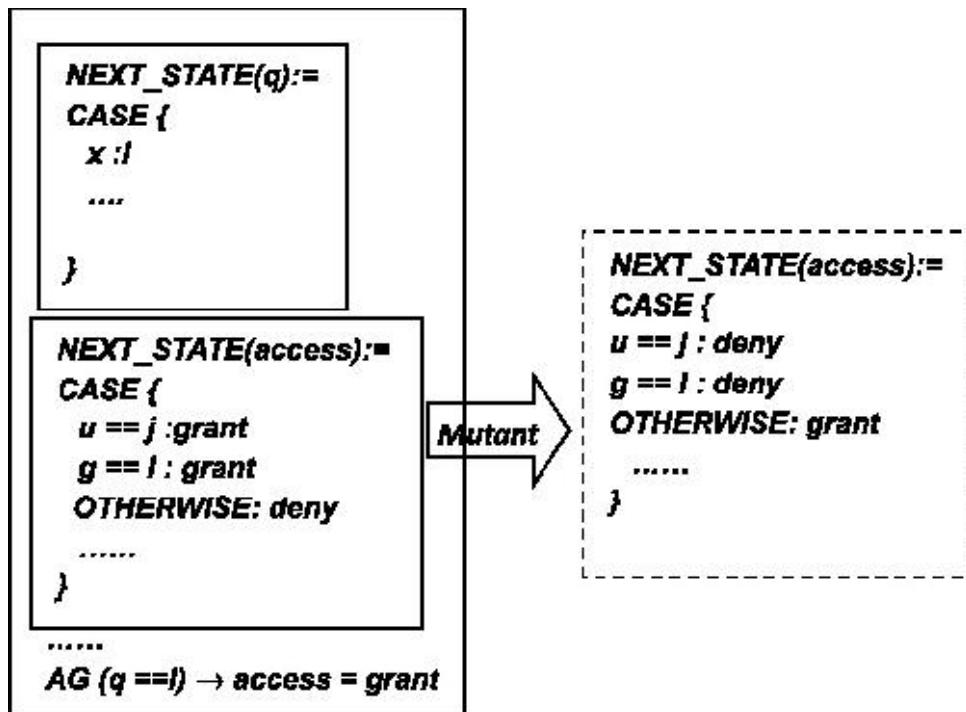


Figure 6.6 Example of uncovered rules in an ABAC model.

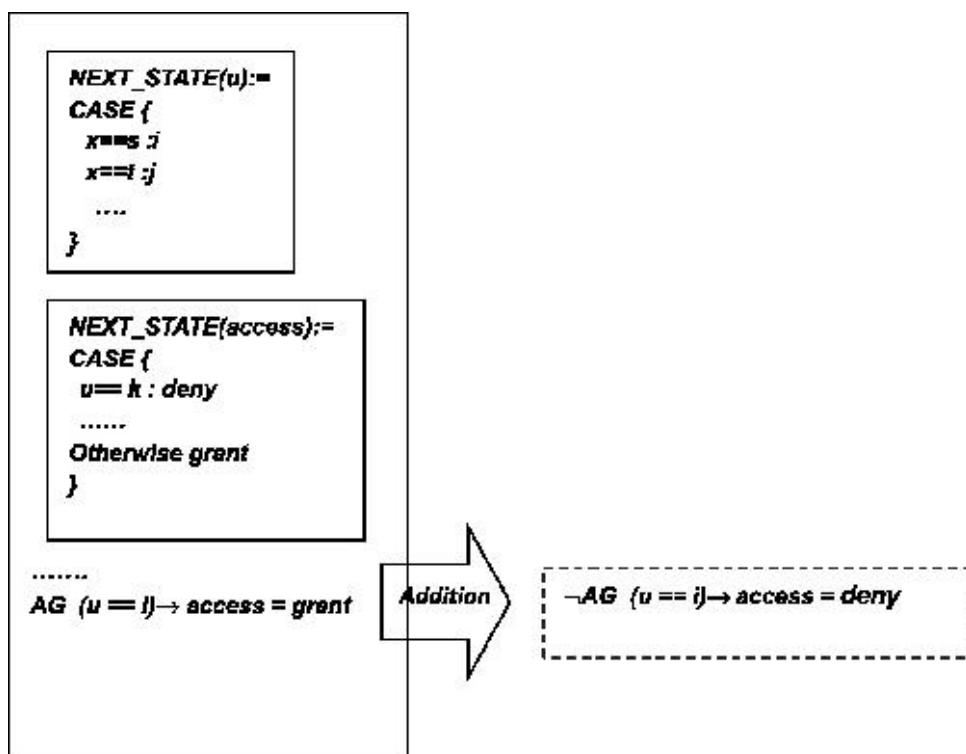


Figure 6.7 Unconfined rule in a property.

Note that it is possible the ABAC policy author intentionally allowed the exception for a safety requirement, and it is necessary to check every safety requirement against the set of rules in the model to achieve thorough verification.

6.4.4 Implementation Test

Black box model checking and white box mutation testing provide methods for verifying the correct model representation of the policy. Once a model is verified, the ABAC mechanism can be implemented based on the design of the model, and additional constraints if needed. Usually, ABAC mechanisms are code developed in a language the ABAC system supports, for example dedicated ABAC language such as XACML [6] is commonly used for access control code implementation. ABAC implementation can be error prone. As the ABAC model is directly implemented by an algorithm, errors are often caused by syntactic faults, such as mistakenly changing the + sign to – sign, or typing letter O instead of 0.

The correct implementation of the policy needs to be tested. To achieve that, a test oracle that contains cases of all possible outcomes of the access control safety requirement is required, because implementation faults are unpredictable without a logical trace for detecting. Thus, all the combinations of the variables in the safety requirements need to be covered in the oracle. For example, a safety requirement $x \text{ read } y \text{ grant}$, where x has three different values and y has five different values, will have $3 \times 2 \times 5 \times 2$ (assume that the actions has two values, *read* and *write*, and permission has only two values, *grant* and *deny*) test cases in the test oracle. The implemented ABAC system will then run these test cases to verify whether the actual test outputs are the same as the expected outputs.

It is not uncommon that a verification test includes hundreds of safety requirements; each contains tens of variables, in such case, the number of test cases in a test oracle for the implementation test is too great to be efficiently performed, therefore, additional techniques [7, 8] for reducing the test case size without sacrificing the capability may be required for the test.

6.5 Implementation Considerations

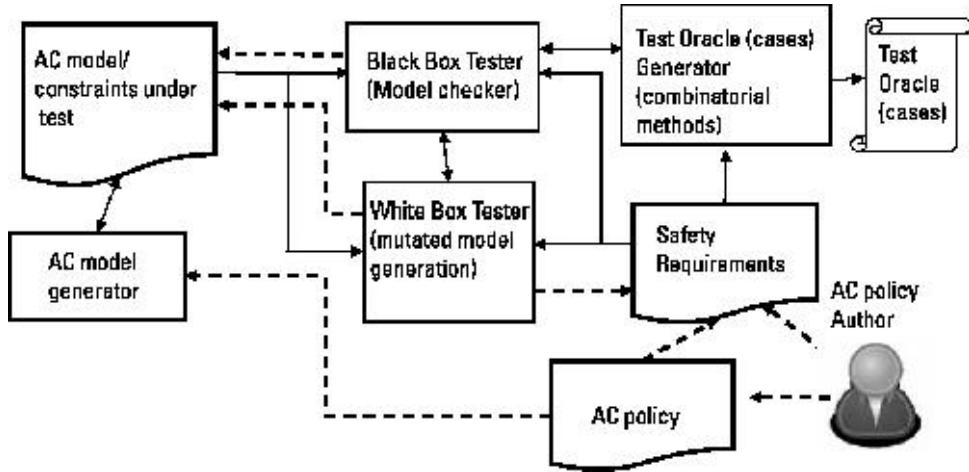


Figure 6.8 ABAC model verification framework.

The general ABAC system testing framework, as shown in [Figure 6.8](#), contains four major functions using the methods as stated in [Section 6.4](#). The ABAC rule real-time error detector is used optionally for designing the initial ABAC models [9]. The black box tester checks if a model (original or mutant) holds for the specified safety requirements. The black box tester (counterexample results) provides information for original model fix (a human action as dotted line in [Figure 6.8](#)) and for mutation killing check for white box tester, it also takes output from the test generator and returns results for test case generation. The white box tester generates and kills mutant models based on the original model and safety requirements; its mutated models are sent to the black box tester for mutation killing check, or to the ABAC model author for original model or safety requirement fix (a human action as shown in dotted lines in the figure). The test oracle generator generates test cases based on the safety requirement and the black box tester's counterexample results. The process steps are listed below:

1. (Optional) ABAC models are designed based on the ABAC policy by using ABAC rule real-time error checker [9].
2. Safety requirements are specified.
3. Completed original ABAC model is checked against safety requirements by black box tester, if an error is found, the original model needs to be fixed (thus repeat steps 1 to 3), otherwise proceed to next step 4.
4. Fixed original model send to white box tester for coverage and confinement check. The white box tester uses a black box tester to decide if generated mutant models were killed. If not, original model or safety requirements need to be fixed (thus repeat step 1 to 4), otherwise, proceeds to next step 5.

5. Test oracle generator generates test cases based on safety requirements, which are sent to black box tester for generating permission results used for test oracle.

Note that the components in [Figure 6.8](#) and steps are not necessarily all required for an ABAC model verification; the selections of components and steps might depend on the complexity of the model and the cost for implementing the test framework. Thus, an ABAC model test framework can contain optional components/ functions in [Figure 6.8](#) except that the black box tested is essential.

6.6 Verification Tools

In addition to the FSM based method, other techniques [10] are available for ABAC model verification such as theorem proof (including first and higher logic proof) and multiterminal binary decision diagrams (MTBDD) [11] methods.

6.6.1 Multiterminal Binary Decision Diagrams

Developed in racket (formal PLT) scheme, Margrave [12] is a software tool suite for verifying safety requirements against ABAC policies written in XACML. Margrave represents XACML policies as MTBDD models, and it allows the user to specify various forms of safety requirements in the scheme programming language. Margrave uses one variable for each attribute-value pair in the XACML policy. Margrave creates MTBDD models for the individual policy rules, then combines these with MTBDD-combining algorithms that implement the XACML rule- and policy-combining algorithms.

Margrave views the policy constants permit and deny as rules; an operation called augment-rule takes a Boolean condition on the variables and a rule, and constrains the rule to also require the given condition. It supports query-based verification and provides query-based views by computing exhaustive sets of scenarios that yield different results including change-impact analysis for comparing a pair of policies. Margrave provides the benefits of static verification without requiring authors to write formal properties; its power comes from choosing an appropriate policy model in first-order logic, and embracing both scenario-finding and multilevel policy-reasoning. In general, Margrave identifies formulas corresponding to

many common firewall-analysis problems automatically, thus providing exhaustive analysis for richer policies and queries.

6.6.2 ACPT

NIST's access control policy tool (ACPT) [13] provides (1) GUI templates for composing AC models, (2) safety requirements verification for ABAC models through a symbolic model verification (SMV) model checker, NuSMV, (3) complete test cases generated by NIST's combinatorial testing tool ACTS, and (4) XACML policy generation as output of verified model. Through the four major functions, ACPT performs all the syntactic and semantic verifications as well as the interface for composing and combining ABAC models for ABAC policies; ACPT is capable of verifying combined policies based on the permission priorities and/ or algorithms specified by the user.

ACPT allows users to specify access control models or their combinations, as well as safety requirements through GUI that contains model templates for three major access control policies: static ABAC, multileveled security, and stated work flow. ACPT then performs black box model check to verify if the specified safety requirements conform to the specified models. If not, non-conformance messages are returned to the user, otherwise, ACPT proceeds to generate test cases through ACTS, which are ready for testing the access control application implemented according to the models.

6.6.3 Formal Methods

Formal methods for the validation of access control policies involving mathematical tools and proofs have also been advocated. Rémi Delmas and Thomas Polacsek [14] have proposed a logical modeling framework to find the inconsistencies and incompleteness in access control policies. Providing a mechanism for the detection of these two properties, they have introduced two new properties, applicability and minimality and their proposed technique is capable verifying these two properties [15]. By using the concepts of signatures, formula and predicates, they have defined some rules for the logical framework, which works for limited or finite data so their rules are also applicable to finite data. They also mentioned that the many-sorted first order logic (MSFOL) [16] formula should be converted to a pseudo-Boolean logic formula to analyze it. The proposed tool is a three-steps procedure where grounding operation gives the grounded formula in

the first step which is converted to a bit-vector expression using the bit-vector encoding in the second step of this process. In the last step of this procedure, the bit-vector expressions are converted into clauses which are in pseudo-Boolean form and give us the pseudo-Boolean formula.

Z [17] is based on axiomatic set theory and first order predicate logic, which can be used for describing and modeling access control policies [18]. Z notation based on apply set theory forms an adequate basis for building the access control model, which allows syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving for model verification by domain checking. Many of the proof obligations are easily proven. In more difficult cases, generating the proof obligation is often a substantial aid in determining whether a specification in the access control model is meaningful to the access control policy.

6.7 Conclusion

This chapter describes a notion of safety for access control, and analyzes verification approaches for static, dynamic, and historical ABAC classes. Static classes are those in which no state is retained, while dynamic class may retain state during a session. Historical classes include long-term subject and object history in access decisions. An ABAC system is safe if no privilege can be escalated to unauthorized principals, but the correct privileges are always accessible to authorized principals. We also describe a rough taxonomy of faults that may be present in access control models.

To verify safety requirements for ABAC models, we provided a general approach that expresses ABAC models and access control safety requirements in the formal specification of a black box model or first order logic checkers for verification. Then the black box verifier verifies the specified models against the specified safety requirements. Most of the verification system supports static, dynamic, and historical ABAC classes. In addition to black box checking, white box checking methods make sure that the semantic coverage of the safety requirements also conforms to the intentions of the ABAC models authors. Finally, the generation of test cases to check the conformance of the models and their implementations is necessary.

ABAC model and safety requirements conformance verification of generic ABAC policies bring benefits to society in two aspects. First, it should lead to improved verification practices for testing and verifying ABAC models in improving ABAC system quality and security in general.

Second, innovations in new testing and verification algorithms and tools tend to propagate quickly across application or task domains where ABAC policies are used.

References

- [1] Hu, V. C., and K. Scarfone, “Guidelines for Access Control System Evaluation Metrics,” *NIST Interagency Report 800-7874*, Gaithersburg, MD, September 2012.
- [2] Hu, V. C., D. F. Ferraiolo, and D. R. Kuhn, “Assessment of Access Control Systems,” *NIST Interagency Report 7316*, Gaithersburg, MD, September 2006.
- [3] Gouglidis, A., I. Mavridis, and V. C. Hu, “Security Policy Verification for Multi-Domains in Cloud Systems,” *International Journal of Information Security (IJIS13)*, Vol. 13, No. 2, Springer Berlin Heidelberg, Germany; July 2014, pp. 97-111.
- [4] Harrison, M. A., W. L. Ruzzo, and J. D. Ullman, “Protection in Operating Systems,” *Communications of the Association for Computing Machinery*, Vol. 19, No. 8, August 1976.
- [5] Hu, V. C., et al., “Model Checking for Verification of Mandatory Access Control Models and Properties,” *Int. J. Soft. Eng. Knowl. Eng.*, Vol. 21, No. 1., February, 2011.
- [6] XACML, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- [7] <http://csrc.nist.gov/groups/SNS/acts/index.html>
- [8] Hu, V. C., R. D. Kuhn, and T. Xie, “Property Verification for Generic Access Control Models,” *Proc. 2008 IEEE/IFIP International Symposium on Trust, Security and Privacy for Pervasive Application (TSP2008)*, Shanghai, China, December 17-20, 2008.
- [9] Hu, V. C., and K. Scarfone, “Real-Time Access Control Rule Fault Detection Using a Simulated Logic Circuit,” *Proc. ASE/IEEE International Conference on Privacy, Security, Risk and Trust*, Washington D.C., September 8-14, 2013.
- [10] Li, A., et al., “Evaluating the Capability and Performance of Access Control Policy Verification Tools,” *Proc. IEEE Military Communications Conference (MILCOM 2015)*, Tampa FL, August 17-24, 2015.
- [11] Clarke, E., et al., “Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation,” *International Workshop on Logic Synthesis*, 1993.
- [12] Fisler, K., et al, “Verification and Change Impact Analysis of Access Control Policies”, *Proc. 27th International Conference on Software Engineering (ICSE'05)*, pp. 196-205, Association for Computing Machinery, New York, NY, 2005.

- [13] <http://csrc.nist.gov/groups/SNS/acpt/index.html>.
- [14] Abassi, R., and S. Fatmi, “An Automated Validation Method for Security Policies: The Firewall Case,” *The 4th Int. Conf. on Information Assurance and Security*, 2008, pp. 291-294.
- [15] Aqib, M., and R. A. Shaikh, “Analysis and Comparison of Access Control Policies Validation Mechanisms,” *I.J. Computer Network and Information Security*, 2015, Vol. 1, pp. 54-69.
- [16] Gallier, J. H., “MANY-SORTED FIRST-ORDER LOGIC,” *Logic for Computer Science: Foundations of Automatic Theorem Proving*, pp. 448–476, Wiley, 2003.
- [17] Potter, B., J. Sinclair, and D. Till, *An Introduction to Formal Specification and Z*, Second Edition, *Prentice Hall International Series in Computer Science*, 1996.
- [18] Hu, V. C., “The Policy Machine For Universal Access Control”, Dissertation, Computer Science Department, University of Idaho, 2002.

-
1. Some of the content in this chapter is derived from *NIST SP 800-192—Verification and Test Methods for Access Control Policies/Models*, by V. C. Hu, R. D. Kuhn, and D. Yaga, June 2017, and *NISTIR 7874—Guidelines for Access Control System Evaluation Metrics*, by V. C. Hu and K. Scarfone, September 2012.

7

Attribute Consideration

7.1 Introduction

Within ABAC, attributes are used to make critical access control (AC) decisions, so properties of attributes must be considered for users to have confidence in their use of ABAC. This chapter outlines factors influencing attributes that an ABAC system must address when engineering and evaluating attributes, and proposes some notional implementation suggestions for consideration. This chapter discusses considerations for attributes from the perspectives of fundamental security requirements: preparation, veracity, security, and readiness, as applied to ABAC.

In addition to these considerations, a general attribute framework with examples is demonstrated to show the importance and efficiency of the semantic and syntactic accuracies of attributes in federated ABAC environments, especially when natural language policies (NLP) are the initial policies. Finally, the discussed considerations are summarized to illustrate attribute evaluation scheme (AES) examples, which are applied to different ABAC requirements.

7.2 ABAC Attributes

ABAC systems using attributes are capable of enforcing a broad range of access control policies, including discretionary AC (DAC) and mandatory AC (MAC) concepts. Attributes (given by a name-value pair) contain characteristics of the subject, object, or environment conditions, which enables precise ABAC, allowing for a higher number of discrete inputs into an AC decision, and providing a bigger set of possible combinations of those variables to reflect a larger and more definitive set

of possible rules to express policies. In addition to the earlier work documented in NIST SP 800-162 [1] and OMB M-04-04 [2] that suggested attribute implementations applied to the subjects and objects within an ABAC system, general attribute considerations need to be addressed.

The terms of access control functions and attribute provider, with the following definitions are used throughout the chapter.

Access Control Functions (AFs) are functions for AC mechanisms or schemes. For example, the Extensible Access Control Markup Language (XACML) scheme architecture includes functions such as policy decision points (PDPs), policy enforcement points (PEPs), policy administration points (PAPs), and policy information points (PIPs), along with some logical components for handling the context or workflow of policy and attribute retrieval and assessment. AFs hosted in local or network systems (called local or remote AF respectively) must function together to provide access control decisions and policy enforcement.

An Attribute Provider (AP) is any person or system that provides subject, object (or resource), or environmental condition attributes to AFs, or other APs (in such case, the AP is called remote AP) regardless of transmission method. An AP may be the original authoritative source or act as an intermediary between the authoritative source and the AF by receiving information from an authoritative source and then re-packaging the attributes for delivery/routing to storage repositories of AF or AP. Attribute values may be human generated (e.g., an employee database), derived from formulas (e.g., a credit score), or system generated (e.g. environment conditions such as time, location, etc.).

Regardless of the source of attributes, an AF should ensure that the attributes obtained associated with the subjects, objects, or environmental conditions to which they apply are not only secure but also free of errors. Attribute trustworthiness proofing by the defined scheme from which organizations can make risk-based decisions is based on the confidence in attributes supplied by an AF, AP, or local attribute resource. [Figure 7.1](#) illustrates the scope of attributes used, including authentication, authorization, and attribute proofing functions. Note that the remote attributes are the attributes provisioned through remote networks.

7.3 Consideration Elements

ABAC relies upon the evaluation of attributes to not only define ABAC policy rules, but also to enforce the rules. Good, reliable, and up-to-date attribute data that supports appropriate, well-informed access decisions are essential. In addition, attributes provided by AF or AP need to be assured through the attribute proofing mechanisms that identify, define, and describe a set of criteria and standardized attribute metadata (the attribute about an attribute) that can be used by AF to help determine assurance in the attributes they are leveraging for access decisions.

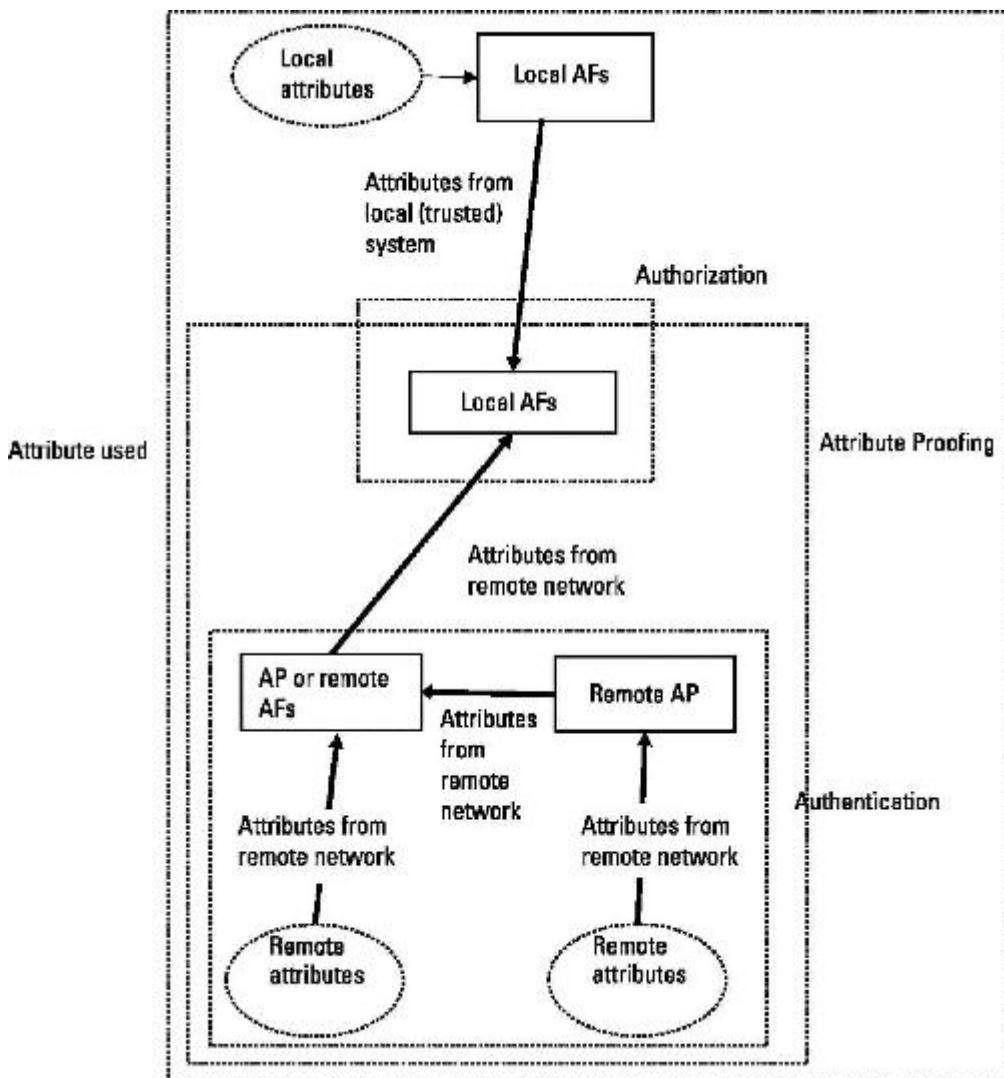


Figure 7.1 Scopes of attributes used: authorization, authentication, and attribute proofing of an ABAC system.

Once the authoritative sources define the appropriate attributes and allowable values, methods for provisioning attributes, appropriate attribute values, and a framework for communicating, storing, retrieving, updating, or revoking attributes need to be established. In addition,

interfaces and mechanisms must be developed or adopted to enable sharing of these attributes. Finally, an attribute evaluation scheme (AES), needs to be established to bring confidence based on the four principal areas of interest:

1. Preparation refers to the planning of the attribute creation and sharing mechanism, as well as rules for maintaining attributes' privacy between APs and AFs. This consideration should be based on the business operation requirements to meet the goals of efficiency and confidentiality of operations.
2. Veracity establishes the policy and technical underpinnings for semantic and syntactic correctness of subject, object, or environmental condition attributes, and ensures that the obtained attributes are trustworthy based on the agreed or trusted definitions, measurements, and maintenance processes of attributes.
3. Security considers different standards and protocols used for secure transmission and repositories of attributes between systems in order to avoid compromising the data integrity and confidentiality of the attributes. Security also prevents exposing vulnerabilities in APs, AFs, or entities or other types of nefarious actions performed by unauthorized entities.
4. Readiness ensures that the update and retrieval of attributes support the AFs in which the attributes are used, that is, the frequency of refresh for attributes that change regularly or over time. This capability also ensures that a recent set of attributes required for the protected resource in question are cached in the event that the most recently updated attributes from authoritative attribute sources or repositories cannot be accessed during an information system emergency (i.e., low bandwidth, denial of service). In addition, the fail-over and backup capability of attribute repositories need to be considered.

7.4 Preparation Consideration

Attributes shared across organizations should be assured as to how they are located, retrieved, published, validated, updated, modified, secured, and revoked. Consequently, all attributes must be defined and

constrained by allowable values required by the appropriate policies. The schema for these attributes and allowable attribute values must be published to all participants to help enable object owners with rule and relationship development. Attributes may be created and shared by multiple organizations so the design of an attribute framework must consider the federated usage, creation mechanism, and maintenance scheme according to the business and ABAC requirements. APs and AFs also need to maintain privacy to meet the confidentiality requirement. In addition, minimizing the number of attribute sources used in authorization decisions may improve performance and simplify the overall security management of the ABAC solution. Organizations planning to deploy an ABAC solution may benefit from establishing a close working relationship among all of the organization's stakeholders who will be involved in the attribute preparations.

7.4.1 Subject Attribute Preparation

Attribute authorities typically provision subject attributes for the type of attribute provided and managed through an AF or AP, except for non-person entities (NPE) such as autonomous services, or applications generated or controlled by operating systems. Usually there are multiple authorities, each with authority over different subject attributes. For example, security might be the authority for clearance attributes, while human resources might be the authority for name attributes. Subject attributes that require assured information sharing to allow subjects from one organization to access objects in another organization must be consistent, comparable, or mapped to allow equivalent policies to be enforced. For example, a member of organization *A* with the role job lead wants to access information in organization *B*, except organization *B* uses the term task lead to denote the equivalent role. [Table 7.1](#) shows an example of a subject's attributes.

As subject attributes may be provisioned by different authorities (human resources, security, organization leadership, etc.), methods of obtaining authoritative data need to be regulated. For example, only security authorities should be able to provision and assert clearance attributes and attribute values based on authoritative personnel clearance information; an individual should not be able to alter his or her own clearance attribute value or any other attribute values. Other subject

attributes may involve the subject's current tasking, physical location, and the device from which a request is sent; processes need to be developed to assess and assure the quality of such subject attribute data.

Table 7.1
Subject Attribute Example

| Subject | Attribute Name | Attribute Value | Policy Applied* |
|-------------|--|-----------------|---|
| Company ID | ID numbers (e.g. organization A) | | Classified and unclassified object access |
| Division | Division name (e.g. software development division) | | Classified and unclassified object access |
| Group | Group name (e.g. testing group) | | Classified and unclassified object access |
| Name | Person's name (e.g. Joe Smith) | | Classified and unclassified object access |
| Clearance | Clearance level (e.g. 1) | | Classified object access |
| Role | Role ID (e.g. job lead, (or task lead)) | | Classified object access |
| Training ID | Training label (e.g. minimum requirement) | | Classified object access |

* The Policy Applied column lists the type of policy rules which require this attribute for the evaluations of access permission if multiple policies are applied to the ABAC system.

In addition, authoritative subject attribute provisioning capabilities should be appropriately dependable for privacy, and service expectations. These expectations may be detailed in an attribute practice statement (APS) [3], which provides a listing of the attributes that will be used, and may identify authoritative attribute sources throughout the organization. Network infrastructure capabilities are required to share and replicate authoritative subject attribute data within and across APs and AFs.

7.4.2 Object Attribute Preparation

The data or resource owner/ custodian of AF or AP typically provisions object attributes upon object creation. For example, object attributes may be bound to the object or externally stored and referenced via a metadata

service and repository. While it may not be necessary to have a common set of object attributes in use across the enterprise, object attributes must be consistently employed within an individual system to fulfill ABAC policy requirements, and available sets of object attributes should be published for those wishing to mark, tag, or otherwise apply object attributes to their objects. At times, it might be necessary to ensure that object attributes are not tampered with or altered (i.e., remain static) to satisfy an access request. [Table 7.2](#) shows an example of an object's attributes.

Table 7.2
Object Attribute Example

| Object Attribute | | |
|-------------------------------|--|---|
| Name | Attribute Value | Policy Applied* |
| Object ID | ID numbers (e.g. 234567) | Classified and unclassified object access |
| Object owner | Name of object owner or organization (e.g. organization B) | Classified and unclassified object access |
| Object creation date and time | Date and time (e.g. May 26, 2015) | Classified and unclassified object access |
| Object deletion date and time | Date and time (e.g. May 26, 2017) | Classified and Unclassified object access |
| Classification | Classification level (e.g. 1) | Classified object access |
| Limited access ID | ID label (e.g. Public) | Classified object access |

*The Policy Applied column lists the type of policies which require this attribute for the evaluations of access permission if multiple policies are applied to the ABAC system.

AC authorities may not be able to appropriately and closely monitor all events. Frequently, object information is driven by non-security processes, and requirements as driven by business cases required by the consumer clientele in question; therefore, measures must be taken to ensure that object attributes are assigned and validated by processes that the object owner or administrator considers appropriate and authoritative

for the application. For example, object attributes must not be modifiable by the subject to manipulate the outcome of the AC decision. Objects can be cryptographically bound to their object attributes to identify whether objects or their corresponding attributes have been inappropriately modified. Mechanisms must be deployed to ensure that all objects created are assigned the appropriate set of object attributes to satisfy the policy used. It may be necessary to have an enterprise object attribute manager to coordinate these requirements. And the object attributes must be made available for retrieval by ABAC systems for AC decisions. Additional considerations for creating object attributes include:

- In general, users will not know the values of an object attribute (e.g., to which access to a sensitive compartment for a given user is granted). Data confidentiality of object attributes should be accounted for, so that authorized users only see the values that are applicable to them.
- As with subject attributes, a schema is required for object attributes defining attribute names and allowed values, to ensure object attributes are valid within its semantics and syntax definitions.
- Attributes need to remain consistent in policies that share the attributes.

There have been numerous efforts within the federal government and commercial industry to create object attribute tagging tools that provide not only data tagging, but also cryptographic binding of the attributes to the object. These capabilities also provide validation of the object attribute fields to satisfy AC decision requirements.

7.4.3 Environment Condition Preparation

Environment condition refers to context information that generally is not associated with any specific subject or object, but is required in the decision process. Environmental Attributes are different from subject and object attributes in that they are not administratively created and managed prior to run time, but instead are intrinsic and must be detectable by the AF while AF makes an appropriate access decision. The AF evaluates environment conditions such as the current date, time,

location, threat, and system status against current matching environment variables when authorizing an access request. Environment conditions drive ABAC policies to specify exceptional or dynamic rules that supersede rules driven by subject/object attributes only. When composing ABAC rules with environment conditions, it is important to ensure that the environment condition variables and their values are globally accessible, tamper-proof, and relevant for the environments where they are used.

7.4.4 Metadata

In the course of managing attributes, metadata is applied to subjects and objects as extended attribute information useful for enforcing fine-grained ABAC policies, incorporating information about the attributes, and managing the volumes of data required for enterprise attribute management. Metadata can also be used to assign an assurance level or measure of confidence as a composite score for attribute veracity, security, and readiness. Standardized attribute metadata are elements of information about each attribute. These elements include information about the attribute, for example, the value (e.g., how often it is updated), the processes used to create or establish the attribute (e.g., whether it is self-asserted, or retrieved from a record), and the source of the attribute itself (e.g., authoritative). Regardless of the ABAC mechanism, establishing a score system for an attribute's metadata elements can support access decisions. The decision to use specific attributes from remote AFs or APs could then be made based on individual attribute confidence scores.

[Table 7.3](#) shows an example of standard (agreed-upon) metadata for sharing provenance information, such as attribute source. The specific attribute value person may be sufficient for accessing data for an unclassified information request, but insufficient for access to a sensitive system since the metadata level of confidence is self-reported and not drawn from an authoritative source.

[Table 7.4](#) shows example criteria of attribute preparation considerations. Note that attribute includes metadata if applicable.

[Section 7.8](#) demonstrates a general attribute framework for integrating and defining attributes for the planning of attributes by using metadata. The example shows an ABAC system initially started from

natural language policy (NLP), which governs multiple AC systems in an enterprise environment.

Table 7.3

Example of Standard Attribute Name and Value for Attribute Source Metadata

| Standard Attribute Name Entity applicability | Standard Attribute Value Person |
|--|--|
| Name | Joe Smith |
| Classification | Unclassified |
| Level of confidence | 1 (self-reported) |
| Assurance detail—refresh | Pulled |
| Assurance detail—Last updated | 3/8/2015 |
| Attribute source | USAJOBS.gov |

Table 7.4

Example Consideration for Attribute Preparation Criteria

| Consideration Criteria | Applied Attributes |
|-----------------------------|--|
| Attribute coverage | Attributes cover all protection policy requirements of the organization (Semantically complete). |
| Attribute policy | Attributes creation, update, and revoking policies and standard procedures are well defined. |
| Attribute management method | Attributes are under federated or unified governance. |

7.5 Veracity Consideration

Except for NPE, the veracity of an asserted attribute is affected by the care the AF or AP takes in obtaining, evaluating, and maintaining the value while in its possession. Two characteristics that influence Veracity include:

- Attribute trustworthiness
- Attribute accuracy

7.5.1 Attribute Trustworthiness

Attribute trustworthiness considers how well the sources of attributes are authenticated, identified and validated. This applies to the attribute source from the remote AP or AF. In attribute trustworthiness, there is a distinction between truthfulness on the attribute's value and authoritativeness of information. However, the focus needs to be on AF or AP's trust (for example, their credentials or federation relations) so that the attributes represent the underlying subject, object, or environment condition. For example, a consideration is that the attribute of a specific credit score may be strongly disagreeable, but the attribute user may trust that it did come from a specific credit reporting agency. [Table 7.5](#) shows an example of attribute trustworthiness based upon four different level of confidence.

Table 7.5
Attribute Trustworthiness Examples

| Low Based On | Medium Based On | High Based On |
|---------------------------|---|---|
| Self-reported | Medium attribute proofing (mostly for subjects) | Derived attributes (independent of underlying factors – original source) |
| Third-party public source | Authenticated source | High identity proofing (mostly for subjects) Authenticated source with service level agreements (SLAs) |

Attribute trustworthiness proofing relies on a schema by which organizations can make risk-based decisions based on the trust in attributes supplied by remote AFs or APs. Approaches to achieving this purpose include:

- Identify, define, and describe a set of standardized attribute metadata that can be used by AFs to help determine confidence in the attributes they are leveraging for authorization decisions.
- Identify, define, and describe a set of criteria that can be used to determine the trustworthiness of attributes (e.g. shown in [Table 7.5](#)), which may include a scoring system mechanism to determine an objective confidence level for a given attribute.

- Develop suggested performance guidelines and specifications for remote AF or AP operations, based on an organization's risk tolerance.

For remote subject attributes (not from local AFs or NPE), the attribute assurance relies on the chain of trust used to determine and report on the attributes. If the remote AF or AP reporting the attributes did not verify them, then it is necessary to provide a chain of evidence that shows that the attributes were authoritatively verified, and the trust of the verifications, and their association with the system to which they pertain, has been maintained.

7.5.2 Attribute Value Accuracy

Given the broad spectrum of entities that will interoperate with each other, synonyms and homonyms of attribute definitions are inevitable, thus interoperability standards and protocols that all entities agree to are essential to enabling cooperation. Agreed-upon standards in both syntactic and semantic attribute values must be developed to ensure successful interoperation of systems. For example, a consideration is that the attribute user may be assured that it did come from a trusted credit reporting agency, but the attribute value of a specific credit score may be strongly disagreeable. Thus, dictionaries with standardized syntax and semantics for attribute namespaces need to be agreed on and published by the AFs or APs.

Attribute value inaccuracy can be due to different data types (integer, string, Boolean, etc.), or different units of measurement (pounds, kilogram, etc.) between AFs and APs. Thus, agreement, federated mitigation, or interpretation/conversion may need to be performed, such that the attribute value is accurate for the policy evaluation. For example, attribute values need to be accurately assigned to the subjects, which are associated with the organization's business functions. Unless the AF or AP is responsible for the standard, algorithm, or protocol in generating the attribute values, the accuracy of a ttribute value is usually evaluated with the attribute trust as described in 7.5.1. [Table 7.6](#) shows examples of consideration of attribute veracity criteria.

[Section 7.8](#) demonstrates an example of a general attribute framework that was initially based on the natural language policy (NLP).

The framework achieves the attribute veracity by applying metadata to integrate and define attributes.

7.6 Security Consideration

Ensuring the security of an attribute's value and its metadata and keeping it free from tampering or corruption are required for a higher level of assurance. AFs and APs should also adequately verify stored attribute information and provide a high level of protection within its enclave. In addition, attribute security also determines how securely the AF or AP supplies attributes to an AF. In other words, how does the AF or AP assure that the attribute that it intends to send is the attribute the AF actually will receive? Attribute security includes evaluating security for both attribute-at-rest and attribute-in-transit conditions. For example, to improve the security of attribute transmission, attributes can be sent via an encrypted and signed mechanism (e.g., through more secure mechanisms, such as signed Security Assertion Markup Language (SAML) [4] assertion, TLS[5]). Therefore, the AES may include methods for nonrepudiation of attribute transmission.

Table7.6
Example Consideration for Attribute Veracity Criteria

| Consideration Criteria | Applied Attributes |
|------------------------|--|
| Verification | Attributes are properly verified for veracity through provision and management. |
| Standard applied | Documented rule or standards for attribute value assignment and definition (syntax and semantic rule). |
| Trust criteria | Criteria that can be used to determine the trustworthiness of attributes. |
| Remote AF/AP guideline | Performance guidelines and specifications for remote AF or AP. |

7.6.1 Attribute-at-Rest

Attribute-at-rest security evaluates the mechanism for the actual attribute store and how well the AF and AP protect the information or attribute-generation processes in the attribute store. Note that attribute-at-rest security ensures the generation and management of an attribute and its value while the attribute value consideration, as described in [Section 7.5.2](#), focuses on the semantic accuracy of attribute values. Factors or capabilities that need to be evaluated include:

- Employ encryption;
- Take measures to detect unintended alteration of attribute values;
- Set data stores on a network behind a proper defense in depth posture;
- Enforce policies on attribute update, copy, revoke, or modify process;
- Logged and audited change of attribute.

The attribute-at-rest factors or capabilities are commonly used to evaluate the local AF because the required information can be rendered locally. However, for AP, remote AF, or remote AP, without local access to the involved systems, a checklist for the evaluation of the factors or capabilities might be required.

7.6.2 Attribute-in-Transit

Attribute-in-transit security evaluates how securely the attribute is transmitted to the AP or AF. Factors or capabilities that need to be evaluated include:

- Security protocols are used for transmitting both attribute requests and attribute values to the AP or AF, for example, transmitting in the clear without encryption versus PKI-enabled TLS sessions.
- Replay attack protection is usually accomplished by including information provided by the AF into the signed message that is provided by the remote AF or AP. This guarantees integrity and confidentiality of the attribute.

Attribute-in-transit applied in a second-tier receipt of attributes, that is, when attributes are sent by remote AF or AP, needs to be considered

such that the assurance token can be passed through the chain of forwarding route. For example, for higher levels of assurance, using digitally signed attributes (crypto-binding) provides a hash of the attribute so that AF can be assured that an attribute was not altered or tampered with before it is received.

In addition to the AF and AP's transmission security, the security arrangements between AFs need to be considered, because in order to make a correct policy decision, the transition of attributes between AFs should be protected from changing by any other processes.

If applicable, a set of consideration elements or schemes (e.g., SAML) should be identified that can be used by ABAC systems to help determine the attribute's security as shown example considerations for security criteria in [Table 7.7](#).

7.7 Readiness Consideration

ABAC AFs need information on how often an attribute's value is pulled or obtained, as well as how securely the attribute's value is processed when it is needed. The consideration of the attribute quality in terms of readiness should include capabilities of refresh, synchronization, cache, backup, and log of attributes.

7.7.1 Refresh

Refresh considers how attribute values are updated or validated—refreshed against ground truth by the AF or AP. Proactive acquisition needs to be considered for the impact of a refresh rate on a specific attribute. We need to know, for example, whether the information is being pushed from another source to the AF or AP or pulled on a schedule proactively. Attribute values on a schedule or on demand give assurance of how current and, therefore, how applicable the attribute value may be.

Table 7.7
Example Consideration for Attribute Security Criteria

| Consideration | Criteria | Applied Attributes |
|----------------------------|---|------------------------------|
| Repository security | Secure or trusted attribute repository (e.g. dedicated or shared attribute repositories). | Subject, object, environment |
| Communication security | Secure communication between AFs, and APs (e.g. encrypted) | Subject, object, environment |
| Process integrity | Transition of attributes between AFs should be protected from changing by any functions. | Subject, object |
| Non-repudiation capability | Methods for non-repudiation of attribute transmission. | Subject, object, |
| Attribute change policy | Formal rules or policy (or standards) to create, update, modify, and delete attributes. | Subject, object |

7.7.2 Synchronization

Synchronization of attribute transmission sequences between AFs needs to be coordinated based on the sequence of the ABAC system's processing scheme or protocol such that the updates of attributes and their value will not result in faulty AC decisions. For example, to keep AF functions in sync in the XACML [6] scheme, updating attributes by PAP should not be allowed while an authorization process is in progress, which means updated or new added attributes should be available after PEP finishes its process.

7.7.3 Cache

Readiness also ensures that a recent set of attributes required for the protected resource in question are cached, in the event that the most updated attributes from authoritative attribute sources or repositories cannot be accessed during an information system emergency (i.e., low bandwidth, denial of service [DoS]). In addition, the failure recovery capability of attribute repositories needs to be considered.

7.7.4 Backup

As attributes are the critical components for an organization's ABAC system, they need to be always available as long as the system is functional. Thus, readiness needs to include the capabilities of fail-over and backup of attributes from the failures of attribute repositories or transmission systems.

7.7.5 Log

For higher security requirements, an organization might require all the activities, including changes (creation, modification, and deletion) and use of attributes, to be logged for later investigation if necessary.

If applicable, identify, define, and describe a set of consideration elements that can be used to help determine the attribute readiness, as shown in the attribute readiness criteria example in [Table 7.8](#).

7.8 An Example of a General Attribute Framework

Preparation and veracity of attributes are especially crucial when applying ABAC to a multi-host environment, such as an enterprise system, where attributes are created and managed by diverse organization units. The attributes are used both for local (organization unit) and global (enterprise) ABAC policies, thus, a mechanism is required to mitigate the syntactic and semantic differences of attributes. An example is the general attribute framework (GAF) that allows attributes to be defined with syntactic and semantic accuracy across federated/ networked systems under the enterprise ABAC domain, where initial AC policies are in natural language (NL) without formal attribute definitions. The following demonstrates the use of GAF for attribute accuracy.

Table 7.8
Example Consideration for Freshness Metadata Criteria

| Consideration Criteria | Applied Attributes |
|-----------------------------|---|
| Attribute refresh frequency | Attribute refresh frequency meets the system performance requirement. |
| Attribute caching | Attribute caching during run time meets the system performance requirement and protocols between AFs. |
| Attribute process sequence | Attribute transition between AFs are coordinated without generating errors. |
| Backup capability | Fail-over or back up attributes support. |
| Access log | Log for attribute changes and access. |

To enforce ABAC policies across the enterprise, the policies need to be in a machine-readable format processed by the computer that performs ABAC of the information system (i.e., decision engine). However, most initial ABAC policies originate in NL that cannot be ingested and processed by the decision engine. Thus, it is necessary to translate the natural language policies (NLP) into machine-readable policy rules; a general approach is to have a resource domain (e.g. laws/statutes for privacy policies) expert examine the system's subject attributes, and map the access privileges to the system's objects according to the policy applied. This work is painstaking and costly because it requires the resource domain experts to comprehend not only the policy rules but also the meanings of the system's subject and object attributes. And after the completion of the work, resource domain experts will again be needed when the policy or the system is updated. Since each system requires the resource domain expert's effort to translate the policy from its local attribute definitions of the information system, and there are an unlimited number of the information systems, the total cost of the administrative overhead may be unmanageable.

This problem also applies to mapping between an enterprise attribute schema and an application-specific schema, particularly ones built before

the enterprise schema is defined and/ or commercial off-the-shelf (COTS) products that come with their own built-in schema, as in the ones usually established for legacy information systems. For attribute accuracy, organizations must normalize subject attribute names and values, or maintain a mapping of equivalent terms for all organizations, and this should be managed by a central authority.

It is important to devise a portable framework that is general enough to be used by AC administrators to compose their ABAC policies without the extra cost of translating or learning resource domain knowledge. Thus, a general attribute framework (GAF) should be constructed from the content and ontology of the intended policy using generic attributes (GA), which can be applied to the specific attributes of any information system in different application domains. The National Identity Exchange Federation (NIEF) attribute registry is a collection of attribute definitions that are intended for use by organizations and communities that wish to implement federated identity and privilege management technologies within the context of the NIEF. Each attribute definition listed there has been developed with the intent to enable organizations to exchange attribute data in a manner that permits machine parsing and understandability [7]. [Figure 7.2](#) shows the relations of the resource domain policy and the machine-readable policy for each individual system. [Figure 7.2\(a\)](#) shows a nonportable view of ABAC systems without GAF which require law/statute experts to map a law/statute to ABAC policies for every system, compared to the portable view of [Figure 7.2\(b\)](#), where the ABAC systems using GAF only need one law/statute expert to do the mapping.

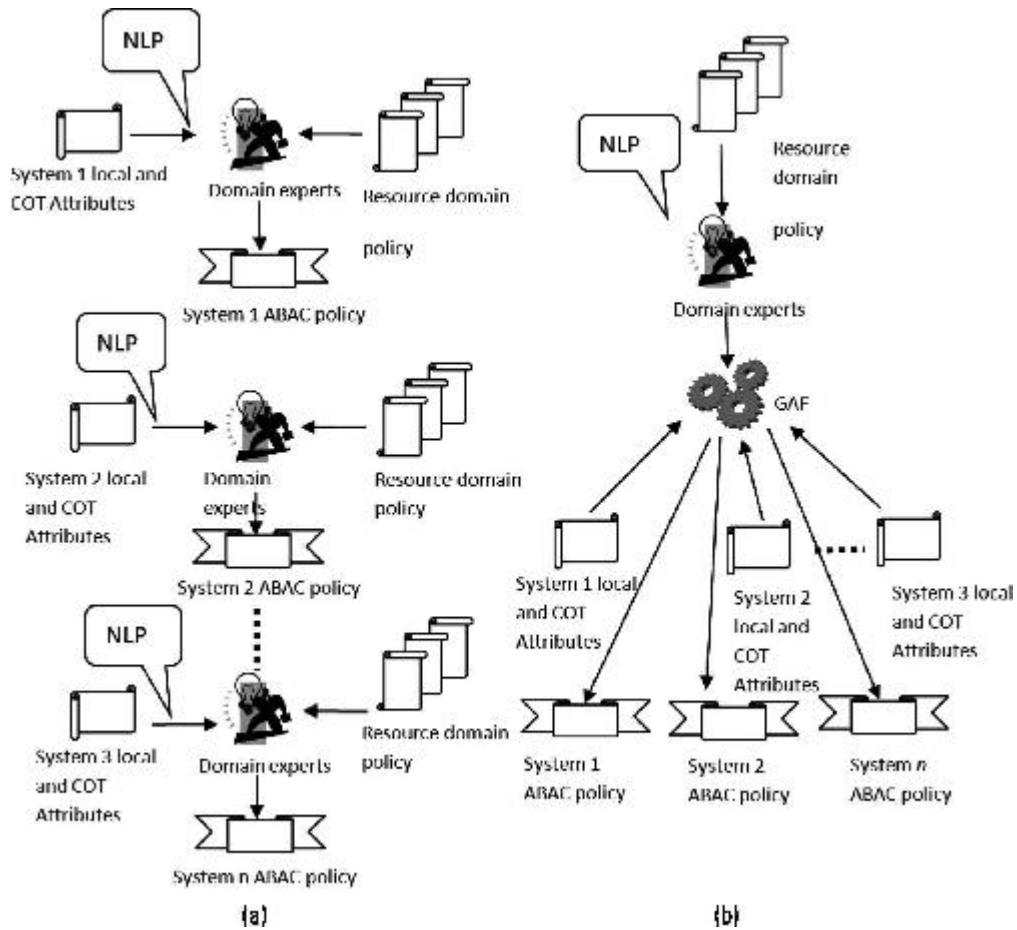


Figure 7.2 Producing ABAC policies (a) without and (b) with general attribute framework (GAF).

The goal of the GAF is to provide a framework as a layer between NLP policy and machine-readable policies/ rules. GAF allows ABAC policy authors to compose policies without the resource domain expert knowledge of the policy related to the object, which is required only at the time when the framework is constructed. Derived from analyzing the content and ontology of the policy rules, the GAF contains access rules associated with the subject and object GAs, which are generic for any domain of an ABAC system. Therefore, a GAF is an ABAC policy with rules in terms of GA based on access control elements: subject/object attributes, environment conditions, and actions. The format of a GAF access control rule is:

```

    IF <subject CA1> ..... AND/OR<subject CAn> AND <environment
    condition 1>....AND/OR <environment condition n>THEN ALLOW <ac-
    tion1> ..... AND <actionn> ACCESS TO OBJECT WITH <object GA1>
    ..... AND/OR <object GAn>
  
```

GAF will provide clear definitions and descriptions of the GAs by using a common vocabulary such that any ABAC policy administrator who does not have the resource domain knowledge can understand. To enforce the policy on the information system, the ABAC policy administrator only needs to assign the GAF's GAs as tags or metadata to the subjects and objects by reviewing the existing subject and object attributes from the system. There is no need to create policy rules, which are already embedded in the GAF.

Figure 7.3 lists part of the original text of privacy rules from the OMB 6-16 and OMB 7-16 statutes [8, 9].

"Implement protections for remote access to personal identifiable information" (Step4)

"Implement NIST Special Publication 800-53 security controls requiring authenticated, virtual private network (VPN) connection" (Step 4.1)

"Implement NIST Special Publication 800-53 security controls enforcing allowed downloading of personally identifiable information" (Step 4.2)

--OMB6-16

Attachment 1 Safeguarding Against the Breach of Personally Identifiable Information, Section C Security Requirement, Item: Control Remote Access: "Allow remote access only with two-factor authentication where one of the factor is provided by a device separate from the computer gaining access".

--OMB6-17

Figure 7.3 Original text of privacy rules from OMB 6-16 and OMB 7-16.

Figure 7.4 shows a GAF containing a list of common GAs in columns for privacy statutes as well as five sample OMB rules in Figure 7.3. The computer column contains the environment condition, the subject attributes column contains the GAs for the subjects, the action attributes column contains the available actions, and the object attributes column contains the GAs for the object, while the audit column lists the actions that need to be performed after an access is granted. For example, the first rule in Figure 7.4 says:

“A subject whose login is remote has federal agencies and 2-factor (level 3) GAs, and is therefore permitted to read resource with PII GA.”

Note that the computer column contains the common GAs that are shared by the subject and object, and the audit column contains the

obligation required after the access action is performed.

The following examples demonstrate the mapping to concrete instances of the OMB7-16 privacy rule GAF (Figure 7.4): Example 1 is for an information sharing center (ISC), in which the local subject and object attributes are assigned based on ISC's data formats. Example 2 is for a federal organization, in which the subject and object attributes are from the human resource department (HRD) of the organization. These two examples show the portability property of a GAF for information systems with different domains. In Tables 7.9 and 7.10, the GAs row is the GAs from the GAF, and the local attributes row is the example of system attributes, which needed to be reviewed to decide the qualification (yes or no) for the mapped GAs. The GAF AC rule for the OMB7-16 rule is composed by “AND” all the GAs in the row.

| <i>Rules</i> | <i>Computer</i> | <i>Subject Attributes/Values</i> | <i>Actions</i> | <i>Resource Attributes/Values</i> | <i>Audit</i> |
|-----------------|-----------------|--|-------------------------|--|--|
| <i>OMB 6-16</i> | Remote User | Employer = Federal Agencies Authentication Level = 2-factor (Level 3) | Permitted to Read | Data Tags = PII | |
| <i>OMB 6-16</i> | All | Employer = Federal Agencies | Permitted to Read/Write | Special Characteristics = Sensitive Data | Action (Audit) = All Data Data Extracts = verify each extract including sensitive data has been erased within 90 days of its use still required |
| <i>OMB 7-16</i> | All | Employer = Federal Agencies | Permitted to Read/Write | Data Tags = SSN | Write (Collect) = Minimum needed for agency function |
| <i>OMB 7-16</i> | All | Employer = Federal Agencies | Permitted to Read/Write | Data Tags = PII | Write (Change) – Corrections or notations agency Justifications Write (Collect) = Minimum needed for agency function |

Figure 7.4 Example rules from OMB 6-16 and OMB 7-16.

Table 7.9
Mapping of GAs of an OMB7-16 Rule to an ISC System

| Attributes | Subject Attributes | | | Actions | Object Attributes | |
|------------------|--------------------|------------------|--------------------|---------|-------------------|-----------------------------|
| GAs | Remote user | Federal agencies | 2-factor (level 3) | Action | PII | PII |
| Local attributes | <remote login ID> | Federation ID | Electronic ID | Read | Vehicle year | Vehicle registration number |

Table 7.10
Mapping of GAs of an OMB7-16 Rule to a HRD System of a Federal Organization

| Attributes | Subject Attributes | | | Actions | Object Attributes |
|------------------|--------------------|------------------|--------------------|---------|-------------------|
| GAs | Remote user | Federal agencies | 2-factor (level 3) | Action | PII |
| Local attributes | <remote login ID> | Agency HRD ID | Remote access key | Read | SSN |

Example 1:

Grant read access for the user who has the attributes: remote user, federal agencies, and 2-factor (level 3) to the resource data with the PII attributes.

Thus, the following AC rule of the ISC can be achieved through the GAF:

Example 2:

Grant read access for the user who is remote login ID, and has federation ID, and electronic ID to the resource data with the vehicle year and vehicle registration number attribute.

Similarly, the following policy rule of the HRD can be achieved through the GAF:

Example 3:

Grant read access for the user who is remote login ID, and has HRD ID and remote access key to the resource data with the SSN attribute.

Note that to ensure the robustness of the GAF, the ontologies between the GAs may be expanded, as they pertain to identified subrules or hierarchical relations of rules. Also, assertion-based policy rules appear in some policies, and the handling of these features need to be addressed in the development of GAF.

7.9 Attribute Evaluation Scheme

Attribute evaluation scheme (AES) should be determined by the requirements and capabilities of an organization through the considerations of risk, performance, and cost. This section does not intend to construct a universal AES that suits all business requirements and capabilities. Instead, it provides mapping examples of AES metrics for ABAC systems, which can serve as prototypes that may be adapted to meet the specific needs of an organization when defining its AES.

7.9.1 AES Examples

[Table 7.11](#) illustrates an example of AES categorization based on the considerations from previous discussions. Note that consideration with AES may be different between systems or organizations depending on the security requirements; therefore they need to be assigned in line with the organization's operation and performance requirements, as well as incorporated when relying on federated systems. Differences in levels between AESs need to be considered for access decisions, for example, if an access decision uses two attributes, and one is low AES but the other is high.

Table 7.11
Example of AES for Attributes That Are Provisioned by Remote AFs or APs

| AES | Preparation | Veracity | Security | Readiness |
|---------|--|--|--|--|
| Level 1 | Attributes cover all protection policy requirements of the organization (semantically complete). | Attributes are properly verified for veracity through provision and management. | Secure attribute repository. Secure communication between APs and AFs. | Attribute refresh frequency meets the system performance requirement. Log for attribute changes and access. |
| Level 2 | Includes level 1. Attributes creation, update, and revoking policies and standard procedures are defined and documented. | Includes level 1. Documented rule or standards for attribute value assignment and definition (syntax and semantic rule). | Includes level 1. Dedicated attribute repositories. | Includes level 1. Attribute caching during run time meets the system performance requirement. |
| Level 3 | Includes level 2. Attributes are under federated or unified governance. | Includes level 2. Criteria that can be used to determine the trustworthiness of attributes. | Includes level 2. Encrypted attribute values and communications between APs and AFs systems. Methods for non-repudiation of attribute transmission | Includes level 2. Fail-over or back up attributes support. |
| Level 4 | N/A | Includes level 3. Performance guidelines and specifications for remote AF or AP. | Includes level 3. Transition of attributes between AFs should be protected from changing by any functions. | Includes level 3. Formal rules or policy (or standards) for logging the creation, updating, modifying, and deleting of attributes. |

Note that as the characteristics of the three attribute types (i.e., subject, object, and environment condition) vary in different operational environments, their AESs may be assigned by different criteria for each of the three types of attributes. This allows the flexibility by compositing sets of AESs that are practical for assurance measurements. For example, AES in Table 7.11 can be applied to an organization, whose attributes may be supplied by remote AFs or APs from outside the organization is different from organizations which do not obtain attributes from outside. Thus, a less restrictive consideration mapping of AES mapping is appropriate as illustrated in Table 7.12.

7.9.2 Attribute Practice Statement

Confidence in remote AFs or APs is gained by evaluating how secure the remote AF or AP's internal processes and procedures are with respect to both intentional attacks and unintentional errors or failures. It is often established on unverified assertions of validity that are not based on commonly agreed-upon standards. An example document that governs the effect of operations on AES is the attribute practice statement (APS) developed by the Identity Ecosystem Steering Group (www.idecosystem.org). APS is based on the Internet Engineering Task Force (IETF) reference document RFC 3647, *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework* [3], and includes additional points that would apply to remote AF, or AP operations. APS could be used for establishing the AES of veracity. The act of developing an auditable APS will provide an impartial assessment of the remote AF or AP's standards of operation and the confidence of the provided attribute. Thus, a higher AES level could be an APS that is audited for compliance with policy. Lower levels of AES could apply to remote AFs or APs that self-report adherence to policy or which do not publish their operation's practices.

7.10 Conclusion

ABAC controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment relevant to an access request. ABAC relies upon the evaluation of the attributes and a formal relationship or access control

rule defining the allowable operations for subject/ object attribute combinations. This chapter discusses considerations for attributes from the perspectives of fundamental security properties: preparation, veracity, security, and readiness as applied to ABAC.

Table 7.12
Example of AES Consideration for Object Attributes Not Provisioned by Remote AF or AP

| AES | Preparation | Veracity | Security | Readiness |
|---------|--|--|--|---|
| Level 1 | Attributes cover all protection policy requirements of the organization (semantically complete). | Attributes are properly verified for veracity through provision and management. | Secure attribute repository. | Attribute refresh frequency meets the system performance requirement. Log for attribute changes and access. |
| Level 2 | Includes level 1. Attributes creation, update, and revoking policies and standard procedures are defined and documented. | Includes level 1. Documented rule or standards for attribute value assignment and definition (syntax and semantic rule). | Includes level 1. Dedicated attribute repositories. | Includes level 1. Attribute caching during run time meets the system performance requirement. |
| Level 3 | N/A | N/A | Includes level 2. Transition of attributes between AFs should be protected from changing by any functions. | Includes level 2. Fail-over or back up attributes support. Formal rules or policy (or standards) for logging the creation, updating, modifying, and deleting of attributes. |

In addition to these considerations, a general attribute framework, with examples, is demonstrated to show the importance and efficiency of the semantic and syntactic accuracies of attributes in federated ABAC environments, especially when NLPs are the initial policies. In the end, the discussed considerations are summarized to illustrate AES examples, which are applied to different security requirements. Clearly, AES

framework development requires additional research and stakeholder outreach of the organizations that an ABAC is managing.

References

- [1] NIST Special Publication 800-162, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, January 2014, <http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf>.
- [2] OMB M-04-04, *E-Authentication Guidance for Federal Agencies*, December 16, 2003.
- [3] Chokhani, S., et al., *Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*, IETF, 2003, <https://www.ietf.org/rfc/rfc3647.txt>.
- [4] Organization for the Advancement of Structured Information Standards, *OASIS Security Services (SAML) TC*, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.
- [5] IETF, *The Transport Layer Security (TLS) Protocol*, <https://tools.ietf.org/html/rfc5>.
- [6] Organization for the Advancement of Structured Information Standards, *OASIS eXtensible Access Control Markup Language (XACML) TC*, web page: <http://www.oasis-open.org/committees/xacml/>
- [7] <https://nief.gfipm.net/attribute-registry/index.html>
- [8] Office of Management and Budget, *OMB6-16*, “Protection of Sensitive Agency Information”, June 23, 2006, <http://www.whitehouse.gov/OMB/memoranda/fy2006/m06-16.pdf>.
- [9] Office of Management and Budget, *OMB7-16i*, “Safeguarding Against and Responding to the Breach of Personally Identifiable Information”, May 22, 2007, <http://www.whitehouse.gov/OMB/memoranda/fy2007/m07-16.pdf>.

8

Deployments in Application Architectures

8.1 Introduction

ABAC has been used in a variety of application architectures, as functions involving attributes can provide the requisite granularity, flexibility, and scalability for access control configuration (model definition, access control data repository creation, reliable attribute sources, etc.) needed for protecting data and transactions in these environments. In this chapter we discuss ABAC deployments in distributed systems—Big Data, Web services, and workflow processes.

8.2 ABAC for Distributed Systems¹

Privacy and security controls are more likely to be compromised due to the misconfiguration of access control policies rather than the failure of cryptographic primitives or protocols [1] in distributed application architecture. This problem becomes increasingly severe as distributed systems (DS) become more and more complex, such as Cloud, Grid, and BigData processing systems, which are deployed to manage a large amount of sensitive information and resources organized into a sophisticated, distributed processing cluster. Basically, DS access control requires the collaboration among cooperating processing domains to be protected as computing environments that consist of computing units under distributed access control [2]. Many DS architecture designs were proposed to address information availability challenges; however, most of them were focused on the processing capabilities. Considerations for security in protecting DS are mostly ad hoc and patch efforts. Even with some inclusion of security in recent DS systems, a critical security

component, like authorization, for protecting DS processing components and their users from the insider attacks, remains elusive.

Reference [3] estimates that the global data population will reach 44 zettabytes (1 billion terabytes) by 2020. This growth trend is influencing the way data is being mass collected and produced for high-performance computing or operations and planning analysis. DSs are constructed for large and/ or disperse data that is difficult to process by using a single data processing unit, for example, to analyze Internet data traffic, synchronize results from parallel computing, or host variety and vast amount of user accesses. BigData, Cloud, Grid, and even IoT applying DS technology is gradually reshaping current data systems and practices. In addition, IT experts are just as keen on harnessing the power of DS to boost security, prevent fraud, enhance service delivery, and improve emergency response.

DS aims to answer IT infrastructures issues with scaling difficulties such as capabilities for data storage, advance analysis, and shared data services. Therefore, distributed data processing systems must be able to deal with collecting, analyzing, distributing, and securing data that requires cooperatively processing diverse data sets that defy non-DS technologies. To maximize scalability and performance, some distributed processing systems might apply massively parallel software running on many commodity computers in distributed frameworks that may include columnar databases and other distributed management solutions, which come with many varieties of architectures. A distributed computing application for either Bigdata, Cloud, Gird or IoT is one in which multiple interconnected but independent computers coordinate to perform a joint computation. Different computers are independent in the sense that they do not directly share memory. Instead, they communicate with each other using messages, information transferred from one computer to another over a network.

8.2.1 Access Control Challenges of Distributed Systems

Enterprises want the same security capabilities for DS as are in place for non-DS information systems, including user authentication and authorization. However, the biggest challenge working with and leveraging technologies for DS data is to maintain data security, for which the fundamental techniques are access control policy enforcement

and management that allows organizations to safeguard their DS in order to meet security and privacy mandates. The existing system models are usually overwhelmed by the processing requirements, which were not designed and built with access control capability in mind [4]. Thus, most of them fail to adequately manage the creation, use, and dissemination of DS data and process. As a result, they either introduce friction into collaboration through excessively strict rules, or risk serious data loss by sharing data too permissively [5].

Authentication is different from authorization, as distinguished in [6]; the authentication management function is not directly related to the data content. For DS, as for non-DS data systems, authentication is generally handled by coordinated systems independently, thus the focus of DS security scheme is on authorization, which is more complex than for non-DS systems, because of the need to synchronize access privileges among the coordinated systems.

The characteristics of DS distributed computing pertains to a unique set of challenges for DS access control, which requires a different set of concepts and considerations; DS access control must not only enforce access control policies on data leaving the individual cooperated system, it must also control access to their local resources. Depending on the sensitivity of the data, it needs to make certain that DS applications on other coordinated systems have permissions to access the data that they are processing, and deal with the access to the distributed process and data from their local users [7]. Support for the DS's features complicates a system's access control implementation, because the difficulties are in general handled by the following techniques, each with its security challenges.

- Distributed computing—DS data is processed anywhere resources are available, enabling massively parallel computation between coordinated systems. This creates complicated environments that are highly vulnerable to attack, as opposed to the centralized repositories that are monolithic and easier to secure.
- Fragmented/redundant data—Data within DS clusters is fluid, with multiple copies moving to and from coordinated systems to ensure redundancy and resiliency. Data can become sliced into

fragments that are shared across them. This fragmentation adds complexity to the data integrity and confidentiality.

- Node-to-node communication—Coordinated systems usually communicate through unsecure protocols such as remote procedure call (RPC) over Transmission Control Protocol/Internet Protocol (TCP/IP) [4].

To answer the above challenges, ABAC is well-adapted for DS access control because ABAC provides granular and meta attribute definitions that support privilege assignments based on the structure of the DS framework, which requires federation and autonomy control between coordinated systems in a DS.

8.2.2 BigData Access Control as a Distributed System Access Control Example

BigData (BD) has denser and higher resolutions required by media, photos, and videos from sources such as social media, mobile applications, public records, and databases; the data is either in static batches or dynamically generated by machines and users by the advanced capacities of hardware, software, and network technologies. Examples include data from sensor networks or tracking of user behavior. Rapidly increasing volumes of data and data objects add enormous pressure on existing IT infrastructures, with scaling difficulties such as capabilities for data storage, advance analysis, and security. These difficulties result from BD's large and growing files, at high speed, and in various formats, as is measured by: velocity (the data comes at high speed, for example, scientific data such as data from weather patterns); volume (the data results from large files, for example, Facebook generates 25 TB of data daily); and variety (the files come in various formats: audio, video, text messages, etc. [8]).

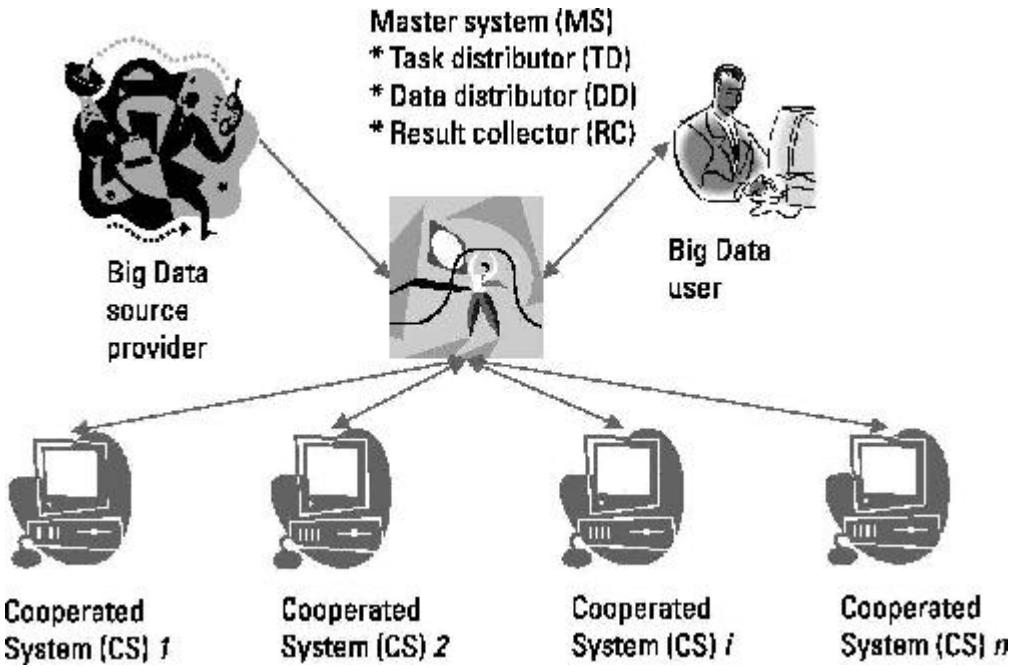


Figure 8.1 General BD model.

The fundamental model for most of the current BD architecture designs is based on the concept of DS [8, 9], which contains a set of generic processing systems as shown in [Figure 8.1](#):

1. Master system (MS) receives data from data source providers, and determines processing steps in response to a user's request. MS has the following three major functions:
 - a. Task distribution (TD) function is responsible for distributing processes to the cooperated (slave) systems of the BD cluster;
 - b. Data distribution (DD) function is responsible for distributing data to the cooperated systems of the BD cluster;
 - c. Result collection (RC) function processes, collects, and analyzes information provided by cooperating systems of the BD cluster and generates aggregated results to users.
2. Unless restricted by specific applications, the three functions are usually installed and managed in the same host machine for easy and secure management and maintenance.
3. Cooperated system (CS) (or slave system) is assigned and trusted by the MS for BD processing. CS reports progress or problems to the TD and DD, otherwise, it returns the computed result to the RC of MS.

A BD Cluster is a BD distributed system that networks MS and CSs to serve users' requests to process source data. The model in Figure 8.2 represents a generic BD DS architecture, such as the Apache Foundation's open source software Hadoop [10], which is used throughout industry and government. Hadoop keeps data and processing resources in close proximity within the cluster. It runs on a distributed model composed of numerous low-cost computers (e.g., Linux-based machines with simple architecture): two main MS components: TD—MapReduce and DD—file system (HDFS-Hadoop file system), and a set of tools. The TD provides distributed data processing across the cluster, and the DD distributes large data sets across the servers in the cluster [11]. Hadoop's CSs are called slaves, and each has two components: task tracker and data node. The MS contains two additional components: job tracker and name node. Job tracker and task tracker are grouped as MapReduce, and name node and data node fall under HDFS.

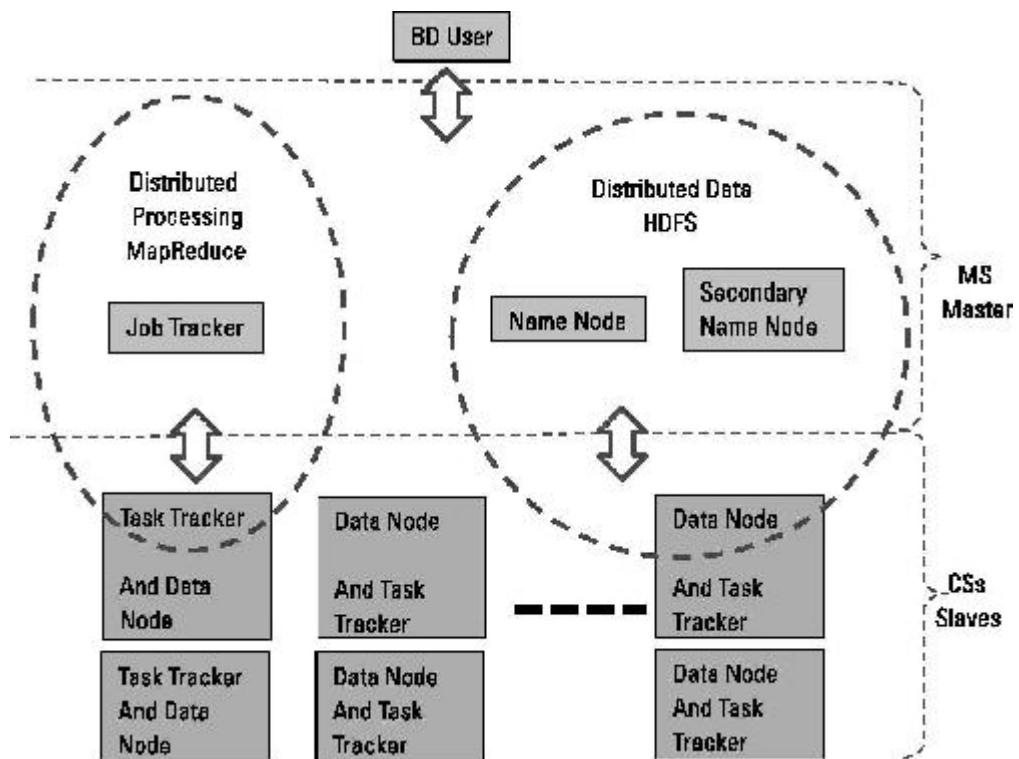


Figure 8.2 Hadoop BD cluster example.

Hadoop combines storage, servers, and networking to break data as well as computation down to small pieces. Each computation is assigned a small piece of data, so that instead one big computation, numerous small computations are performed much faster, with the result aggregated and sent back to the application [8]. Thus, Hadoop provides

linear scalability, using as many computers as required. Cluster communication between MS and CSs manages what traditional data service systems would not be able to handle.

Example ABAC Scheme for BD

Many access control models and mechanisms can be applied to support the BD scheme. One of the most versatile is the attribute-based access control (ABAC) model, because of its capabilities in configuring and managing attributes and access control policies. It also supports the access control requirements of enterprise systems, where BD is usually serviced as described in [Chapter 8](#) and [\[6\]](#), which provides guidance for ABAC enterprise implementation by the initiation, acquisition and development, implementation and assessment, and operations and maintenance phases of an enterprise.

Fundamentally, the ABAC requirements for a BD cluster are no different than non-BD systems. However, due to the facts that (1) BD is processed by distributing its processes and data from MS to CSs, and (2) BD data has no formal scheme for database management, BD ABAC needs additional ABAC capabilities than non-BD systems. A BD cluster is a construct of an enterprise system that requires MS's ABAC mechanism to be incorporated with CSs'. In terms of ABAC privilege, as defined in [\[6\]](#), when MS passes the process/ data to a CS, the MS is the subject, the BD process/data and required CS local resources are the objects, and the required actions are the actions in CS's ABAC policy. [Figure 8.3](#) shows an example ABAC scheme based on the general BD model described in [Section 8.2.2](#). The scheme includes ABAC components to meet the BD access control requirements as described below.

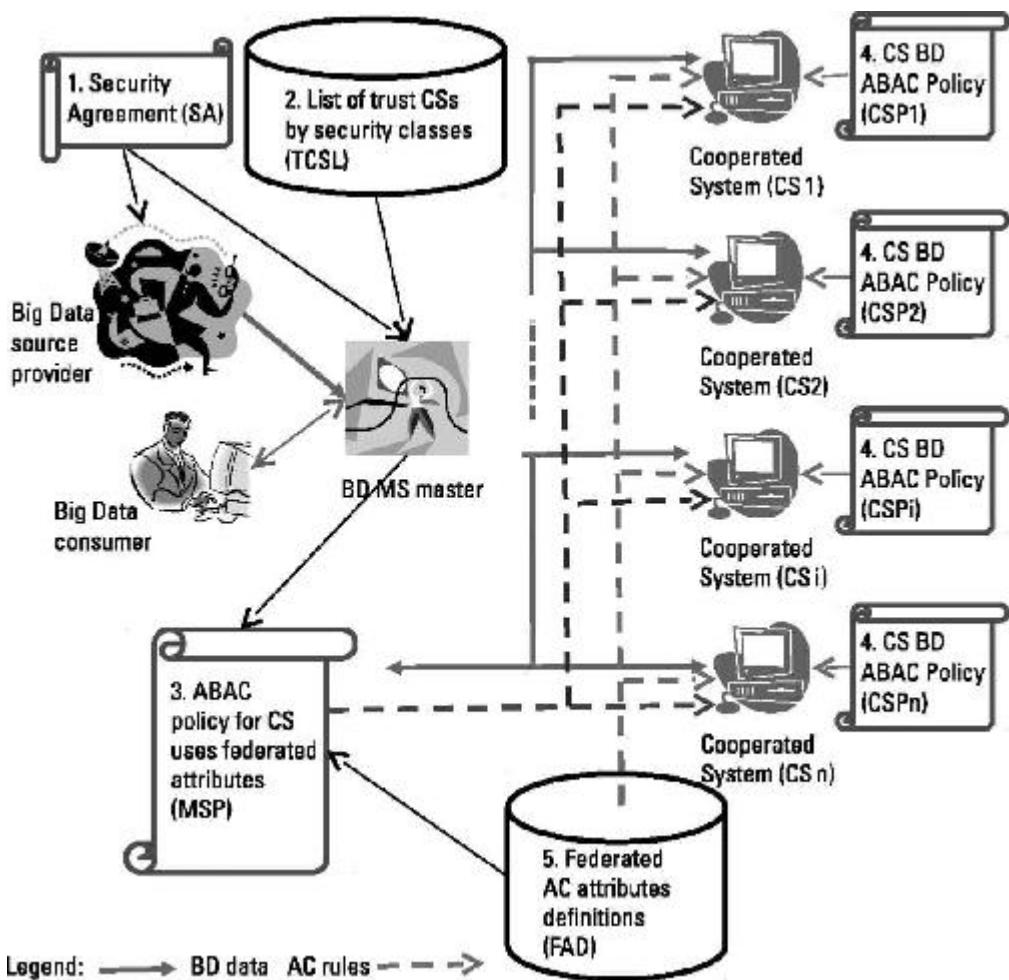


Figure 8.3 A generic BD ABAC architecture.

Security Agreement (SA) is a mutual agreement between BD source provider and the MS for defining security classes of data source. The purpose of the SA is for the data source provider and the MS to define and agree upon security classes (ranks), so that it can be referred by the MS and CSs to decide the levels of security (or trust) that a CS is qualified for processing the BD. For instance, a data source may be an email log file, and in considering the confidential level, the log can be accessed only by a CS with security class from 1 to 3.

Trust CS List (TCSL) lists the trusted CSs recognized by the MS. The TCSL categorizes CSs by the security classes according to the SAs worked with BD source providers. TCSL is managed by a MS security officer based on their knowledge of the associated CSs. For example, $CS-i$ is assigned to class 1, $CS-j$ and $CS-k$ are assigned to class 2, and $CS-l$ is assigned to class 3. In other words, TCSL allows the MS to determine how CSs are trusted for the distributions of BD process and data.

MS ABAC Policy (MSP) is managed by the MS security officer. MSP specifies a set of ABAC rules that are imposed by MS to enforce ABAC on CSs. For example, the distributed BD data can be read only by subjects with the attribute company employee, or the BD process can be executed only by processes with the subject attribute system administrator in a CS.

CS ABAC Policy (CSP) is managed by the CS security officer. CSP allows the CS to control the access to the distributed BD data/process by considering the processing capabilities (e.g., system load) and security requirements of the CS. For example, the distributed BD data cannot be written to disk space that is shared by other local CS users that have nothing to do with the BD, or BD data can be printed only from local printers. Additionally, the CSP needs to handle a situation when ABAC rules from other CS local policies conflict with CSP rules.

Federated Attribute Definitions (FAD) list the common attributes used by MS and CSs, so that the MSP and CSPs can be composed using the common ABAC attributes in the FAD dictionary. For example, the attribute local user is defined as all users who can log into the CS system, company employee is defined as all the CS users who have company's employee identifications, and system administrators is defined as the CS user who has system administrator privilege on the CS system. So, the FAD serves as the federated dictionary of ABAC attributes that should be syntactically and semantically agreed upon by the MS and CSs.

To apply the scheme, the following tasks need to be performed before the application:

- Coordinate BD source providers and MS for SA agreements;
- Collects information for MS about CSs based on the knowledge of CSs' security capabilities, levels of assurances, or trust. The information is required for MS to define security classes in sync with BD source providers for SA;
- Coordinate MS and CSs to define attributes in FAD based on the common syntactic and semantic values of attributes for the BD processing needs;
- Prepares information for CS about how it trusts the MS (i.e., what local resources are available for the BD processing) and considerations for performance and security capabilities of the

CS's local system, as well as responsibility for disseminating the distributed BD process and data. All the information will be translated into CSP ABAC policy rules;

- Composes meta-rules for CS to handle conflicts between CS's local ABAC policies and CSP, as well as between MSP and CSP. Note that unless specifically agreed on, CSP ABAC policy rules should have higher priority than MSP rules when determining access privileges. If CS denies a BD process's access, it should notify MS for transferring the task/ data to other CSs; and
- Consider including access activity audit capability for BD access logs for MS and CSs.

Formally, the proposed scheme can be represented by:

- A set of security classes from c_1 to c_n in the set $C = \{c_1, \dots, c_n\}$.
- A set of data source providers from bd_1 to bd_n in the set $BD = \{bd_1, \dots, bd_n\}$.
- A set of CSs from cs_1 to cs_n in the set $CS = \{cs_1, \dots, cs_n\}$.
- A set of federated attributes from at_1 to at_n in the set $FAD = \{at_1, \dots, at_n\}$.
- A set of (bd_x, c_x) pairs in the set $SA \subset BD \times C$.
- A set of (cs_x, c_x) pairs in the set $TCSL \subset CS \times C$.
- A set of MS ABAC policy rules from mp_1 to mp_n in the set $MSP = \{mp_1, \dots, mp_n\}$, each $mp_i = (at_x, a_x, bd_x)$ is a tuple where, $at_x \in FAD$ is an attributes, a_x is an action, and $bd_x \in BD$ is a source provider that means subject with attribute at_x is permitted to perform action a_x on object from bd_x .
- A set of ABAC policy rules for CS CSi in the set $CSPi = \{cspi_1, \dots, cspi_n\}$, each $cspi_i = (bd_x, a_x, rs_x)$ is a tuple where $bd_x \in BD$, a_x is an action, and rs_x is a local resource from CSi that means subject from bd_x is permitted to perform action a_x on object rs_x .

Let the $BDU = (u, a_u, bd_u)$ represent a BD user request; where u is an authenticated BD user by the MS, a_u is a requested action, and bd_u is a

BD source provider of the data/ process that u is request to perform a_u from. The algorithm to accept (u, a_w, bd_u) on the CS cs_l is:

```

BDAC {
    C_u = {c_1,...,c_k} such that (bd_j, c_i) ∈ SA;
    if C_u = ∅ {
        request = deny /* for this cs;
    else
        CS_u = {cs_1,...,cs_k} such that (cs_i, c_i) ∈ TCSL and
        c_i ∈ C_u;
        if CS_u = ∅ or cs_1 ∉ CS_u {
            request = deny /* for this cs,
        else
            if (there exist (mp_x = (al_x, a_x, bd_x) ∈
MSP such that a_c == a_x and bd_j == bd_x)) and (there exist
(csp_x = (bd_x, a_x, rs_x) ∈ CSP such that a_c == a_x and bd_c =
= bd_x and rs_x == resource required)) {
                request = grant /* for this cs_1 ;
                if perform a_c on rs_x == success {
                    return result to RC
                else
                    return "RC resource from cs_
unavailable"
                }
            else
                request = deny /* for this cs-
            }
        }
    }
}
}

```

The following demonstrates an example for the algorithm. The data source provider x enforces security class 2 as defined in SA. Class 2 is a level of moderate trust by some mutual criteria between x and MS. MS sets up MSP and regulates that only system administrator can process x 's BD after the subject attribute: system administrator is syntactically and semantically agreed between MS and participating CSs. MS also assigns $CS-i$, $CS-j$, and $CS-k$ (assuming the higher the number, the higher the security classes) to security classes greater than or equal to class 2 in the TCSL. Assume $CS-i$'s local ABAC policy does not allow to process nonlocal data, as well as $CS-i$'s CSP gives local policy higher priority than MSP. And $CS-j$ does not give any system administrator privilege to nonlocal process; thus, only $CS-k$ can process request for x 's data.

Figure 8.4 depicts the BD ABAC control/manage domains where MS ABAC domain is enforced collectively by information in the SA, TCSL, MSP, and FAD entries, which are all configured and managed by MS. CS ABAC domain is enforced collectively by information in MSP, FAD,

and CSP entries; however, only CSP entries are configured and managed by the CS. Note that at_x in the FAD and cs_x in TCSL are used to determine if a CS is permitted to process the data process request. The cs_x is independently decided by the MS, but at_x needs to share the responsibilities with trusted CSs for access privilege decisions. Thus, the chain of trust is from data source provider to MA through SA, then from MS to CS through TCSL, MSP and FAD, also from CS to MS through FAD and CSP which shows that both MS's and CS's ABAC domains do not allow unauthorized access without the coordination between MS and CSs.

To be general, a core concept of the scheme is provided. For practical implementations, additional sets and rules for the algorithm that cover other system components such as backup, logs, and so forth, might need to be included.

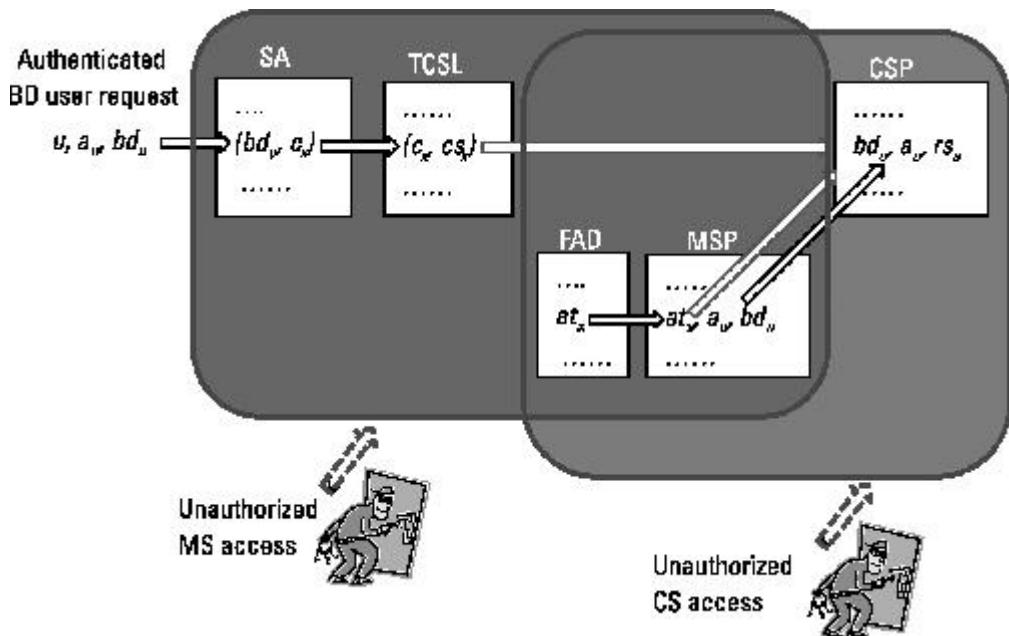


Figure 8.4 BD ABAC control/manage domain.

8.2.3 Implementation Considerations

In addition to the scheme, operational considerations that are common for DS ABAC mechanisms also need to be addressed as below.

Trust Establishment

For better performance, an individual coordinated system might need to separate parallel tasks that process immense volumes of data. In this case, each task process should also protect the data from an untrusted coordinated system, especially when their processes are delegated in a virtual environment such as that of a Cloud. For example, some suggest ensuring the trustworthiness of coordinated systems by way of mandatory access control (MAC) mechanisms, so that each coordinated system must be authenticated and given properties, and only when they're competent can they be assigned tasks. After this qualification, periodic updates must be made to ensure each coordinated consistently meet established policies [12].

Content Attributes

For some DS applications such as BD, it is hard to retrieve required attributes, for example, by traditional database queries or analytic tools. This is because established procedures for manipulating data typically rely upon imposition of a rigid structure or schema (defining elements of the data to be a location, a device type, a zip code, a user ID, etc.) and the precalculation of indexes to recall specific data fields (attributes), such as used in RDBA or NoSQL. These approaches excel in managing highly structured data but are less well suited for DS applications that handle unstructured data [5].

Access Control Auditing

The nature of distributed processing in DS clusters poses a challenge for authorization activity auditing, which tends to be incapable of analyzing the interconnected data between coordinated systems that are responsible for tracking the actual interactions with files and resources spread across their operating systems and applications. To minimize the problems, organizations may want to implement auditing functions in the infrastructure layer, simplify complexity in the application space, and adopt authentication and MAC. However, scaling these technologies to the levels necessary to accommodate DS can present its own set of unique challenges [5, 12].

Additional Consideration for Cloud DS

With the emergence of Cloud platforms, many might consider processing data in a Cloud environment, to handle the growing volume and complexity of data. However, Cloud computing has both positive and negative effects as data becomes more accessible through the Cloud, there are three major threats to Cloud DS: malfunctioning infrastructure components, infrastructure outside attacks, and infrastructure inside attacks. To address these threats, the Cloud server should improve trustworthiness and the usability of client (coordinated) systems by strengthening the ABAC with fine-grained attributes.

8.2.4 Analysis and Conclusions

To harvest DS benefits, security challenges must be overcome. Security professionals apply most controls at the very edges of the network. However, if attackers penetrate the DS cluster, they will have full and unrestricted access to the DS [4]. Thus, many organizations are being required to enforce access control and privacy restrictions on DS to meet regulatory requirements. In this section, we introduced a general DS process and challenges for DS access control. As an example, we presented a BD ABAC scheme and considerations based on trust between BD source providers and BD master system (MS), and consequently, between MS and cooperating systems (CSs). The scheme is focused on ABAC to protect BD processing and data from insider attacks in the BD cluster, under the assumption that authentication is already established. In addition to ABAC components, we demonstrated the formal sets and algorithm, and the domain of protection for the scheme that showed that no unauthorized privileges, either from MS or CSs, are possible. Finally, we discussed some operational and implementation issues for practical application. These issues are tied to the application, and should be handled according to the DS security requirements.

Depending on the security requirements of the DS application, many ABAC mechanisms can be applied to DS access control which stems from the fundamental ABAC attributes of subjects, objects, actions, and sometimes environment conditions which are the building blocks of the ABAC model, to determine access permission.

8.3 ABAC for Web Services

In the previous section ([Section 8.2](#)), we saw the application of ABAC for Big-Data systems. In this section, we look at ABAC deployments for another class of distributed systems—Web services. Stand-alone Web services and Web services called from workflow processes are both discussed.

8.3.1 Web Services— A Brief Background

The first generation Web services technology platform is comprised of the following core open technologies and specifications: Web Services Description Language (WSDL), XML Schema Definition Language (XSD), SOAP (formerly the Simple Object Access Protocol), Universal Description, Discovery, and Integration (UDDI), and the WS-I Basic Profile [13, 14]. The most popular paradigm nowadays for architecting high-performance and scalable Web service applications consists of using microservices [15] and REST interfaces [16]. To protect the resources of RESTful services (applications), ABAC has been found to be the most suitable access control model because of its ability to support a wide range of policies as well as the ability of an ABAC-based framework to authorize each request to a RESTful resource individually.

A Web service application can either be stand-alone or integrated with workflow processes. A stand-alone Web service is one that just receives a request and returns an execution result. A workflow process (such as a business process described by the Business Process Execution Language (BPEL)) on the other hand, is characterized by a dynamic state that is described by various parameters (e.g., process instances associated with the workflow), including variables relating to execution history, current task, and control flow (sequence, parallel, etc.).

In [Section 8.3.2](#), we discuss the features of ABAC that make it suitable for access control in Web service environments. We also precede the discussion of these features by pointing out the limitations of other access control models for Web services. In [Section 8.3.3](#) we consider the ABAC deployment for environments with just stand-alone Web services (without workflows), while in [Section 8.3.4](#), we consider environments which contain Web services that are part of (or invoked from) workflow processes.

8.3.2 ABAC Suitability for Web Service Environments

The limitations of previous access control models that do not use multiple attributes (e.g., Standard RBAC) for deployment in Web service environments are as follows:

1. The only abstraction for capturing user (subject) characteristics is the role. Hence granularity or differentiation in user privilege assignments based on other characteristics of the user (e.g., skill level between two users belonging to the same role) cannot be captured.
2. The characteristics of resources (other than identifiers) are not used in other access control models such as RBAC and hence specialized processing based on them (e.g., inventory maintenance operations targeted for resources labeled as high value) cannot be incorporated into access control requirements.
3. Fine-grained definitions of services for Web service call for using large number of attributes associated with resources and subjects (perhaps much more compared to actions and environment). To provide fine-grained access control for these services requires the use of correspondingly large number of security (access) relevant attributes, which is possible in RBAC only through definition of more roles [17]. The increase in the number of roles contributes to a high overhead in associated administrative operations such as user-role assignments and role-permission assignments. The usage of a large number of attributes involves much less overhead in the case of access control requirements expressed in rules and policies because of the flexibility available through the use of conjunctions and disjunctions available in first order logical expressions.

ABAC provides clear advantages for modeling access control requirements for Web service environments due to the following:

1. A Web service environment has standardized syntax for data representation (e.g., XML, JSON) and standardized communication protocols (e.g., HTTP). An ABAC deployment architecture for that environment can mirror the same principles with standardized XACML syntax for expressing policies, standardized XACML architecture for runtime access request processing, and standardized syntax and protocol for encoding

and transmitting attributes such as SAML syntax and SAML protocol respectively [18].

2. The richness of functionality and consequently the value for Web service applications comes from fine-grained definition of services using multiple characteristics or attributes. Hence an access control system based on attributes with the same semantics provides an effective authorization mechanism due to the following: (1) the access control elements mirror the semantics of the service definition constructs and hence no extra mapping is required (e.g., role to permission mapping), and (2) a finer level of granularity is possible for access control because of the use of service attributes as security-relevant attributes for authorization. ABAC, with its capability to use any number of attributes with unrestricted associated values (e.g., single, multivalued or complex data structures), becomes the natural fit for effective access control model for Web services environments.
3. Granular authorizations enabled by ABAC policy rules enable fine-grained alterations or adjustments to a subject's access profile, thereby enabling certain security requirements to be met, such as principle of least privilege and static separation of duty (SoD). Also, under some circumstances, any unexpected violation of SoD due to unexpected values of attributes at rule/policy evaluation time can be addressed through suitable definition of obligations that accompany a rule/ policy [19].
4. An additional benefit of using ABAC for Web Service is that it allows for flexible definition of subjects. Specifically, since policy rules can be attached to any actor, it can be extended to Web Service software agents as well [19].
5. Access control rules in ABAC can be defined based on any arbitrary set of attributes as opposed to specific identifying attributes (e.g. unique identifier) of the subject or resource. Thus, neither users (subjects) nor information assets (resources) need to be identified in advance which makes it easier to collaborate and to share information across security domains [19].

8.3.3 ABAC for Web Service Environments without Workflows

To provide effective access control to Web services with cross-domain user base (users identities are not stored in the host systems and the users are from different domains), ABAC policies should capture (or be based on) the semantic information associated with the Web service. This is due to the fact that the same term or attribute (e.g., group) may mean different things in different Web services. An example of a Web service with this requirement is the one that provides data sharing in a collaborative environment. ABAC deployment in such a Web service environment requires the following:

- To capture semantic information about a Web service, various ontologies must be created (such as requestor ontology, the service ontology, and environment ontology) by using an ontology management system. The ontology management system monitors and analyzes domain information pertaining to the Web service based on ontology and stores the inferred knowledge in a semantic knowledge base. Examples of inferred knowledge in a semantic knowledge base include derived and extended attributes associated with subjects, resources and environments.
- The policy administration point (PAP), in the XACML-based ABAC deployment, can then use the semantic knowledge base to formulate semantic-aware ABAC policies based on extended/derived attributes.

An example of an ABAC deployment, called *A Semantic and Attribute-Based Access Control Framework (S_ABAC)*, incorporating the above features has been proposed in [20]. The data flow diagram showing the sequence of steps involved starting from access request to access decision response/ obligations is shown in [Figure 8.5](#). The description of each step follows.

Step 0: This step is a preprocessing step. It starts by creating the related ontologies (such as the requestor ontology, the service ontology, and the environment ontology) using the ontology management system. The ontology management system monitors and analyzes domain information based on the ontology and stores the inferred knowledge associated with subjects, resources and environments.

Step 1: A policy administration point (PAP) creates an XACML policy and provides it to the PDP. The PAP differs from that in the standard XACML architecture in the sense that PAP looks up the ontology at the ontology management system in order to use the extended/ derived attribute terms to formulate semantic-aware ABAC policies.

Step 2: An access requester sends an access request to a resource which gets processed by the PEP.

Step 3: The PEP sends the request to the context handler which may include attributes of the subject, resource, action, and environment.

Step 4: At run time the context handler creates and constructs an XACML request and sends it to the PDP.

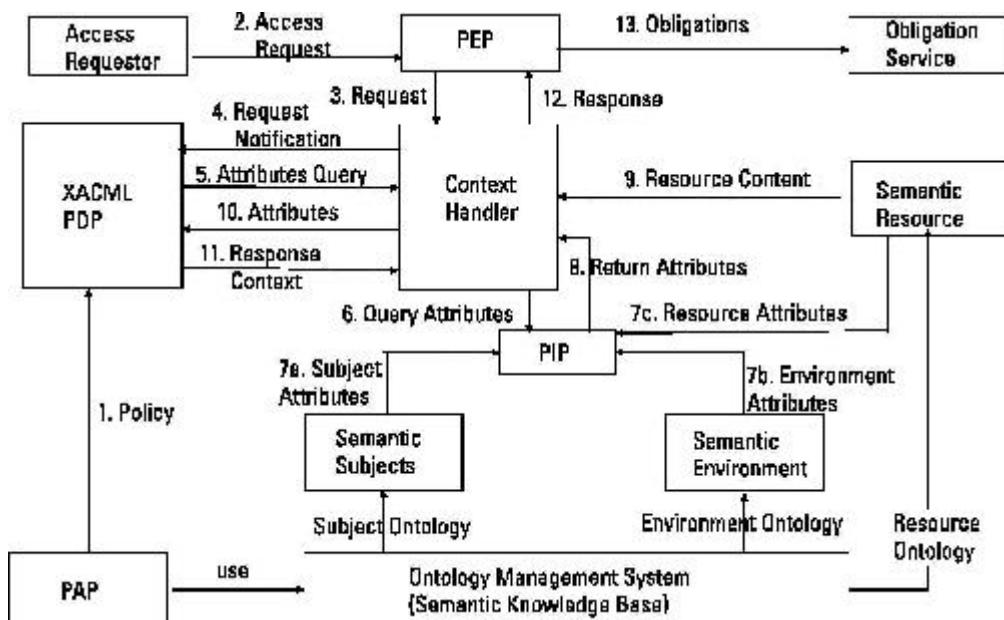


Figure 8.5 ABAC deployment for Web service without workflow.

Step 5: The PDP requests any additional subject, resource and environment attributes from the context handler.

Step 6: The context handler requests those additional attributes from a policy information point (PIP).

Step 7: The PIP obtains the requested attributes (along with their values) from the semantic subject, semantic resource, and semantic environment, which means the attributes may be the contexts of the extended/ derived attributes inferred from the related ontologies.

Step 8: The PIP returns these attributes to the context handler.

Step 9: Optionally, the context handler includes the resource content in the request.

Step 10: The context handler sends the requested attributes (including the extended/ derived attributes) and optionally, the content of the resource to the PDP. The PDP evaluates the access request against the policies.

Step 11: The PDP returns the response decision to the context handler.

Step 12: The context handler translates the response message back to the native response format and returns the response to PEP.

Step 13: The PEP satisfies the possible obligations.

Step 14: (not shown in [Figure 8.5](#)) If access is granted, the PEP allows access to the resource. Otherwise, access is refused.

The reader can note the following differences between the architecture of S_ABAC (ABAC deployment framework for Web service without workflows) and the standardized XACML architecture for ABAC deployment as follows:

- The extra step, Step 0, that uses an ontology management system to generate the semantic knowledge base.
- The PAP that looks up the semantic knowledge base to construct semantic-aware ABAC policies, which are made up of extended/ derived attributes and provides them to PDP. These extended/ derived attributes are extracted from ontologies associated with subject, resource, and environment.
- The PIP obtains extended/ derived attributes from semantic subject, semantic resource and semantic environment repositories so that they are semantically consistent with such attributes found in semantic-aware ABAC policies in the S_ABAC PDP. These semantic subject, semantic resource, and semantic environment repositories are constructed from the semantic knowledge base based on respective ontologies. It should be noted that the attributes originating from the context handler should be in the context of the extended/ derived attributes obtained from these repositories. In other words, the user (subject), resource, and environment attributes in the user access request are used as the basis for extracting associated extended/ derived attributes. The

mapping between the attributes in the user access request to extended/ derived attributes is provided by the ontological description of the semantic subjects, semantic resources, and semantic environment repositories or directly from semantic knowledge base.

In summary, we have looked at an ABAC deployment for a stand-alone Web service environment (i.e., without workflow) that requires semantic access control. The deployment framework makes use of an ontology management system that performs semantic inference prior to access control time and stores the results of inference in a semantic knowledge base. This pre-processed information is then used by PAP to formulate semantic-aware ABAC policies, thus reducing any run-time delays in processing an access request that is usually encountered when using semantic-aware access control policies.

In the ABAC deployment for Web services without workflows, described so far, the standard XACML architecture has been augmented with domain ontology to leverage the semantic information associated with Web service in the formulation of access control policies. In this architecture, the resource ontology (one of the three ontologies used, along with subject and environment ontologies) merely contained the semantic schema of the resource. However, there is a specialized class of Web service called data providing Web service (DPWS) that implements a class of a generic data providing service (DPS) [21] that not only allows data retrieval but requires this function for very fine-grained access to data. The semantic annotation of a DPWS is called a DPWS profile. As in any standard Web service, the eXtensible Markup Language (XML) is used for specifying the underlying data structures, the Web Service Description Language (WSDL) is used for Web service contract, and the Simple Object Access Protocol (SOAP) is used for exchanging information. In such a Web service, it is not sufficient that the resource ontology merely contains the semantic schema elements. It should associate a filtering class with each data element so that access decision can be made at the level of each data element. To meet this requirement, an ABAC deployment that contains not only a domain ontology (for resource), but also a filtering ontology has been proposed [22].

A detailed description of the filtering ontology and its associated components is needed to understand the ABAC deployment architecture for DPWS environment. To begin, what distinguishes the filtering ontology from the domain ontology for a resource is that the latter contains the semantic schema for resources while the former consists of a set of filtering classes. The classes are arranged in the form of hierarchy. This is to facilitate the assignment of access rights in the following way. The grant or deny access can be applied to a parent filtering class and the rule/ assertion will apply to all children filtering classes. The filtering classes are formed from the domain ontology by grouping the terms in domain ontology using a set of domain to filtering rules (D2F). Any resource that is not specifically linked to a filtering class is mapped to a special filtering class called general. This general filtering class is made the apex filtering class of the filtering class hierarchy. Assigning grant or deny to this general filtering class enables creation of simple access control policies (ACPs) that specify either a blanket access or denial of access respectively to every possible data element in the filtering classes below it. An example of a filtering ontology showing the hierarchy of filtering classes (and the related domain ontology) is shown in [Figure 8.6](#). The filtering ontology (filtering classes) and D2F rules are combined to form the published filtering description (PFD) [22].

The PFD and the DPWS profile (referred to earlier) are both used by a module called filtering class decision point (FCDP) to determine what filtering class an individual XML data element belongs to. This XML element is a domain ontology-mapped version of the local XML element that is part of the user resource access request. In fact, what the FCDP does is to take an XML element described in local terms and finds the mapping to the entity in domain ontology. It then checks all D2F rules to figure out which filtering class the element belongs to. Since a typical user access request can consist of multiple XML elements, the FCDP is invoked multiple times by the context handler (in the ABAC deployment architecture of DPWS) for determining all the filtering classes associated with a user access request. As there is a possibility of several user access requests having common XML elements, the FCDP uses caching to speed up the retrieval of filtering class corresponding to an XML element in a new access request.

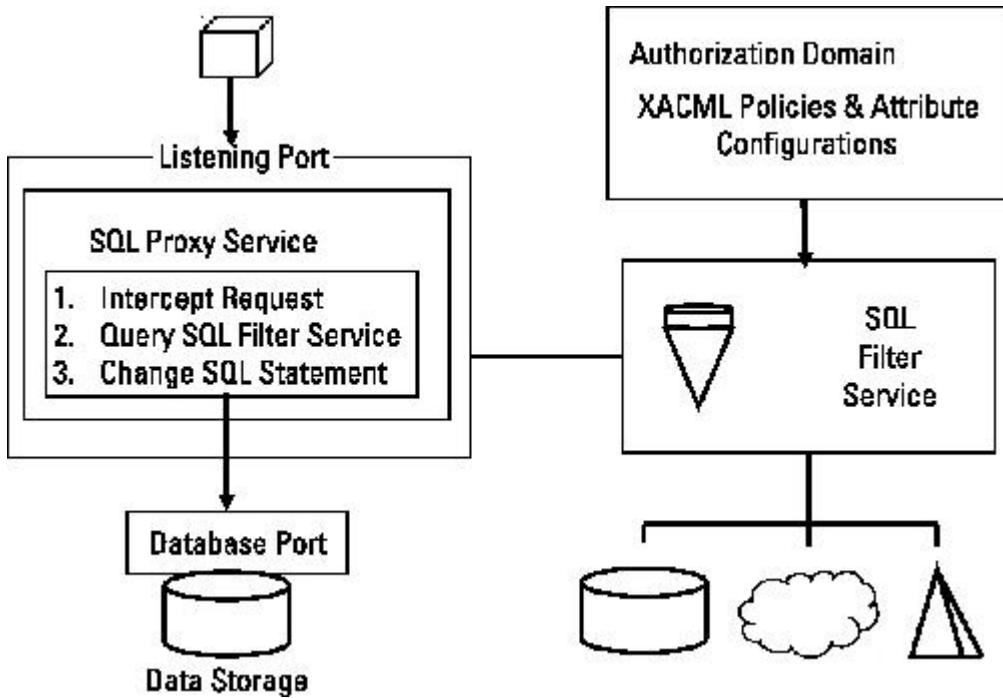


Figure 8.6 Domain and filtering ontologies.

The use of FCDP to retrieve filtering classes is necessitated by the fact that access control policies (ACPs) in DPWS ABAC architecture specifies grant and deny access based on a combination of a role and a filtering class. Stated another way, any ACP in DPWS ABAC deployment [22] defines which role should be granted or denied access to a particular filtering class. The ACPs are generated locally by each organization within the collaborative environment. Out of the two components (role and filtering class) of ACP, from our description of filtering class generation, we can see that filtering class formulation is also made by each organization using the DPWS profile and the domain ontology associated with resources provided by its own Web service. The other component of ACP (i.e., the role) is also defined by the organization within the context of its own Web service offering within the collaborative environment. This role in the DPWS ABAC architecture is an abstraction of the attributes of the user making the request. In other words, each organization defines a role based on the user attributes it receives. Since the user may belong to any organization within the collaborative environment, it is imperative that the user attributes have common semantics across all organizations, so that the local role definition is meaningful. To facilitate this, a common set of user attributes is agreed to beforehand by all organizations within the collaborative environment. These user attributes are retrieved and

supplied by a single sign-on system (SSO) after user authentication. These attributes are then used in role determination. Both user sign-in (authentication) and role assignment (collectively called the authorization component) are implemented in PIP in the DPWS ABAC architecture.

The role that is mapped by PIP is sent to the context handler. We saw that FCDP supplies the filtering classes associated with each XML data element sought in the user access request to the context handler. Since both components (role and filtering class) are now available to context handler, it uses this information to obtain an access decision for each XML element from the PDP. Since many XML data elements may map to a filtering class and since the context handler would have already encountered those role-filtering class combinations before, it can avoid making a fresh call to PDP for every instance of this combination and instead obtain the access decision based on previous calls that are stored in its cache. Unlike the context handler in a standardized XACML architecture, the context handler in DPWS ABAC deployment has the extra job of labeling the XML data elements with grant or deny labels based on the decision sent by PDP.

The XACML architecture for DPWS ABAC deployment in [10] also contains another component called filtering definition (FD). The FD is an extensible stylesheet language transformations (XSLT) file that will be applied on the labeled XML file coming from the context handler. It translates the XML so that each element in the output is referred to in domain ontology terms as well as removes elements that have been labeled deny. In the case where removing an element would render the XML invalid (the element would be a mandatory element as opposed to being an optional element in XML schema) with regards to the schema, the XSLT keeps the element tag and assigns the value deny.

A brief description of the process involved in generating the XSLT filtering definition file is in order. The XSLT filtering definition file is generated by an off-line module (not part of the XACML architecture) called a filtering definition generator (FDG). The FDG is only needed when the DPWS profile or the Web service contract changes and it is responsible for creating (re-creating) the filtering definition. The FDG takes the service contract as well as the DPWS profile as inputs. Using these inputs, a filtering definition is created that takes into account the domain ontology and ensures that the schema is preserved.

The overall access decision process in the XACML architecture for DPWS ABAC deployment is illustrated in [Figure 8.7](#) below and a description of the individual steps in the process follow.

Step 0: A user signs into the system. Based on a set of attributes the user is assigned a role. The role is kept by the PIP.

Step 1: The user requests a data resource from the DPWS.

Step 2: The DPWS forwards the request to the context handler where the request is formatted in such a way that the PDP can interpret it.

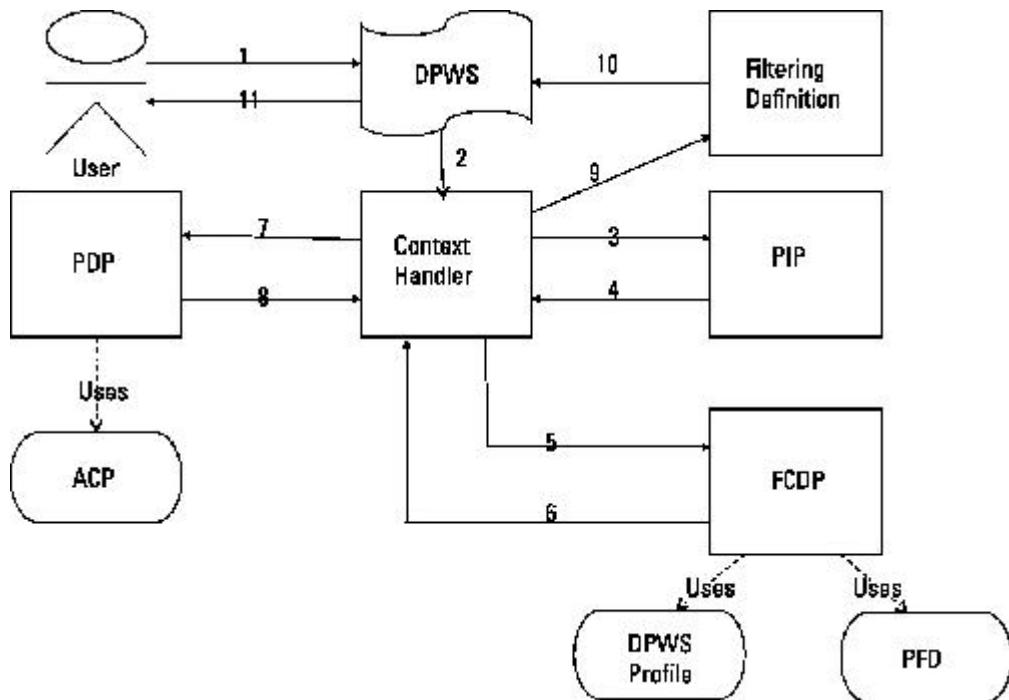


Figure 8.7 ABAC deployment for data providing Web service (DPWS).

Step 3: The context handler will request information about the user from PIP.

Step 4: The PIP will return to the context handler the role name corresponding to the user. The role name is needed as part of the request to the PDP.

Step 5: The context handler will send a single XML data element to the FCDP.

Step 6: The FCDP will use the PFD and DPWS profile in order to determine which filtering class that element belongs to. The

filtering class will be returned to the context handler where it is needed for the PDP request.

Step 7: The context handler sends the resource request (in the form of user role and resource filtering class) to PDP.

Step 8: The PDP uses the XACML access control policies (ACP) to determine if the user should be granted permission to that filtering class. The outcome of the decision is conveyed to context handler. If the outcome of the decision process is to deny permission, the requested XML data element is labeled as deny by context handler. Otherwise the element is left as is with the underlying logic that any element not explicitly labeled as deny is permitted. This reduces the number of labels to be processed during the subsequent filtering process.

Steps 5 through 8: These are repeated for each element in the XML data that has been requested by the user. As the PDP is basing its decisions on the filtering class rather than on a specific element, if the context handler, through the use of FCDP, determines that multiple XML elements in the user request belong to the same filtering class, the element will be labeled accordingly. This way, the number of requests that must be evaluated by the PDP is reduced.

Step 9: After each XML element has been labeled based on the PDP's decisions, the entire document will be sent to the filtering definition. An example of a labeled XML file is given in [Figure 8.8](#) below.

Step 10: The filtering definition is an XSLT file that will be applied against the labeled XML file (i.e., labeled with deny label). In this process, every element that has been labeled deny will be removed. In some cases, the filtering definition will determine that the element cannot be removed completely as it would break the schema. In these cases, the element would remain in the output although if a value is needed, that value would be deny.

Step 11: The filtered response is returned to the user. An example of a filtered response corresponding to the labeled XML file in [Figure 8.8](#) is given in [Figure 8.9](#).

The reader can note the following differences between the modified XACML architecture for DPWS ABAC deployment and the standard XACML architecture.

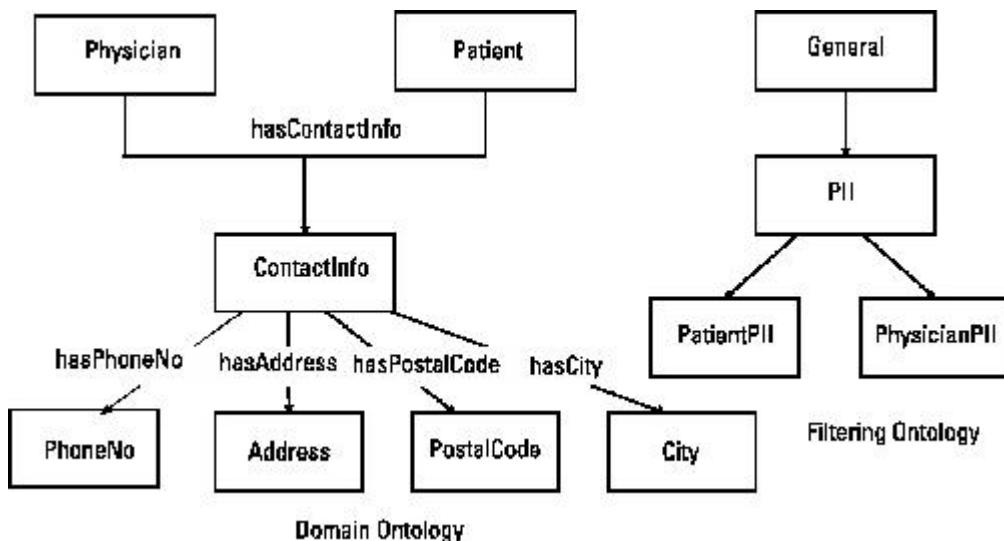


Figure 8.8 A labeled XML file.

```

<Physician>
    <PhysicianID>56575859</PhysicianID>
    <Name>John Smith</Name>
    <Contact>
        <Address permission="Deny">111 Wilson Lane</Address>
        <City permission="Deny">London</City>
        <PostalCode>M1M2M2</PostalCode>
        <Phone permission= "Deny">519-422-4242</Phone>
    </Contact>
</Physician>

```

Figure 8.9 A filtered response.

- There is no PEP module. The user request is made directly to the DPWS resource server holding the resource described using XML data elements. All relevant XML data elements (with their associated values) are passed on to the context handler under the initial assumption that the user has full access to the specified resource.

- The PIP does not perform the function of retrieving the user (subject), resource, and environment attributes as in standard XACML architecture. Instead it first performs the role of single sign-on (SSO) module with functions such as user authentication and retrieval of standardized set of user attributes. The PIP then uses a set of role assignment rules (expressed in terms of these attribute) to determine the role associated with the requesting user based on his or her attributes.
- There is an additional module called filtering class decision point (FCDP) which retrieves the filtering class associated with each XML data element in the user request. This module is invoked by the context handler.
- The context handler, in addition to querying the PDP for access decision (grant or deny) for each role-filtering class pair, does the extra function of labeling each of the XML data elements it has received from DPWS resource server (and that is associated with the filtering class) with the grant or deny decision as appropriate. Usually only the XML data element that has been denied access is labeled with a deny label and those that are permitted are left as is with their corresponding values. Then the labeled XML file is sent to the module holding the filtering definition.
- The last additional module is the one holding the filtering definition file. This is an XSLT file that is applied on the labeled XML file from the context handler. Depending upon the need to preserve the structure of XML schemata, the XML data element with a deny label is either completely removed (name tag as well as value) or the name of the XML element is retained, carrying the value deny. The resulting file is called the filtered response and is passed on by DPWS resource server back to the user that made the request.

8.3.4 ABAC for Web Service Environments with Workflows

ABAC deployments for Web services that are called from workflow processes need a different architecture and requirements than those that are stand-alone. Before we look at those requirements, brief background information on workflows and workflow-based systems will be helpful.

A workflow involves a set of activities (or tasks) to be executed by certain subjects with certain actions over certain resources to accomplish a goal. It is thus a chain of activities that have to be executed in certain order. Each activity may involve calling one or more Web services. A real-world example of a workflow is a business process. A business process is thus a composition of not only a set of activities but also a set of Web services called by those activities. Hence, a number of XML-based business process specification languages have been proposed and the one widely used is Web Services Business Process Execution Language (WS-BPEL) [23], or BPEL for short.

As already stated, application environments can consist of not only stand-alone Web services, but also BPEL processes (workflow processes). The workflow processes are modeled using a workflow model (or workflow process definition) which outlines all its component activities each with its starting and stopping conditions, the users who are allowed to participate, the tools and/ or data needed to complete the activity, constraints on activity execution, and last but not least the control and data flows among the activities. BPEL is used for orchestration of all services in such an environment. Orchestration involves ensuring that Web services interact with each other at the message level, including the business logic and execution order of activities. The dynamic behaviors in this type of Web service application are related to run-time states and histories, including variables of workflow processes and flow control results and histories. The execution of a workflow-based system is intrinsically dynamic; for example, the branching that occurs during workflow execution may depend upon the values of certain variables in a process instance, and the presence of branching in a previous execution may influence the subsequent execution. Thus, static specification of access control cannot fulfill the requirements of the workflow-based Web service environment, since dynamic behaviors of a workflow-based system influences access control.

BPEL is a XML-based language for the composition of Web services to form an executable process. Process, as described by BPEL, specifies a set of Web service operations, their execution order, dependencies, sharing data, exchanging messages, and so on in the business (workflow) process. The BPEL process describes the execution order of Web service operations by activities. There are two types of activities in BPEL: basic

activities and structured activities. An example of a basic activity is invoking an activity used to do a remote Web service operation. The purpose of a structured activity is to define the flow logic of a BPEL process: flow for parallel execution, sequence for sequential execution, and so forth. BPEL also includes some other elements, such as the <partnerLinks> element that are used to identify external Web services invoked from within the process. Another example of such an element is the <variables> element that defines the data that flows within the process. BPEL however does not provide the access control mechanisms for invocation of Web services within it. However, since it has been established that ABAC can meet the access control requirements of a Web service, an ABAC model is a feasible proposition for specifying access control requirements for invoking Web services in a business (workflow) process context. The applicable ABAC model elements in the context of a Web service in workflow process are shown in [Table 8.1](#) below.

Access control requirements for any domain where ABAC is used are captured through a combination of ABAC policies and an ABAC deployment architecture that will facilitate run-time evaluation of access requests. Hence it is imperative that we look at the unique access control requirements for the domain described by WS-BPEL, in order to appreciate the ABAC deployment architectures that have been proposed for that environment.

8.3.4.1 Access Control Requirements for Web Services in Workflow-Based Systems

Table 8.1
ABAC Model Elements for Web Services in Workflow

| ABAC Model Element | Referenced Entity in Workflow Process Context |
|--------------------|--|
| Object or resource | A named Web service called from the workflow process |
| Action | Calling (or invoking) a named Web service |
| Subject | The calling workflow process |

Access control requirements for invoking a Web service in a workflow context should be based not only on the semantics of a Web service (captured using several attributes of the Web service as discussed in [Section 8.3.3](#)) but also on the invocation context (i.e., the workflow

(business) process from which it is invoked as well as the particular activity within that process). These additional requirements, as well as the ABAC policies to capture those requirements, are described in the following paragraphs.

- Typically, in a workflow environment, the same Web service is used by multiple business processes. An example to illustrate this situation is given in [24]. The example considers a document flow system where there are two processes (secret document flow process and ordinary document flow process) that need users to call the document signing service at particular steps. The two processes need different policies to permit or limit users' access. In secret document flow process, if a user wants to access a document signing service, his or her trust-level must be at high-level or higher. In the other process (ordinary document flow process), if the user's trust-level is at middle-level or higher, he or she can access the service. This is not surprising since the trust-level of users participating in the secret document flow process must be higher than those participating in the ordinary document flow process. From the ABAC model perspective, a trust-level is an example of a subject attribute being assigned to a user depending upon the level of trust that the system wants to place on that user. Since different subject attribute values are required for invoking the same resource (i.e., document signing service) depending upon the process from which it is invoked, it is clear that different access control policies are applicable depending upon the calling process. Hence in an ABAC deployment using an XACML architecture, the PAP module must be business process-aware when it supplies the relevant policies/rules to the PDP module.
- From the above discussion, it should be clear that from the Web service ABAC model perspective, the business process and the particular activity from which the Web service is invoked represents the execution context. Since in a typical ABAC model the execution context is captured through environmental attributes and associated values, it is clear that the XACML architecture for ABAC deployment should have a source for

supplying the name of the process and the particular activity within the process as environment attribute values.

A modified XACML architecture for access control for Web services in a workflow process context (WS-BPEL environment) has been proposed [24]. In this architecture, the need for a business process-aware PAP module is met by a process policy management module that provides the appropriate ABAC policy (depending upon the process) to the PDP module. The need for supplying the appropriate environment variables (associated with the process name and activity name) and their values are supplied by a process state management that keeps track of the current activity in all processes. These two modules, process policy management and process state management, together with a process execution engine that executes the individual activities of the process based on the business (workflow) rules specified in BPEL file, are encapsulated in a process management point.

A schematic diagram of the ABAC deployment for WS-BPEL environment based on the above referenced architecture is shown in Figure 8.10 [24]. A detailed description of access request processing at run time based on this framework follows:

Step 1 (data flows 1-3 in Figure 8.10): The process execution engine with the process management point reads the BPEL files (which contains description of the underlying workflows in the business process), chooses to execute an activity in BPEL. If the running activity needs a user to call a Web service, the process policy management module would activate the access control policy associated with this activity. In fact, it is the same process policy management module that creates the access control policies for invocation of various Web services from various processes and activities and stores them in a XML document that associates with the BPEL file containing the corresponding process description. The process state management module would modify the state of the process and update and hold values about the state variables, such as next activity and current activity.

Step 2 (data flows 4-5 in Figure 8.10): Before the user assesses Web services, he/she sends requests to PEP. PEP does not respond to the

user immediately, but asks PDP whether the user has permission to call the service.

Step 3 (data flows 6-10 in [Figure 8.10](#)): The PDP receives the request of PEP, gets user attributes from the subject attribute engine, gets service attributes from the object attribute engine, and gets environment attribute from not only the environment attribute engine, but also from the process state management module within the process management point, as it is an essential component of the workflow-based business process environment. The PDP also receives the access control policies relevant to the business process from which the user made the access request to the Web service. It then decides whether to grant or deny the access request based on the policies and attribute values.

Step 4 (data flows 11-12 in [Figure 8.10](#)): PDP sends the result of policy decision to PEP. If the result is grant (allowed or permitted), the PEP would send the user request to target Web service, otherwise, PEP would send message informing users that their request is denied.

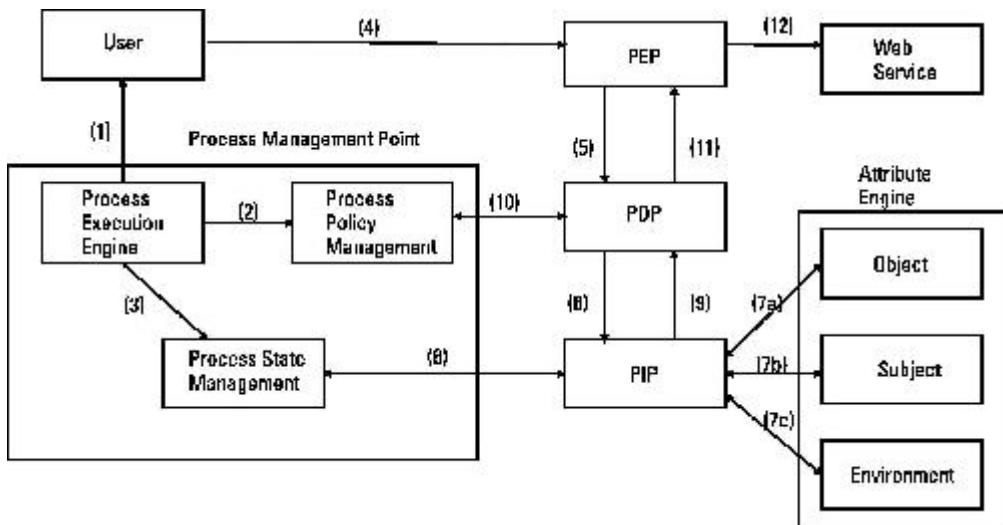


Figure 8.10 ABAC deployment for web services in workflows.

8.3.5 Combined Challenges in Using ABAC for Web Service Environments (with and without Workflows)

The inherent characteristic such as a federated architecture results in different trust levels and semantics for the attributes from the different domains in a Web service environment. Further, in Web services with

workflow contexts, the process state variables are used as access policy attributes (as we saw in [Section 8.3.4.1](#)). In addition, both access policies and attributes can be in a state of flux. The impacts of these characteristics on the governance of ABAC policies, such as attribute trust maintenance and semantic consistency, attack detection, and audit strategies, are described below.

1. The integrity of access control rule evaluation, and from hence the integrity of the final access control decision in ABAC depends upon the quality, integrity, and timeliness of attribute values. This requirement is essential for any ABAC environment, but becomes all the more critical for a Web service application because of its federated architecture with multiple security (trust) domains since the attribute sources from diversified domains should carry the necessary trust assurance. Hence, a higher level of governance measures for attribute maintenance (such as integrity verification, trust conveyance, etc) is required compared to the corresponding measures in both stand-alone and other distributed environments.
2. The access policies in any domain are defined with respect to the semantics of subject, action, resource, and environment attributes (elements), as well as associated values local to that domain. Therefore in evaluating an access request from a subject in another domain, the above referred attributes pertaining to that domain (where the request has originated) have to be mapped to relevant attributes in the local (target) domain, where the resource is located and managed. This requires extensive maintenance of ontology mappings between the domains. Furthermore, in some instances the granularity of resources and access/ action definitions vary from one domain to another, and so definitions and mapping of filter classes between domains is needed as well.
3. When some attributes that define a system state are used as security-relevant attributes in the ABAC model, the log entries that contain past access decisions are based on the system state at the time of such decisions. This means meaningful analysis of these log entries for finding attacks, vulnerabilities, and privilege

leaks requires carrying the history of system state information as well, making it a complicated process.

4. The dynamic approach to access control based on current values of attributes associated with subjects, resources, and environment calls for a more sophisticated approach to audit techniques as well. For example, to answer the fundamental question of who got access to what resources at any point in the past depends on: (1) policies in effect at that time and (2) attribute values associated with subjects (users) and resources at that point in time. Due to this, the auditing tools for validating access decisions of the past have to exhibit awareness of the change histories associated with both policies and attributes.

8.3.6 Web Services Environment—Summary of Requirements

The discussions in [Sections 8.3.3](#) and [8.3.4](#) highlighted requirements for ABAC policy features and functional capabilities in associated XACML architectures for deployment of ABAC in Web services environments without and with workflows. [Section 8.3.5](#) outlined the ABAC policy governance challenges associated with both of these two environments. From an analysis of access control requirements and the proposed XACML architectures, the distinguishing ABAC policy features and the functional capabilities of the supporting XACML architectural components for any Web service environment can be summarized as follows:

- Semantic-aware ABAC policies: ABAC policies should be expressed based on entities that reflect the semantics of the Web service domain in order to provide the necessary protection assurance. However, the repositories that carry user, resource, and environmental information may carry attribute information associated with these entities in their native formats and terms. Hence the first functional component needed for ABAC deployment is an ontology management system that converts the subject, resource, and environmental attributes to their corresponding semantic counterparts and stores them in a semantic knowledge base. Secondly, the PAP component of the architecture should have the capability to interact with the

semantic knowledge base repository and formulate ABAC policies in terms of entities that reflect the semantics of the Web service domain (i.e., semantic-aware ABAC policies) instead of being based on the syntax of the terms found in the user, resource, and environment attribute namespaces as illustrated in [20]. Further, in some specialized Web services such as data providing Web service (DPWS), the XML schema elements that describe the structure of the data may not reflect the semantics of the underlying data sharing service. These environments therefore require classification of data elements into filtering classes and access policies defined in terms of those filtering classes, as in [22]. In order to support this policy feature, additional deployment modules such as filtering class decision points may be needed to store filtering class information pertaining to XML data elements, the capability in context handler to retrieve these filtering classes, and the logic in PDP to process relevant policies based on filtering classes to retrieve the set of permissible resources.

- Calling process-aware ABAC policies (for Web services in workflow environments): The need for different access control policies for invoking the same Web service from different processes was outlined in [Section 8.3.4.1](#), along with the need to carry additional contextual information in the form of some process state variables such as current activity and next activity in these policies. Therefore, in ABAC deployments for Web services callable from business processes, the PAP component should have the capability to be a complete process management point that can supply not only the access control policies for a Web service that is applicable to the specific process, but also the needed contextual information in the form of process state variables, as well the specific activity within a process, as in [24].

8.4 ABAC for Stand-Alone Workflow Processes

In [Section 8.3.4](#) we saw ABAC deployments for Web services in a workflow context. However, there are systems that implement a workflow process without calling any Web services. In these environments, ABAC policies have to merely cover specific activity

(task) access requests based on some allowable parameters (specific task instance, specific resource within the task instance (e.g., a named document), etc.). Before we look at ABAC deployment for stand-alone workflow processes, we have to look at the unique challenges in formulating ABAC policies for workflow processes and the associated information resources needed to define, maintain, and enforce those policies:

8.4.1 Challenges and Requirements for ABAC Configuration for Stand-Alone Workflow Processes

There are many IT applications used for automating or facilitating business processes. These processes are commonly found in areas such as healthcare, assembly line manufacturing, purchase order processing, etc. The effectiveness of these IT applications in meeting the process needs depends on the proper definition of workflows with appropriate activities (tasks). The tasks, in turn, are executed by certain subjects (a user or program acting on behalf of a user) with certain actions over certain resources under certain conditions. Since ABAC policies are meant to govern the legitimacy of these tasks based on attributes associated with subjects, actions, resources, and environmental conditions, it is obvious that in order for ABAC policies to achieve their intended goal, they must be closely aligned with workflow definitions.

Having discussed about workflow definitions and ABAC policies, it is necessary to look at the drivers for each of them and some concrete examples of how mismatches between them can occur, before we look at the challenges and requirements for ABAC. The drivers for formulating the workflow definitions and ABAC policies are definitely different and are outlined below:

- Workflow definitions flow from business process engineering principles and the associated tools based on these principles.
- ABAC policies are based on data and IT resource protection principles, driven either by external regulations or by internal IT security norms/ best practices.

Some concrete examples to show how the misalignment between the workflow activities definition and ABAC policies may occur are given below:

1. Workflow definitions may not explicitly take into account certain attributes. For example, a workflow definition may state that a teller may generate withdraw and deposit transactions (action attributes) against active customer accounts (resource attributes) during normal business hours (environmental attributes), but may not explicitly state that these actions can be performed only from teller workstations carrying an IP address within the bank's domain (additional environmental attributes). In these situations, the workflow activity definition has to be enhanced to include these additional attributes.
2. In certain scenarios, the workflow activity definition may involve complex functions based on the content of the resource, while the ABAC policy definition may merely be based on certain attribute values in the resource's schema. For example, the workflow definition may state that a payment authorization (action attribute with action-id = write or append) on a purchase order record (resource attribute with resource-id = purchase order) can be performed by an accounts-payable clerk (user attribute with role = accounts-payable clerk) under the condition that the purchase order has been approved by a purchase manager and the approval date is not earlier than 60 days. The conformance measure for this definition calls for verification of the valid authorized user in the approving official field and the computation that the approval date value is no more than 60 days in the past. The ABAC policy governing the authorization for accounts-payable clerk for this resource may merely be based on the value of the order-status attribute of the resource carrying the value payment-approved and not based on the complex function involving the value of multiple fields in the resource's content.

Based on the background information regarding drivers and scenarios for misalignment between workflow definitions and ABAC policies, it is obvious that the misalignment has access control security assurance implications, and therefore it is necessary to perform some form of cross-analysis of workflow activity definitions and access control policies in order to address the misalignments, so that the resulting ABAC policies are effective and fully address the security needs of the workflow application.

The primary challenges in ABAC deployment for stand-alone workflow process are the following:

- Semantic mismatch between workflow definition elements and attributes used in ABAC policy constructs at the level of granularity and abstraction. For example, the task definition may be in terms of resource name and content (loans with a value greater than \$500,000) while the ABAC policy constructs may be in terms of some resource attributes obtained through metadata or schemata (loan objects with type attribute standing for the type of loan such as car, home, etc.). This may result in lack of coverage of ABAC policies for tasks in all execution scenarios. The insufficient coverage may result in information/ data leakage.
- When workflow definition (as a whole) or an individual task definition changes, the task attributes impacted by it (e.g., location of the task in the execution sequence, the pre-conditions and post-conditions of task execution) must be correctly identified in order to make appropriate changes to relevant ABAC policies since the latter is based purely on task and user attributes.

The requirements for ABAC deployment for stand-alone workflow processes, which can meet the above challenges as well as provide additional safety features, are as follows:

- Have a methodology to express workflow definitions and ABAC policies based on some common terms.
- Reliable sources for providing several types of contextual information needed for ABAC policy evaluation at run time. Some examples of sources are: workflow engine for supplying the workflow status information (such as current task), task status manager for supplying the status of all currently executing tasks, execution history repository for supplying the task execution history information, and policy constraints repository for supplying the necessary policy constraints (such as separation of duty, etc.).

8.4.2 ABAC Deployment for Stand-Alone Workflow Processes: Integrated Approach

In the ABAC deployments for stand-alone workflows that have adopted the integrated approach, the access control subsystem enforcing access control rules is tightly integrated with a workflow subsystem enforcing workflow rules. In other words, the run-time access decision modules interact in real time with the workflow modules. An architecture based on such an integrated approach has been described in [25]. The salient features of the underlying ABAC model and deployment architecture are the following:

1. From the ABAC modeling perspective in the stand-alone workflow process environment, a particular process is treated as an environment with a set of attributes characterizing it. Since a workflow consists of a certain set of tasks executed in certain order, the task into which an executing workflow process instance runs into is characterized by the attribute current task. Thus, current task becomes an attribute of the environment.
2. Similarly, an example of an ABAC resource is the task (or activity) and an attribute of the task is its state (or status). In the workflow environment pertaining to the architecture under discussion, a task can be in one of six states: submit, exist, run, complete, failed, and suspended. The operating privileges on a task at any point in time depend upon its state at that time. Correspondingly, users have different operating privileges when the same task is in different states.
3. User attributes are generally in the form of authorized tasks and operating authority (on what states of a task they have rights to). This makes the user and task attributes consistent with each other and enables assignment of set of privileges to the user that is strictly equal to the operating privilege set that is appropriate for the task state at any given time. A user authorized to execute a task is assigned the corresponding attributes and the relevant values to enable successful task execution. Apart from the attributes/ attribute values needed to invoke a task in multiple states, a user may be assigned multiple tasks and hence the set of attributes associated with a user will always be a superset of the attributes associated with any single assigned task.
4. In addition, certain constraints are to be defined. Some are enforced at the time of assignment of attributes to a user and

some are enforced at run time in the form of privilege restrictions. For example, for a workflow process dealing with purchase order (PO) flow in an organization, the roles (user attributes) PO-APPROVER and PO-PAYMENT-AUTHORIZER are not assigned to the same user. An example of a pair of roles whose association is enforced at run time is: PO-INITIATOR and PO-APPROVER. These two roles can be assigned to the same user but for a particular PO instance, if the user has exercised the privilege stemming from his or her PO-INITIATOR role, he or she cannot exercise the privilege associated with his or her PO-APPROVER role on the same PO instance. The fact that the user has exercised the privilege stemming from his or her PO-INITIATOR role on a PO instance is obtained from the history information and used for appropriate privilege restriction (in our example in the form of preventing the user from assuming the role of PO-APPROVER) for operating on that same PO instance.

An ABAC model incorporating the above features, called a task-attribute-based workflow access control model has been described in [25] with an associated access enforcement framework or architecture. The data flow diagram of this architecture is given in [Figure 8.11](#), and the data flows involved in evaluating an access request to render an access decision are also described below.

Step 1: A user Ui sends a task request for task Tj to the PEP server.

Step 2: The PEP server, based on the request, augments it with attributes, builds an attribute-based access request (AAR), and passes it on to the PDP server. AAR describes the requester and the requested task.

Step 3: The PDP server queries the task manager about the current task in the workflow sequence. If the current task matches the task the user requested, the PDP server requests the verification manager for further evaluation of user's request. Otherwise, the system executes Step 8 and returns deny to the PEP server. Usually a user requests a task's operating authority when the workflow runs into an instance of the task, but the PDP (and the authorization decision process as a whole) cannot take this for granted but should independently verify it.

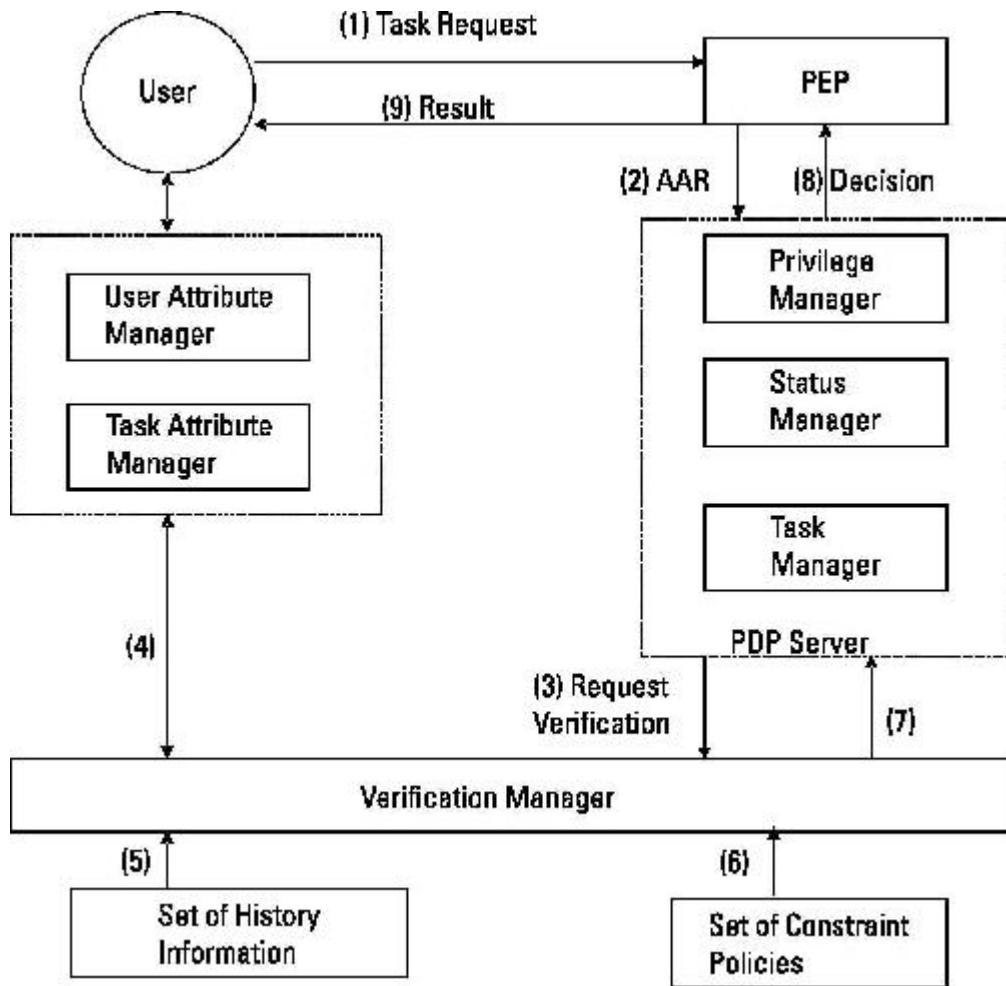


Figure 8.11 ABAC deployment for workflow process using an integrated approach.

Step 4: The verification manager extracts the set of attributes for user Ui and the task Tj from the user attribute manager and task attribute manager respectively. It then verifies that Ui 's attribute set includes the attribute set for task Tj . If not, the system executes Step 8 and returns deny to the PEP server.

Steps 5 through 7: If it does, the verification manager then reads the set of history information related to user and resource instance. The history information is in the form of a 4-tuple (Ui, Aui, Tn, n) where Ui is the user, Aui is the set of user attributes, Tn is the particular task, n denotes the sequence number of the task in the workflow. It then reads the set of constraint policies and dynamically executes constraint verifications to user's attributes and the task the user requested. An example of a constraint policy is the static separation of duty. For example, a user may be authorized to perform the task to approve payment for purchase orders as well as initiate a

purchase order. Although this generic authorization policy exists, in order to prevent that user from approving payment for the same purchase order he or she initiated (giving opportunity to commit a fraud), a constraining static separation of duty policy is needed. After ensuring that such a conflict of interest does not exist, the verification manager returns an authorization certificate (AC) to the PDP.

Step 8: The PDP then checks with the task status manager to find out the status of the task. During the execution process of a task, the privileges conferred to a task (accessible resources and allowed actions/ operations) vary according to the status of the task. In the lifecycle of a task, the status can assume one of the following values: submit, exist, run, complete, failed, and suspended. Based on the task status, the PDP then consults with the privilege manager to find out the allowed privileges for the requested task based on its current status. The result of authorization is sent to the PEP server.

Step 9: The PEP server returns the results of authorization to the user. The authorization result can be one of the following three values: permit, deny, or undetermined. The semantics for permit or deny are clear. When the authorization result is undetermined, the system may guide the user to disclose further information. When the system is executing Step 8, if the status of the task is completed in the status manager, the corresponding privilege is immediately withdrawn and the next job in the workflow sequence will become the current task. The initial status of the current task will be submit and the system will automatically provide the requisite operating privileges to the user invoking the new current task whose request has received the authorization certificate (AC) from the verification manager.

In summary, in the above architecture, a user request to execute a particular task with a particular operating authority will succeed if:

- Invoked task is the current task in the workflow sequence as indicated by task manager;
- The user attribute set provides coverage for task attribute set;
- The assigned user attributes meet the constraint policy requirements such as static separation of duty and dynamic

- separation of duty;
- Corresponding to the state or status of requested task as indicated in the status manager, the privilege manager indicates the corresponding operating authority.

8.4.3 ABAC Deployment for Stand-Alone Workflow Processes: Loosely Coupled Approach

We are starting with a scenario where there is an existing database or data repository on which an ABAC model, and an access enforcement architecture, has been deployed. There arises a need in the enterprise to use a workflow engine to automate a business process which wants to leverage the data objects (resources) already present in the database. Because of the presence of an ABAC deployment, there exists a repository of well-defined sets of attributes associated with users (subjects) and data objects (resources). The organization therefore wants to leverage the attribute repository to identify a set of subjects who will be performing each task in the workflow process and the resource(s) which will be used by each task.

What is needed for the organization at this point is to devise a methodology by which the existing ABAC model that is presently used for control of access to data objects in the existing database by multiple applications can also be used for meeting the access control requirements of the workflow process as well. One approach that can achieve this objective is to align workflow definitions and attributes used in ABAC policy constructs using some common terms, as we saw earlier. A formalization of this approach has been proposed in [26]. The approach adopted is to cross-examine the workflow tasks (activities) and ABAC policies (for the database objects) in terms of some common access elements to find matches and discrepancies. We know that ABAC policies have four access elements: subject, action, resource, and environment, abbreviated as SARE elements. An ABAC deployment (an ABAC model instance) consists of a set of ABAC policies expressed using multiple attributes (and their values) associated with each of the SARE elements. The SARE element values associated with an ABAC policy can be extracted from its policy targets and rule targets.

Now to meaningfully compare an ABAC policy and a workflow task, the latter also must be expressed in terms of its SARE element values.

This is possible using the workflow process definition since a workflow task can be described in terms of attributes involving the resources it uses and the users qualified to execute the task. Once this is done, comparison between the values associated with corresponding SARE elements in a workflow task and ABAC policy target can begin.

Let us now look at the process outlined in [26] for cross-examination of workflow tasks and ABAC policies using SARE element values with an example. Let us assume an ABAC deployment with three policies, P1, P2, and P3, and four tasks, T1, T2, T3, and T4. In [14], the term activity is used. Here, the term task is used instead, in order to be consistent with our workflow discussion in the rest of this chapter. To keep the discussion simple, we assume that there are no multiple values associated with the elements action and environment, and therefore we used a single fixed value, A1 and E1 respectively, across all tasks on the workflow side and across all policy targets on the ABAC deployment side. The SARE element values associated with tasks T1, T2, T3, and T4 are given in [Table 8.2](#) below.

The SARE element values entry {S1, A1, R1, E1} for task T1 above indicates that task T1 can be executed by subject S1, by action A1 on Resource R1 under the environmental condition E1. Similarly, the three ABAC policies P1, P2, and P3 can also be expressed using SARE element values. The semantics associated with these values in the case of ABAC policies are that these are values found in the policy targets and hence provide the contexts for allowable execution of the tasks that carry the subset of these values.

The next process step after expressing workflow tasks and ABAC policies using SARE element values is to see whether each workflow task is covered by at least one ABAC policy by way of matching SARE element values. Ignoring the values A1 and E1 which are common across all tasks (and policies), we find that the subject S1 and resource R1 in workflow task T1 is covered by ABAC policy P1 (take the set of SARE element values in each row of [Table 8.2](#) and look for coverage of that set in each row of [Table 8.3](#)). Going down the workflow task list in [Table 8.2](#), we find that workflow task T2 is covered by ABAC policy P2. Similarly, we find that workflow task T3 is covered by ABAC policy P1 again (this was previously identified to cover workflow task T1). However, we find that the workflow task T4 is not covered by any ABAC policy. The lack of ABAC policy coverage for Task T4 can be

addressed in two ways: add a new ABAC policy, or modify one of the existing policies to provide the coverage (in our case, by including the resource R4 within the target of ABAC policy P3, we can provide coverage for workflow task T4). However, care should be taken to see that such a modification to an ABAC policy does not result in privilege escalation for any other application (besides the workflow application) due to its expanded set of data objects in the ABAC policy target.

Table 8.2
SARE Element Values for Workflow Tasks

| Task Identifier | SARE Element Values (Extracted From Workflow Definition) |
|------------------------|---|
| T1 | S1, A1, R1, E1 |
| T2 | S1, A1, R2, E1 |
| T3 | S2, A1, R3, E1 |
| T4 | S3, A1, R4, E1 |

Table 8.3
SARE Element Values for ABAC policies

| ABAC Policy Identifier | SARE Element Values (Extracted From Policy Target) |
|-------------------------------|---|
| P1 | {S1, S2, A1, R1, R3, E1} |
| P2 | {S1, A1, R2, E1} |
| P3 | {S3, A1, R3, E1} |

The net result of the above methodology is to ensure that the existing ABAC model for the database with a few modifications can meet the access control needs of the workflow process. This methodology demonstrates that even though workflow activity descriptions and ABAC policies are in a different level of abstraction, expressing each of them using common SARE elements, it is possible to adapt an existing ABAC model deployment for a workflow process.

8.4.4 Analysis and Conclusions

We have seen two approaches for ABAC deployment in stand-alone workflow processes. They are:

- An integrated approach where the ABAC deployment is tailored to the workflow environment from the ground up ([Section 8.4.2](#)).
- A loosely coupled approach where an existing ABAC model for a database is modified to support the access control requirements of a workflow process that is going to be running on top of that database ([Section 8.4.3](#)).

Out of the two approaches, it is no surprise that the integrated approach is likely to provide better access security assurance for stand-alone workflow process because of the following semantic alignments between workflow process definitions and ABAC deployment model constructs:

- The unit of resource assignment is the task in workflow definition. The current task that is allowed for access is captured through the current task attribute of the environment in the ABAC model.
- The privileges needed for task invocation are dictated by the current status of the task in the workflow definition, and the ABAC deployment model can exactly capture this task state (status) by including the task and its current status through a set of associated resource attributes. Further user attributes that go with the set of resource attributes (capturing task and current status) can be combined together to form the appropriate target in the ABAC policies, thus capturing the total set of workflow process requirements for a task execution.

The above ABAC modeling paradigm enables design of an ABAC access decision engine that, at run time, can process an access request and render an access decision that is based on the combination of workflow process state and the invoked task status. Therefore, for any ABAC deployment to provide the needed security assurance to the workflow process, the ABAC model should use attributes that mirror workflow artifacts and the access decision logic should make use of state variables in the workflow engine.

References

- [1] Hu, V. C., et al, “An Access Control Scheme for BigData Processing,” *Proc. 10th IEEE International Conference Collaborative Computing*, October 22–25, 2014 Miami, Florida.
- [2] Hu, V. C., et al, “Model Checking for Verification of Mandatory Access Control Models and Properties,” *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)* Vol. 21, No. 1, 2011.
- [3] DATAIQ, “Big data to turn ‘mega’ as capacity will hit 44 zettabytes by 2020,” <http://www.dataiq.co.uk/news/20140410/big-data-turn-mega-capacity-will-hit-44-zettabytes-2020>, October 2014.
- [4] Zettaset, *The Big Data Security Gap: Protecting the Hadoop Cluster*, Zettaset White Papers, http://www.zettaset.com/wp-content/uploads/2014/04/zettaset_wp_security_0413.pdf, 2014.
- [5] Miller, P., “Applying big data analytics to human-generated data,” *Analyst Report*, Gigaom Research, <http://research.gigaom.com/report/applying-big-data-analytics-to-human-generated-data>, January 2014.
- [6] Hu, V. C., et al, “Attribute Based Access Control Definition and Consideration,” *NIST Special Publication 800-162*, Gaithersburg, MD, 2013.
- [7] Smith, K. T., “Big Data Security: The Evolution of Hadoop’s Security Model,” *InfoQ*, <http://www.infoq.com/articles/HadoopSecurityModel>, August 2014.
- [8] Mir, H., “Hadoop Tutorial 1: What is Hadoop?” *ZeroToProTraining*, <http://ZeroTOPro-Training.com>.
- [9] Bell, W., “The Big Data Cure,” MeriTalk, http://www.meritalk.com/wp-content/uploads/2015/12/MeriTalk_Big_Data_Cure_Press_Release.pdf, 2014.
- [10] Hadoop.apache.org
- [11] Moore, J., “How big data is remaking the government data center,” GCN, <http://gcn.com/articles/2014/02/14/big-data-data-centers.aspx>, February 2014.
- [12] Shea, S., “CSA top 10 big data security, privacy challenges and how to solve them,” *SearchCloudSecurity*, TechTarget, November 2013.
- [13] Erl, T., *SOA Principles of Service Design*, Prentice Hall/ Pearson PTR, 2007.
- [14] Paik, H., et. al., *Web Service Implementation and Composition Techniques*, Springer International Publishing AG, Switzerland, 2017.
- [15] Stetson, C., *Microservices Reference Architecture*, NGINX, 2017.
- [16] Barry, D. K., *Web Services Explained*, http://www.service-architecture.com/articles/web-services/web_services_explained.html, Barry & Associates, Inc., August 2017.

- [17] Esfandi, A., and M. Sabbari, “Study of Access Control Issue in Web Services,” *International Journal of Computer Applications* (0975-8887), Vol.49, No. 1, July 2012.
- [18] Sabbari, M., and H. S. Alipour, “Improving Attribute Based Access Control Model for Web Services,” *2011 World Congress on Information and Communication Technologies*, IEEE, 2011.
- [19] Alipour, H. S., M. Sabbari, and E. Nazemi, “A Policy Based Access Control Model for Web Services,” *6th International Conference on Internet Technology and Secured Transactions*, IEEE, December 11-14, 2011, Abu Dhabi, United Arab Emirates.
- [20] Hai-bo, S., “A Semantic and Attribute-based Framework for Web Services Access Control,” *2nd International Workshop on Intelligent Systems and Applications (ISA)*, May 22-23, 2010.
- [21] Brown, K. P., and M. A. M. Capretz, “ODEP-DPS: Ontology-Driven Engineering Process for the Collaborative Development of Semantic Data Providing Services,” *Information and Software Technology*, 2013. DOI: 10.1016/j.infsof.2013.02.011.
- [22] Sazio, M., M. and A. M. Capretz, “Web Service Semantic Access Control,” *3rd International Conference on Innovative Computing Technology (INTECH 2013)*, IEEE, 2013.
- [23] Jordan, D., and J. Evdemon, “Web Services Business Process Execution Language (WS-BEPL) Version 2.0,” *OASIS Standard*, April 2007.
- [24] Zheng, S., and Z. Bin, “An Extension Approach that Supports Attribute Based Control for WS-BPEL,” *International Conference on Computer Science and Network Technology (ICCSNT)*, IEEE, 2011.
- [25] Yi, L., X. Ke, and S. Junde, “A task-attribute-based workflow access control model,” *2013 IEEE International Conference on Green Computing and Communications (GreenCom) and IEEE Internet of Things(iThings) and IEEE Cyber, Physical and Social Computing(CPSCom)*, 2013.
- [26] Lakkaraju, S., and D. Xu, “Integrated Modeling and Analysis of Attribute Based Access Control Policies and Workflows in Healthcare,” *International Conference on Trustworthy Systems and their Applications (TSA)*, IEEE, 2014.

-
1. Some of content in this section is derived from *An Access Control Scheme for Big Data Processing*, by V. C. Hu, et al., in the Proceedings of the 10th IEEE International Conference of Collaborative Computing, October 2014.

9

ABAC Life-Cycle Issues: Considerations*

9.1 Introduction

This chapter examines ABAC life-cycle issues: planning, design, implementation, and operational considerations for employing ABAC within an enterprise with the goal of improving information sharing while maintaining control of that information. This chapter should not be interpreted as an analysis of alternatives to ABAC or other access-control capabilities as it focuses on the challenges of life-cycle issues of implementing ABAC rather than on balancing the cost and effectiveness of other capabilities versus ABAC.

When deployed across an enterprise for the purposes of increasing information sharing among diverse organizations, ABAC implementations can become complex—supported by the existence of an attribute management infrastructure, machine-enforceable policies, and an array of functions that support access decisions and policy enforcement.

In addition to the basic policy, attribute, and access control mechanism requirements, the enterprise must support management functions for enterprise policy development and distribution, enterprise identity and subject attributes, subject attribute sharing, enterprise object attributes, authentication, and access control mechanism deployment and distribution. The development and deployment of these capabilities require the careful consideration of a number of factors that will influence the design, security, and interoperability of an enterprise ABAC solution. These factors can be summarized as a set of activities:

- Establish the business case for ABAC implementation;

- Understand the operational requirements and overall enterprise architecture;
- Establish or refine business processes to support ABAC;
- Develop and acquire an interoperable set of capabilities;
- Operate with efficiency.

This chapter serves to help planners, architects, managers, and implementers fulfill the information sharing and protection requirements of organizations through the employment of ABAC.

9.2 Enterprise ABAC Concepts

While ABAC is an enabler of information sharing, when deployed across an enterprise, the set of components required to implement ABAC become more complex. At the enterprise level, the increased scale requires complex and sometimes independently established management capabilities to ensure consistent sharing and use of policies and attributes and the controlled distribution and employment of access control mechanisms throughout the enterprise. The following represents a definition of enterprise for this chapter:

Enterprise: A collaboration or federation among entities for which information sharing is required and managed.

[Figure 9.1](#) presents an example of the major components required to enable enterprise ABAC. Some enterprises have existing capabilities that can be leveraged to implement ABAC. For example, most have some form of identity and credential management to oversee the population of subject attributes, such as name, unique identifier, role, or clearance. Similarly, many enterprises may have some organizational policy or guidelines to establish rules authorizing subjects' access to enterprise objects. However, these rules are usually not written in a machine-enforceable format that can be integrated consistently across all applications. ABAC policies must be made available in a machine-enforceable format, stored in repositories, and published for access control mechanism (ACM) consumption. These digital policies include subject and object attributes required to render access control decisions. The enterprise subject attributes must be created, stored, and shared across organizations within the enterprise through a subject attribute

management capability. Likewise, enterprise object attributes must be established and bound to objects through an object attribute management capability. At this point, the ABAC-enabled access control mechanisms must be deployed [1–3].

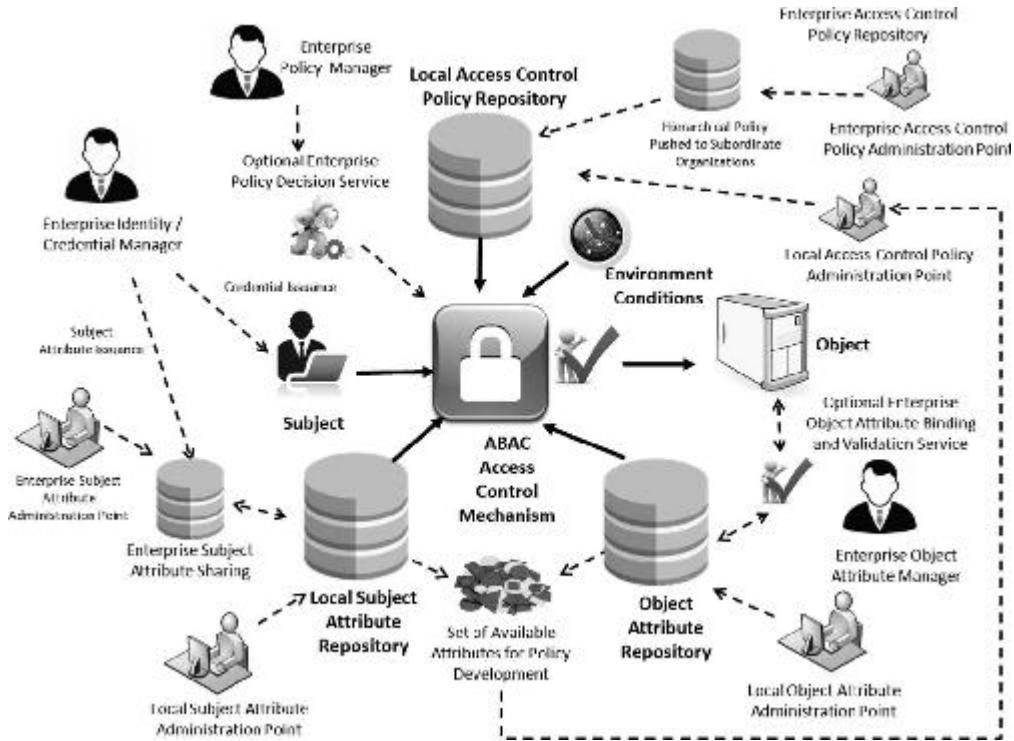


Figure 9.1 Enterprise ABAC scenario example.

Access control mechanism (ACM): The logical component that serves to receive the access request from the subject to decide and to enforce the access decision.

9.2.1 Enterprise ABAC Policy

Natural language policies (NLPs) are high-level requirements that specify how information access is managed as well as who, and under what circumstances, may access what information. NLPs are expressed in laymen's terms and may not be directly implementable in an access control mechanism. NLPs may be ambiguous and thus hard to derive in formally actionable elements, so the enterprise policy may be difficult to encode in machine-enforceable form. While NLPs can be application-specific and thus taken into consideration by the application system, NLPs are just as likely to pertain to subject actions that span multiple applications. For instance, NLPs may pertain to object usage within or

across organizational units or may be based on need-to-know, competence, authority, obligation, or conflict-of-interest factors. Such policies may span multiple computing platforms and applications. NLPs are defined in this chapter as follows:

Natural language policies: Statements governing management and access of enterprise objects. NLPs are human expressions that can be translated to machine-enforceable access control policies.

Given that relevant NLPs exist for each organization in an enterprise, the next step is to translate those into a common set of rules that can be enforced equally and consistently within the ACMs across the enterprise. In order to accomplish this, it is necessary to identify all required subject/ object attribute combinations and allowable operations. Often, these values will vary from organization to organization and may require some form of consensus or mapping to each organization's existing attributes to accommodate enterprise interoperability. The agreed-upon list of subject and object attributes, the allowable operations, and all mappings from existing organization-specific attributes are then translated into machine-enforceable format.

NLPs must be codified into digital policy (DP) algorithms or mechanisms. For efficiency of performance and simplicity in specification, an NLP may require decomposition and translation into different DPs that suit the infrastructure of operation units in the enterprise. DPs are defined in this chapter as:

Digital policy: Access control rules that compile directly into machine executable codes or signals. Subject/object attributes, operations, and environment conditions are the fundamental elements of DP, the building blocks of DP rules, which are enforced by an access control mechanism.

Multiple DPs may require metapolicies (MPs) or policies dictating the use and management of DPs to handle DP hierarchical authorities, DP deconfliction, and DP storage and updates. MPs are used for managing DPs. Depending on the level of complexities, hierarchical MPs may be required based on the structures for the priority and combination strategies specified by NLP. MP is defined in this chapter as:

Metapolicy: A policy about policies, or policy for managing policies, such as assignment of priorities and resolution of conflicts between DPs

or other MPs.

Once DPs and MPs are developed they need to be managed, stored, validated, updated, prioritized, deconflicted, shared, retired, and enforced. Each of these operations requires a set of capabilities that will often be distributed across the enterprise and is collectively termed digital policy management (DPM). There may be multiple policy authorities and hierarchies within organizations that have variations on enterprise policy. The rules for how DPs and MPs are managed may be determined by a central authority.

Proper DP definition and development are critical to the identification of subject and object attributes that are needed to render an access control decision. Remember that a DP statement is comprised of the subject and object attribute pairings as well as the environment conditions needed to satisfy a set of allowable operations. Once the full set of subject and object attributes needed to satisfy the entire set of allowable operations for a given set of enterprise objects is identified, this set of attributes comprises the entire set of attributes needed to be defined, assigned, shared, and evaluated for enterprise ABAC access decisions. For this reason, identifying the NLP and DP must be accomplished by the support of attributes when implementing an enterprise ABAC capability.

9.2.2 Attribute Management in Enterprise ABAC

Next, consider the lists of attributes developed while examining the NLPs and DPs. Without a sufficient set of object and subject attributes, ABAC does not work. Attributes need to be named, defined, given a set of allowable values, assigned a schema, and associated with subjects and objects. Subject attributes need to be established, issued, stored, and managed under an authority. Object attributes must be assigned to the objects. Attributes shared across organizations should be located, retrieved, published, validated, updated, modified, and revoked.

Subject attributes are provisioned by attribute authorities—typically authoritative for the type of attribute that is provided and managed through an attribute administration point. Often, there are multiple authorities, each with authority over different attributes. For example, *security* might be the authority for *clearance* attributes, while *human resources* might be the authority for *name* attributes. Subject attributes

that need to be shared to allow subjects from one organization to access objects in another organization must be consistent, comparable, or mapped to allow equivalent policies to be enforced. For example, a member of *organization A* with the role *job lead* wants to access information in *organization B*, except *organization B* uses the term *task lead* to denote the equivalent role. This problem also applies to mapping between an enterprise attribute schema and an application-specific schema, particularly those built before the enterprise schema is defined and/or commercial off-the-shelf (COTS) products that come with their own built-in schema. Organizations must normalize subject attribute names and values or maintain a mapping of equivalent terms for all organizations. This should be managed by a central authority.

Object attributes need to be established, maintained, and assigned to objects as objects are created or modified. While it may not be necessary to have a common set of object attributes in use across the enterprise, object attributes should be consistently employed to fulfill enterprise policy requirements. Available sets of object attributes should be published for those wishing to mark, tag, or otherwise apply object attributes to their objects. At times, it might be necessary to ensure that object attributes are not tampered with or altered to satisfy an access request. Objects can be cryptographically bound to their object attributes to identify whether objects or their corresponding attributes have been inappropriately modified. Mechanisms must be deployed to ensure that all objects created are assigned the appropriate set of object attributes to satisfy the policy being employed by the access control mechanism. It may be necessary to have an enterprise object attribute manager to coordinate these requirements.

In the course of managing attributes, the concept of meta-attributes—or characteristics of attributes—arises. Meta-attributes apply to subjects, objects, and environmental conditions as extended attribute information useful for enforcing more detailed policy that incorporates information about the attributes and for managing the volumes of data needed for enterprise attribute management. Meta-attributes are defined in this chapter as:

Meta-attributes: Information about attributes necessary to implement MP and DP processing within an access control mechanism.

9.2.3 Access Control Mechanism Distribution in Enterprise ABAC

Finally, consider the distribution and management of access control mechanisms throughout the enterprise. Depending on the needs of the users, size of the enterprise, distribution of the resources, and sensitivity of the objects that need to be accessed or shared, the distribution of ACMs can be critical to the success of an ABAC implementation. The functional components of an ACM may be physically and logically separated and distributed within an enterprise rather than centralized as described in the system-level view of ABAC.

Within the ACM are several functional points that are the service node for retrieval and management of the policy along with some logical components for handling the context or workflow of policy and attribute retrieval and assessment. [Figure 9.2](#) shows the main functional points: the policy enforcement point (PEP), the policy decision point (PDP), the policy information point (PIP), and the policy administration point (PAP). When these components are in an environment, they must function together to provide access control decisions and policy enforcement.

A PDP performs an evaluation on DPs and MPs in order to produce an access control decision. PDP and PEP are defined in this chapter as follows:

Policy decision point: Computes access decisions by evaluating the applicable DPs and MPs. One of the main functions of the PDP is to mediate or deconflict DPs according to MPs.

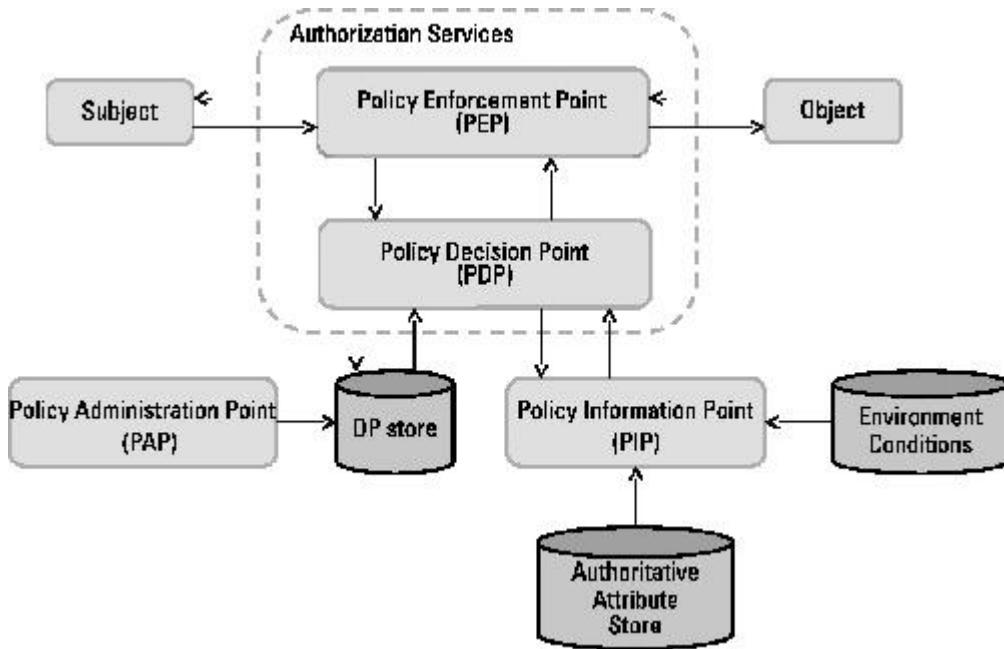


Figure 9.2 Example of access control mechanism functional points.

The PEP, described below, enforces decisions made by the PDP.

Policy enforcement point: Enforces policy decisions in response to a request from a subject requesting access to a protected object; the access control decisions are made by the PDP.

PDP and PEP functionality can be distributed or centralized and may be physically and logically separated from each other. For example, an enterprise could establish a centrally controlled enterprise decision service that evaluates attributes and policy and renders decisions that are then passed to the PEP. This allows for centralized management and control of subject attributes and policy. Alternatively, local organizations within the enterprise may implement separate PDPs that draw on a centralized DP store. The design and distribution of access control mechanism components requires a management function to ensure coordination of ABAC capabilities.

To compute access decisions, the PDP must have information about the attributes. This information is provided by the PIP. The PIP is defined in this chapter as:

Policy information point: Serves as the retrieval source of attributes, or the data required for policy evaluation to provide the information needed by the PDP to make the decisions.

Before these policies can be enforced, they must be thoroughly tested and evaluated to ensure they meet the intended need. This action is carried out by the PAP. The PAP can be defined as:

Policy administration point: Provides a user interface for creating, managing, testing, and debugging DPs and MPs, and storing these policies in the appropriate repository.

Finally, as an optional additional component within the access control mechanism, the context handler manages the order of policy and attribute retrieval. This can be important when time-critical or disconnected access control decisions must be made. For example, attributes may be retrieved in advance of an access request or cached to avoid the delay inherent in retrieval at the time of the access request. The context handler also coordinates with PIPs to add attribute values to the request context and converts authorization decisions in the canonical form (e.g., XACML) to the native response format. The context handler is defined as:

Context Handler: Executes the workflow logic that defines the order in which policy and attributes are retrieved and enforced.

9.3 ABAC Enterprise Considerations

Many factors must be considered before deploying an ABAC system across an enterprise. This section addresses consolidation of available guidelines based on the state of the technology to date and lessons learned through multiple attempts within organizations to deploy ABAC capabilities throughout a large enterprise. The guidelines are presented according to the phases of the NIST system development life cycle (SDLC) illustrated in [Figure 9.3](#). For more general information regarding the definitions of the phases and expected outputs, refer to [\[4\]](#). Most considerations for employment of enterprise ABAC fall within the first four phases: initiation, acquisition/ development, implementation/ assessment, and operations/ maintenance. This section focuses on those phases exclusively.

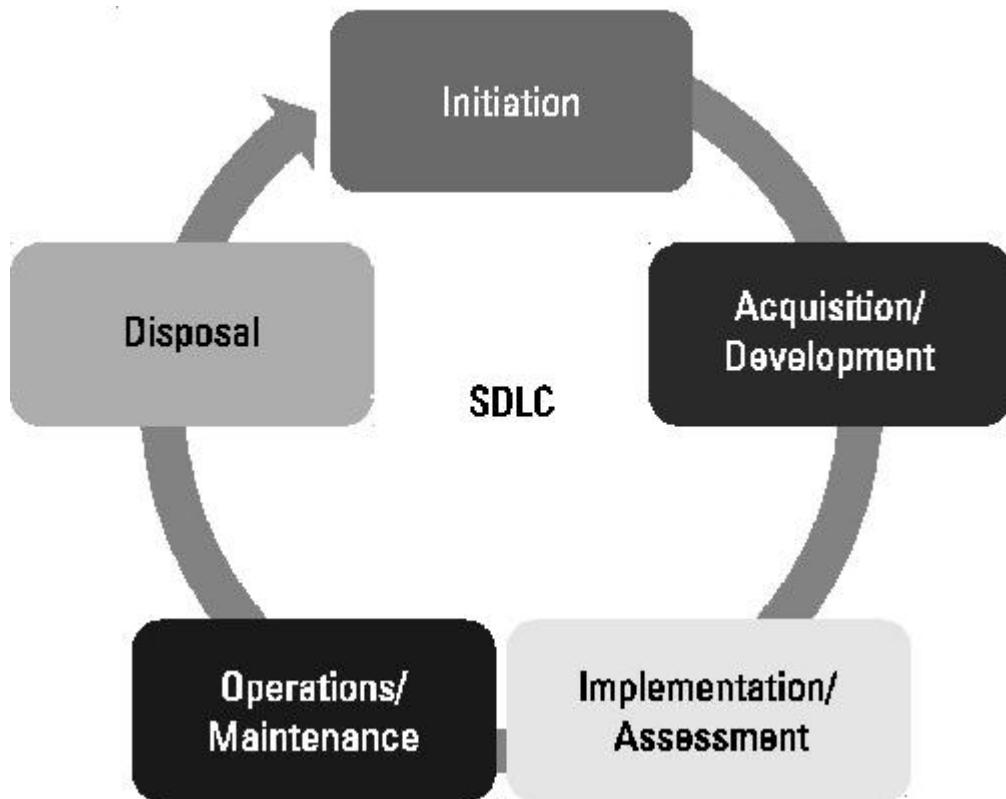


Figure 9.3 Access control mechanism: NIST SDLC.

The development and deployment of an enterprise ABAC capability requires the careful consideration of a number of factors that will influence its design, security, and interoperability. These factors lead to a set of questions that should be considered:

- *Establish the business case for ABAC implementation.* What are the costs of developing/ acquiring new capabilities and transitioning away from old capabilities? What are the important benefits provided by ABAC? What new risks, if any, are introduced by ABAC, and what new governance structures are required to manage shared capabilities and documentation of policies that were previously human-in-the-loop decisions? Which data sets, systems, applications, and networks need ABAC capabilities? How is liability for data loss or misuse of data managed?
- *Understand the operational requirements and overall enterprise architecture.* How are privileges managed, monitored, and validated for compliance? What interfaces and objects will be exposed by the enterprise for information sharing? What ACM will be used? How will subject and object attributes be shared

and managed? What are the access control rules and how are they captured, evaluated, and enforced? How is trust managed within the enterprise?

- *Establish or refine business processes to support ABAC.* Are access rules and policies fully understood and documented? How are required attributes identified and assigned? How are multiple policies applied in a hierarchy and deconflicted? How are access failures handled? Who creates new policies? How are common policies shared and managed?
- *Develop and acquire an interoperable set of capabilities.* How will interoperability be achieved? How are subject attributes from identity management integrated into ABAC? How are diverse or special needs for identities handled? How are subject attributes shared and maintained across enterprise entities? What are the trade-offs with centralization versus distribution of authentication, authorization, attribute management, decision, or enforcement capability? How are environment conditions considered in access decisions? How is confidence in security, quality, and accuracy measured, conveyed, and used in access decisions? How are subject attributes mapped between organizations? How are policies developed to incorporate the latest set of available subject, object, and environment condition attributes?
- *Evaluate performance.* How are subject attributes managed for disconnected and bandwidth-limited or resource-limited users? How available are interface specifications for new participants to the enterprise? How are quality and timeliness of changes to attributes and policies measured and enforced? Will overall system and end-to-end performance be adequate?

The following sections address these principles and questions in more detail.

9.3.1 Initiation Phase Considerations

During the initiation phase (see [Figure 9.4](#)), the organization evaluates the need for an ABAC system and its potential use. It should be determined whether the ABAC system will be an independent

information system or a component of an already-defined system. Once these tasks have been completed and a need has been recognized for ABAC capabilities, several processes must take place before the ABAC system is approved to include clearly defining goals and defining high-level requirements. During this phase, the organization defines high-level business and operational requirements as well as the enterprise architecture for the ABAC system.

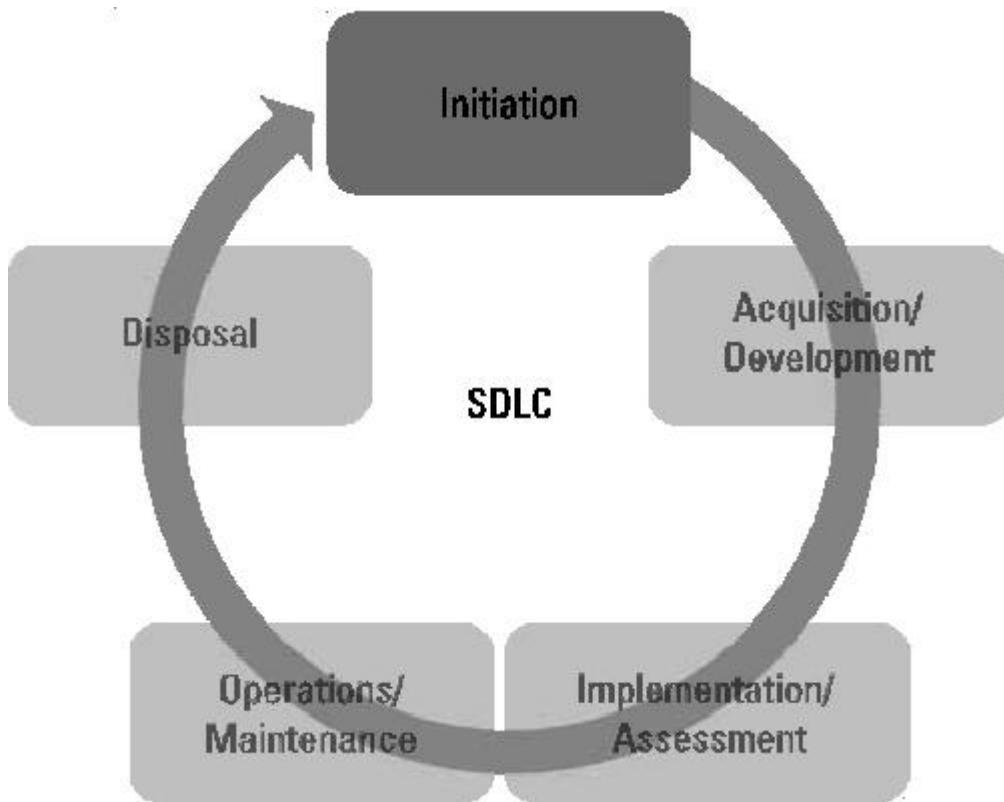


Figure 9.4 Initiation phase.

9.3.1.1 Building the Business Case for Deploying ABAC Capabilities

As with any major system deployment, the deployment of enterprise ABAC capabilities should be preceded by an evaluation of significant requirements, trade studies, and planning activities to include the determination of whether ABAC is the right type of access control capability needed and feasible given the application portfolio. ABAC has the virtue of providing access without prior knowledge of or information about the subject and large-scale enterprise information sharing of a limited set of mission or business-critical objects.

Before any technical requirements are generated or deployment decisions are made, it is important to evaluate and establish a business case for the deployment of ABAC capabilities as well as to define the scope of the enterprise targeted for these capabilities. Enterprise ABAC carries with it significant development, implementation, and operations costs as well as a change in the way enterprise objects are shared and protected. Case studies or experience reports from other organizations may be helpful in planning the ABAC deployment. It may be more practical to take an incremental approach and implement ABAC protections for a limited set of objects. This implementation would establish and utilize, to the maximum extent possible, policies and attributes appropriate for the enterprise as a whole. Feedback from incrementally building out this ABAC capability will refine policy and attribute definitions and exercise the governance and configuration management capabilities necessary to support broader ABAC use throughout the enterprise. It should be noted that without addressing the issues presented in the following subsections, an enterprise may incur significant delay and additional cost in its ABAC deployment.

9.3.1.2 Scalability, Feasibility, and Performance Requirements

Scalability, feasibility, and performance are important issues when considering the deployment of an ABAC product or technology. Enterprise ABAC—allowing an organization to have access to authorized objects managed by another organization in the same enterprise—requires complex interaction between ABAC components. Often these components are distributed throughout the enterprise across organization boundaries and sometimes on different networks. The larger and more diverse the enterprise, the more complex these interactions become. What may have been a simple request to access a document in a repository may now require a policy request from an enterprise service, multiple attributes from numerous logically and physically dispersed attribute sources, a third-party validation of the integrity of the object attributes bound to the document, and a decision made at one point in the enterprise while the enforcement of that decision occurs at a different point in the enterprise. Feasibility evaluation should check whether applications can support ABAC, whether natively or through third-party applications. All of these potential interactions have a performance cost

that must be evaluated when determining the scope of objects that may be shared through an enterprise ABAC implementation. To mitigate potential performance and scalability concerns, a variety of architectures should be considered. The distribution of ABAC components should take into account the underlying enterprise architecture and location of necessary data and objects to be shared. For instance, PDPs and PEPs may be deployed under the same administration.

Development and Maintenance Cost

While ABAC provides many important new features when deployed across an enterprise, the cost of development, deployment, and maintenance of ABAC may exceed its benefits in the long term. In addition, the cost of retrofitting applications to use ABAC is wholly separate from procuring, setting up, and maintaining an authorization infrastructure. It is possible that a large portion of maintenance savings will be offset by the cost of managing and maintaining subject attributes and the policies needed for ABAC as well as the additional system support required. The benefits of having more precise¹, consistent, and flexible security must be quantified and used to determine the right balance between cost of risk and cost of security. Given these considerations, ABAC is not the right solution for every access control problem but can prove viable for environments where subjects and objects carry a rich set of attributes and access decisions involve complex relationships among these attributes.

Cost of Transition to ABAC

The governance and business process changes that must accompany the shift to ABAC represent a significant transition to an approach where objects are controlled by enterprise-governed policies, enterprise-controlled attributes, and sometimes local control. These objects may now need to be associated with an additional set of characteristics that may not have been used in access control until now. Users accustomed to logging onto their network and having broad access to resources may no longer have that luxury. While policy makers will do their best to reflect current mission and business needs in policies, there will be unexpected but inevitable denials of access to those with critical mission or business functions.

As ABAC products are implemented and an organization's access control changes, new processes and capabilities will need to be integrated into the users' day-to-day business processes and enterprise policies. During the transition, it will be important to ensure that users understand why these access control changes are being implemented and what impact they will have on the way business is done. These users will need to be educated in the new ABAC systems and processes. These changes need to be properly communicated to show the benefits of an enhanced user experience, the enhanced security and safeguarding of critical information, the requirements of the new ABAC system, and the legacy access control systems, if replaced, that will be phased out. Users may be comfortable with existing processes and may not see an immediate value in switching to an ABAC capability. It may be important to emphasize areas in which ABAC enhances the security posture of the enterprise in contrast to areas where it complements existing access control mechanisms.

Need to Review Privilege and Monitor Authorizations

Some enterprises may desire the ability to review the capabilities associated with subjects and their attributes as well as the access control entries associated with objects and their object attributes. More succinctly, there are some requirements to know what access each individual has before the requests are made. This is sometimes referred to as a before-the-fact audit. A before-the-fact audit is often necessary to demonstrate compliance to specific regulations or directives. Another commonly desired review feature is determining who has access to a particular object or to the set of resources that are assigned to a particular object attribute. An ABAC system may not lend itself well to conducting these audits efficiently. Rather, a key feature of ABAC is the ability of the object owner to protect and share the object without any prior knowledge of individual subjects. Evaluating the set of subjects that have access to a given object requires a significant data retrieval and computation effort—possibly requiring every object owner to run a simulation of the access control request for every known subject in the enterprise. Limiting the scope of ABAC implementation can help in predetermining access authorizations, but other methods of ensuring the validity of access authorizations should be explored if the enterprise requires such validation.

Understanding Object Protection Requirements

Within the various parts of an enterprise there are a number of different operation and object types over which policy needs to be enforced. These may include operating systems, applications, data services, and database management systems. While some NLPs may exist to help determine authorized access, access to most objects is controlled through local group policy governed by local business rules, undocumented evaluation factors, and inherited nonstandard doctrine. Implementing ABAC requires, first and foremost, a thorough understanding of the objects and their protection requirements. Without that understanding, the cost to develop and implement the technology required for enterprise ABAC increases dramatically. It is recommended that enterprise ABAC implementations be initially applied to objects that are well defined, controlled, and documented.

Enterprise Governance and Control

Successful enterprise ABAC requires the coordination and determination of several business processes and technical factors as well as establishment of enterprise responsibilities and authorities. Without the proper governance model in place, organizations will develop stovepiped solutions and enterprise interoperability will be delayed significantly. It is recommended that an enterprise governance body be formed to manage all identity, credential, and access management capability deployment and operation and that each subordinate organization maintain a similar body to ensure consistency in managing the deployment and transition associated with enterprise ABAC implementation. Additionally, it is recommended that the governance body develop a trust model that can be used to illustrate the trust chain and help determine ownership and liability of information and services, needs for additional policy and governance, and requirements for technical solutions to validate or enforce trust relationships. The trust model can be used to help influence organizations to share their information with clear expectations of how that information will be used and protected.

Additionally, enterprise authorization services should be tightly integrated with security audits, data loss prevention, security configuration management, continuous monitoring, and cyber defense

capabilities. Authorization services alone are not enough to ensure the security needed to protect the mission-critical objects residing on networks. Efforts should be undertaken to fully understand enterprise security requirements and the impacts an ABAC implementation will generate. For example, when using a distributed ACM architecture, there may be consequences for the ability to audit access control decisions and events.

ABAC systems can benefit from deployment in environments governed by a trust framework. A comparison of representative trust chains for legacy access control list (ACL) use and ABAC use ([Figures 9.5](#) and [9.6](#)) shows that there are many more complex trust relationships required for ABAC to work properly. ACLs are established by the object owner or administrator, who ultimately enforces the object access rules by provisioning access to the object through the addition of a user to an ACL. In ABAC, the root of trust is derived from many sources such as subject attribute authorities or policy developers.

When managing the risk inherent in information sharing during the deployment of an enterprise ABAC solution, two perspectives of risk must be addressed. First, an ABAC solution may be considered one of many security control options that help protect an enterprise from risk. The risk of unauthorized access to protected resources can be reduced with an ABAC implementation because precise policies can be implemented consistently and updated more easily to address changing threats. Second, use of ABAC may increase or decrease the operational risks of an enterprise by exposing protected objects to access by unknown entities. Assuming that attributes are issued appropriately, the ABAC system is partially dependent on the attribute-issuing authorities. This multiplicity of risk sources presents a number of challenges that must be managed through governance and a formal trust model.

When establishing a governance model for managing the risks inherent in ABAC, it is important to ensure that there are mechanisms and agreements in place with each responsible organization to monitor and manage these roots of trust and any liabilities that occur as a result of unwarranted access.

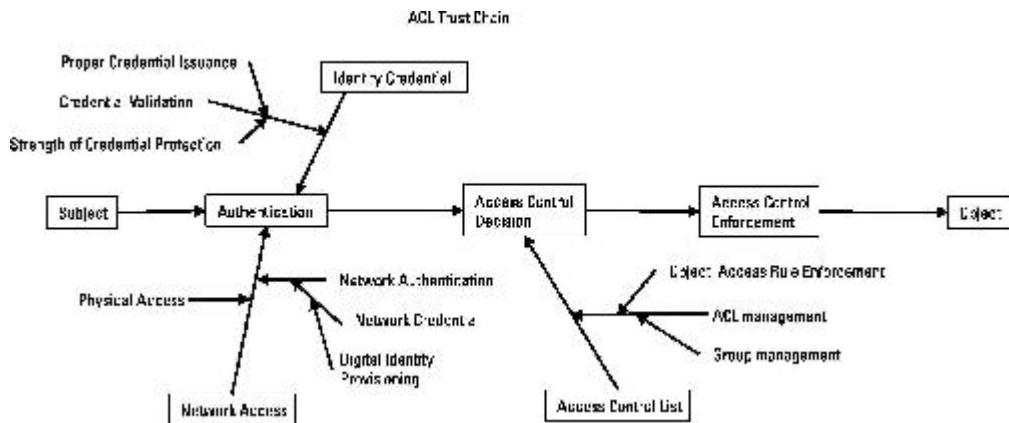


Figure 9.5 ACL trust chain.

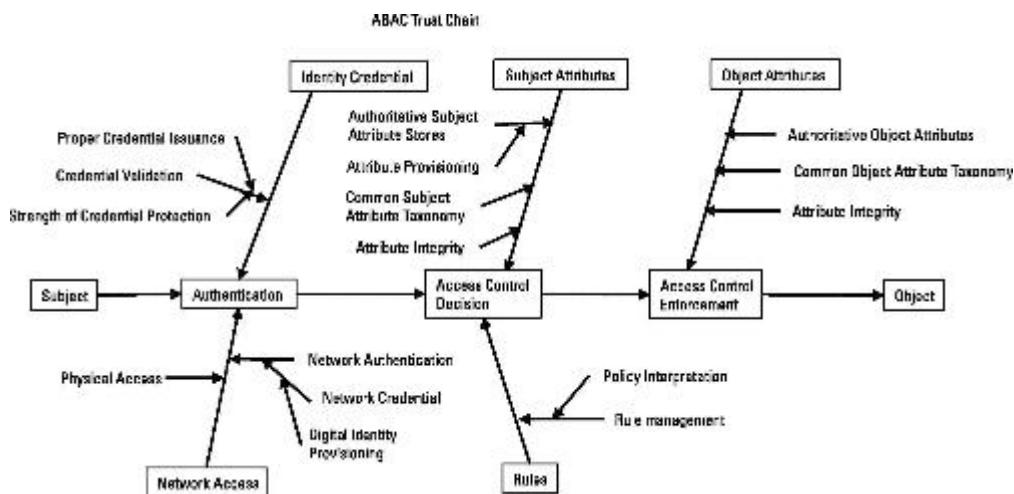


Figure 9.6 ABAC trust chain.

9.3.1.3 Developing Operational Requirements and Architecture

Several high-level operational and architecture planning requirements must be satisfied before implementing an ABAC solution:

- First, identify the objects that will be shared and protected by ABAC;
- Second, define the rules or policies that govern their protection;
- Third, identify and define the subject and object attributes as well as their associated authorities in coordination with the access control rule developers;
- Fourth, develop processes regarding how the access control policies are written, validated, and managed;
- Finally, determine how the ACMs will be segmented or distributed throughout the enterprise and how attribute, policy,

and decision requests and responses will be rendered.

Object Identification and Policy Assignment

The objects selected to be shared and protected by the ABAC solution will vary based on organizational requirements. Each object or class of object must be identified, and the policy or rules protecting each must be documented. A set of business processes need to be established to identify, classify, and assign policy to each new object created within the scope of the ABAC implementation.

Attribute Architecture

Access control policies are expressed in terms of attributes. Consequently, all required attributes must be established, defined, and constrained by allowable values required by the appropriate policies. The schema for these attributes and allowable attribute values must be published to all participants to help provide object owners with rule and relationship development. Once attributes and allowable values are established, methods for provisioning attributes and appropriate attribute values to subjects and objects need to be established along with an architecture for any attribute repositories, retrieval services, or integrity checking services. Interfaces and mechanisms must be developed or adopted to enable sharing of these attributes.

Subject Attributes

Many human subject attributes are typically provisioned upon employment with the organization and may be provisioned by several different authorities (e.g., human resources, security, organization leadership, etc.) For these, approaches to obtaining authoritative data are well known. As an example, only security authorities should be able to provision and assert clearance attributes and attribute values based on authoritative personnel clearance information; an individual should not be able to alter his or her own clearance attribute value. Other subject attributes may involve the subject's current tasking, physical location, and the device from which a request is sent. Processes need to be developed to assess and assure the quality of such subject attribute data.

Authoritative subject attribute provisioning capabilities should be appropriately dependable with regard to quality, assurance, privacy, and service expectations. These expectations may be defined in an attribute practice statement (APS). An APS provides a list of the attributes that will be used throughout the enterprise and may identify authoritative attribute sources for the enterprise. Still further network infrastructure capabilities (including the ability to maintain attribute confidentiality, integrity, and availability) are required to share and replicate authoritative subject attribute data within and across organizations.

Object Attributes

Object attributes are typically provisioned upon object creation and may be bound to the object or externally stored and referenced. It is to be expected that access control authorities cannot closely monitor all events. Frequently, this information is driven by nonsecurity processes and requirements. Good attribute data that supports good access decisions are essential, and measures must be taken to ensure that object attributes are assigned and validated by processes that the object owner or administrator considers appropriate for the application and authorization. For example, object attributes must not be modifiable by the subject to manipulate the outcome of the access control decision. The object attributes must be made available for retrieval by access control mechanisms for access control decisions. Additional considerations for creating object attributes include:

1. In general, users will not know the values of an object attribute (e.g., which sensitive compartment a given user is authorized). This should be accounted for in ACMs such that users only see the values that are applicable to them.
2. A schema is required for object attributes defining attribute names and allowed values.
3. Attributes need to be kept consistent in DP, MP, and NLP.

There have been numerous efforts within the federal government and commercial industry to create object attribute tagging tools that provide not only data tagging, but also cryptographic binding of the attributes to the object and validation of the object attribute fields to satisfy access control decision requirements.

Environment Condition

Environment condition refers to context information that generally is not associated with any specific subject or object but is required in the decision process. They are different from subject and object attributes in that they are not administratively created and managed, but instead are intrinsic and must be detectable by the ABAC system. Environmental conditions such as the current date, time, location, threat, and system status, are usually evaluated against current matching environment variables when authorizing an access request. Environment conditions allow ABAC policies to specify exceptional or dynamic AC rules that cannot be described by subject/object attributes alone. When composing ABAC rules with environment conditions, it is important to ensure that the environment condition variables and their values are globally accessible, tamperproof, and relevant to the environments where they are used.

Access Control Rules

In ABAC, all AC rules must include some combination of attributes and allowable operations. They may also involve conditions, hierarchical inheritance, and complex logic. Together, these provide a rich array of options when implementing ABAC. Rule sets and the application of rule sets to objects must be governed and managed appropriately. Rules must accurately and completely reflect the NLP, and be authoritatively developed (some by organizations, some by resource owners), applied, maintained, shared, and asserted. ABAC allows multiple rules from multiple stakeholders. New techniques are needed to coordinate and obtain the proper balance of sharing and protection. In some settings, one might limit the visibility of which rules apply to which objects to minimize the likelihood of unauthorized subjects manipulating attributes to obtain authorization. In other circumstances, subjects that are denied access should have a method, by which to verify or rectify the circumstances that caused the denial. Some organizations may wish to track the denials to see if the rules were appropriate. Similarly, rule definition and employment mechanisms and processes should include a robust rule deconfliction (i.e., resolution for the different decisions of rules) capability to determine rule conflicts and resolution processes.

Access Control Mechanism and Context Handling

The distribution and orchestration of ACM must be predetermined to avoid conflicts and weaknesses in object protection. For example, if an identical object is held by two different organizations, an unauthorized subject should not be able to access the version held by the organization with lesser restrictions. ACMs should be managed, maintained, and employed in a consistent manner to ensure interoperability and comprehensive security.

The order in which the ACM retrieves information, evaluates the situation, and enforces a decision can differ greatly based on the specific requirements of the implementation. It may even take into account environment conditions during access control decision rendering. This is known as context handling and simply refers to the workflow the ACM undertakes when gathering the data needed for a decision.

Additionally, where and how policy, attribute, and decision information are stored and exchanged throughout the enterprise is an important consideration for performance and scalability purposes.

9.3.2 Acquisition/Development Phase Considerations

During the acquisition/ development phase (see [Figure 9.7](#)), the system is designed, purchased, programmed, developed, or otherwise constructed. The organization prepares the business processes needed for enterprise-wide execution and defines the systems to be deployed and integrated. During the first part of this phase, the organization should simultaneously define the system's security and functional requirements. During the last part of this phase, the organization should perform developmental testing of the technical and security features/functions to ensure that they perform as intended prior to launching the implementation/ assessment phase.

9.3.2.1 Business Process Generation and Deployment Preparation

Documentation of Rules

For each of the types of objects controlled by an organization, there should be an accompanying set of access control rules documented in an

NLP. (Use cases might provide the easiest means for enterprise participants to define an NLP for a set of objects.) These rules should dictate who can and cannot create, view, modify, delete, forward, or interact with data and services controlled by the organization as well as under what context or environment conditions they have those privileges. The documentation of these rules incorporates the organization's interpretation of applicable policies as well as guidance, the specific sensitivities of applicable objects, and knowledge of appropriate user communities that will need the objects.

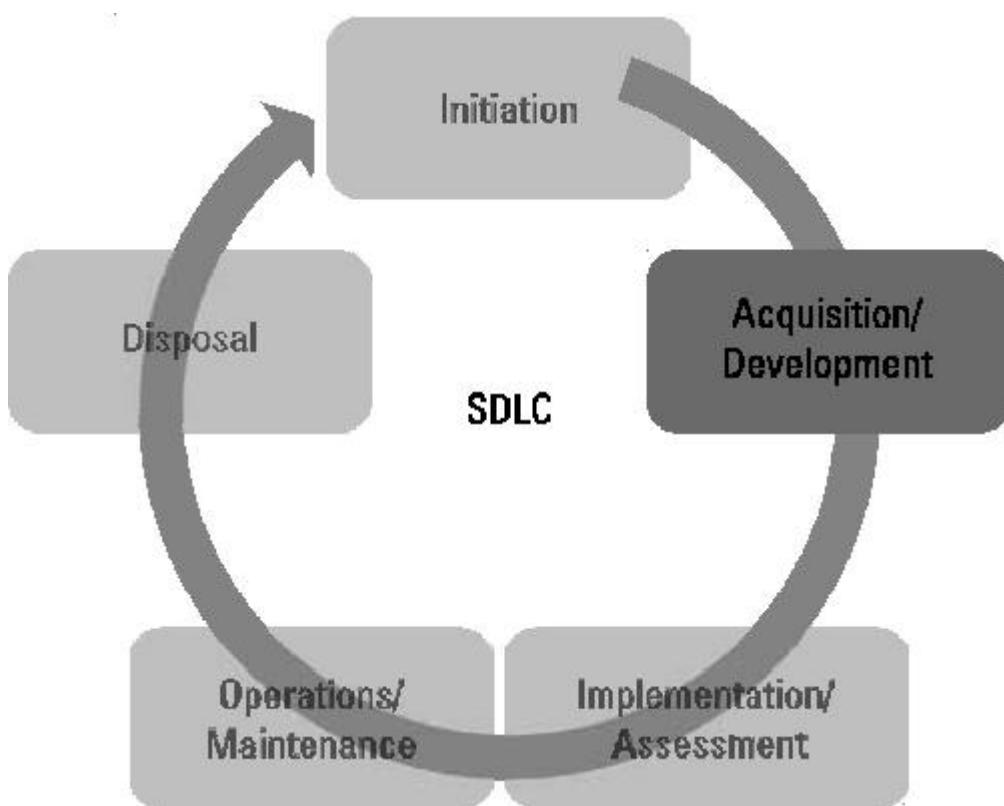


Figure 9.7 Acquisition/development phase.

Documenting NLP facilitates the development of DP and provides traceability back to the written policy. For example, many organizations have difficulties transitioning their authorization capabilities from ACLs into a more robust ABAC infrastructure because no corresponding NLP exists. As an example, consider that when a request for access is received, the data owner evaluates a set of criteria—usually undocumented—such as, “Is this person a member of the working group?” or “Am I familiar with this person or his or her organization?” The data owner then renders a decision before adding the requestor’s name to the appropriate ACL. Well-documented NLPs enable transition

from human generated decision making to a consistent, automated, policy driven access control decision.

Customizing Policy

Unless required by higher authorities or obligations, subordinate organizations should not make local policies less stringent. If subordinate organizations in an enterprise are able to independently relax the restrictions established for enterprise policy, the security inherent in the system is undermined, possibly allowing local access to enterprise objects where it would otherwise be forbidden.

Local access policies implemented in a federated enterprise should reflect the enterprise policy associated with the requested object based on mapped attributes from the requestor's organization. Depending on the sharing agreements between organizations, objects with shared ownership or control should be protected according to the most restrictive policy.

Agreement and Understanding of Attributes

A consistent set of valid values must be defined and applied for enterprise subject and object attributes. This allows authorization decisions to be based on known values that are consistent throughout the enterprise. The life-cycle management of attributes is the responsibility of the provisioning organization, whether the attributes are used exclusively within an organization or across organizations.

Understanding Meaning of Attributes

Attribute service providers need to describe attributes and their relationship with other attributes so that they may be properly and effectively used by consumers. Attribute service providers must document the definitions and meanings of enterprise authorization attribute values and provide guidance on the use of the attributes. In some cases, attributes must be used in combination with other attributes to establish a valid context, such as the combination of role and organization—a role has no meaning unless it is defined within the context of an organization. For example, the director of operations for an entire organization, whose responsibilities may include the finance,

human resources, legal, and other departments, has an entirely different contextual meaning from the director of operations within the Web services branch of the IT Department. Without the understanding of the guidance related to the attribute, its context, and the knowledge that these attribute values are required to render a decision, the DP—and hence the decision—may be generated on insufficient information or using faulty logic.

Processes and Procedures for Access Failures

A set of procedures and requirements for communicating exception handling, access denials, and errors should be established to provide users a means to remediate access decisions given mission, role, and need-to-know imperatives. As authorization services mature from the traditional method of provisioning an account and populating an ACL to an automated decision process, it will be more difficult for system users to understand and remedy access denials. A well-established process for properly discovering and obtaining the attributes needed for access approval will help ease the transition. This can be expanded to address dropped connections or other difficulties in accessing the authorization service component.

In a mission-critical role, the subject should be able to understand the limitations and request an exception, be pointed to an authoritative source of help, or attempt an alternate path to access equivalent information or services.

Attribute Privacy Considerations

ABAC capabilities should be developed to comply with all applicable privacy laws, directives, and policy. Due to the personal and descriptive nature of subject attributes, implementing attribute sharing capabilities may increase the risk of privacy violation of personally identifiable information (PII) due to inadvertent exposure of attribute data to untrusted third parties or aggregation of sensitive information in environments less protected than the originator's. Organizations engaged in attribute sharing should employ trust agreements to ensure the proper handling of PII and enforcement of PII regulations. These trust agreements should detail authorized PII use and handling for all components in the trust chain as well as methods for validating,

remediating, and adjudicating liability for regulatory infractions. A second consideration is that subject attributes can be revealed by the patterns of grant/ deny decisions. If a subject accesses a particular object, the subject must possess attributes as specified in the access rule for that object. The organization should protect access logs or other means of discovering grant/ deny decisions.

Digital Policy Creation and Maintenance

Each DP should be specified to satisfy the requirements of an NLP. DPs are sensitive and need to be protected in the same way as objects, according to an appropriate policy. These policies may pertain to creation and modification of specific portions of the DP. DPs should be written or modified only by individuals who can interpret NLPs and have authority to write the DP. Implementing a particular NLP may require specification of multiple DPs. Special consideration should be taken to ensure that subordinate policies do not conflict with higher level policies. Individual organizations should develop and maintain local policy and unique policy that applies only to their constituent or subordinate organizations.

9.3.2.2 System Development and Solution Acquisition Considerations

Standardization and Interoperability within the Enterprise

Implementers of ABAC should strongly consider using a comprehensive standards-based approach that enables current day interoperability and future deployment flexibility by making use of products or capabilities that meet these objectives. An established practice to achieve interoperability and cost-efficient ABAC deployments is to use a series of standards, specifications, and standardized configurations (specifying a subset of standard options; i.e., a profile). Standards that have optional elements may be implemented inconsistently by developers, making it possible for services or applications that are fully compliant with a standard to be noninteroperable. For this reason, well-defined and standardized profiles should be encouraged, especially in cross-organizational environments. When acquiring ABAC solutions, implementers should use commonly agreed-upon tailored profiles as well

as leverage the standards and profiles contained within existing standards registries.

Individual authorization service components (e.g., policy decision point, policy enforcement point, policy retrieval point, attribute retrieval point, meta-attribute retrieval point) should be developed with standard, open interfaces so that systems from multiple products can be employed while ensuring interoperability. Enterprises should consider a set of requirements addressing functionality, interfaces, infrastructure, and product support to employ as a filter within the procurement process for all acquisitions regardless of categorization or affiliation.

Identity Management Integration

A request for access to an object must be authenticated as originating from a unique subject. Authentication is achieved through use of identity credentials, and must occur before an access decision can be made. The ABAC system needs to support the prevalent and strategic authentication mechanisms and credentials used by the organization. This may mean the organization needs to make enhancements to its authentication infrastructure, if its current state impedes ABAC adoption. The subject attributes conveyed in these credentials should uniquely determine the subject, and the identity vetting process used to issue credentials should be sufficient to hold the identified entity accountable. The issuance and vetting processes should be recognized throughout the enterprise as trustworthy and sufficient to enforce accountability requirements. Strong authentication methods should be used that are of sufficient assurance for the request [5, 6]. Once the subject is authenticated, attributes associated with the subject can be used to determine an access decision, and access decisions can be captured in required audit records/ systems to provide attribution of the request. For example, a request transferred via a Transport Layer Security (TLS) 1.2 session with client authentication [7] depending on X.509 certificates issued by a trusted certificate authority is associated to the entity bound by the certificate authority to the distinguished name.

Support for Nonperson Entity

Support for nonperson entity (NPE) in access control services has special requirements. Authorization services use attributes associated with

entities in any form. The attributes bound to the NPE not only help define the unique NPE but also reflect the context of that entity within an organization.

In some cases, an NPE subject may be acting on behalf of one or more human subjects. These NPEs may carry their own identity credentials independent of any human subject. Note that the access control system basing an access decision on an NPE credential will not be able to attribute the request to the individual or individuals who may be acting in that role or are logged into the group account at the time of the request. NPEs may act either independently or on behalf of an authenticated individual. NPEs may include network devices (e.g., switches, routers), processes running on servers (e.g., portals), workstations, and other endpoint devices. As mission and security functions are increasingly automated, NPEs will play a larger role as actors in authorization service interactions.

Authentication and Data Integrity between ABAC Components

The authorization service requires strong mutual authentication between ABAC components (e.g., PEP, PDP) when authorization service components exchange sensitive information. For each exchange, proof of origin, data integrity, and timeliness should be considered. For example, when the authorization service needs to obtain attributes from an authoritative attribute service, mutual authentication should be used, followed by mechanisms for validating message integrity and message origin. Authentication protocols based on strong methods (e.g., X.509 authentication) should be used to provide the level of assurance needed by both parties involved in the attribute exchange.

Integrating Other Controls with ABAC

Authorization services alone are not enough to ensure the security needed to protect the mission-critical objects distributed throughout the enterprise. Comprehensive and cohesive security capabilities are needed to establish the desired level of assurance, and they must be tightly integrated and able to seamlessly feed the security information needed for making and enforcing access decisions. These other controls may include subject authentication, security audit, security configuration management, intrusion detection, and monitoring capabilities.

Selection and Accessibility of Attribute Sources

Authorities should be clearly identified so that the attribute source is able to provide attributes to the policy decision point from an authoritative source. When multiple attribute services are available, possibly with different metaattributes (such as *assurance level*), the attribute store/policy information point should balance the retrieval of attributes that satisfy the most restrictive policies with performance and availability requirements.

A Shared Repository for Subject Attributes

Direct use of shared repositories for subject attributes should be considered where there is sufficient network connectivity to take advantage of economies of scale, increased quality control, and standard interfaces. Another advantage of using shared attribute repositories is that they provide a single access point for data from multiple sources. Building and managing a connection to a single access point may be less complex than managing multiple connections. In some cases, limited connectivity, insufficient bandwidth, or intermittent connections may prevent authorization service providers from being able to use shared repositories reliably. It may be necessary to maintain local copies of data that cannot be continuously in sync with a shared attribute repository, and thus not have access to current data.

Minimum Attribute Assignments

In some enterprises, a minimum set of attributes may be defined. With a standard set of enterprise subject attributes and object attributes, DPs can more easily be developed and modified to reflect changes in policy. One example of where this approach applies is with classification and compartmentalization markings within classified networks. In most cases, an object cannot be placed on the network without proper marking, and access control policies are written to address the finite and well-known set of classification and compartmentalization markings.

Environment Conditions

Some systems required by policy, environment, or contextual information can be fed into the access control process. Examples include threat level, subject/ object location, method of authentication, or time of day. The environment conditions may change more rapidly over time than subject and object attributes.

Attribute Management

Authorities for assigning attributes should be clearly defined and consistent with an appropriate attribute policy. Some form of validation, integrity, and provenance mechanisms (to verify the completeness, allowable values, integrity, and change history of attributes) should be integrated into the system framework used to manage attributes.

NLP/DP Traceability

A comprehensive and coherent traceability between high-level enterprise written policy/NLP and low- level enterprise or local DP should be maintained by an appropriate authority. This will enable changes to written policy to be evaluated and subsequent DPs to be altered accordingly. With this policy traceability, the plethora of DPs resident in local organizations will be auditable, verifiable, and alterable given any change to requirements.

Rules or Policies Based on the Agreed Attributes

If an organization has an agreement with one or more organizations to use a defined list of attributes (some industry and use case-specific groupings of attributes are available today), the organization that owns the objects must ensure that it writes access control policies based only on those attributes. Every effort should be made to use any accepted common set of shared enterprise attributes, no matter how limited, to ensure basic interoperability if only to effect a limited secure information sharing capability. As new requirements arise, the enterprise may choose to introduce new enterprise attributes and rules for sharing them.

For example, the OASIS XACML Export Control-US (EC-US) and Intellectual Property Control (IPC) Profiles serve as examples of domain-specific standardized attributes with generally constrained attribute values. The EC-US Profile documents the attributes common to

access control decisions for the U.S. Department of Commerce Export Administration Regulations (EAR) and the U.S. Department of State International Traffic in Arms Regulations (ITAR).

Externalization of Policy Decision Services

It is common to implement PDPs as services that are separate from individual enterprise services and applications. Doing so removes the burden and expense of providing similar decision services for every enterprise service or application, since a single PDP can support multiple enterprise authorization services. Allowing authorization service providers to use PDP services that are provided by the larger enterprise or by the organization greatly simplifies service/ application development, saves money that would otherwise be spent on licensing, training, configuring, and deploying disparate instances of these services, and moves operations and maintenance away from individual programs.

9.3.2.3 Considerations for Other Enterprise ABAC Capabilities

When developing and implementing ABAC enterprise authorization capabilities, architects and program managers must keep in mind that there will inevitably be a long transition from the current access control methods in use now to the desired end state. As standards and technology mature, organizations will need to embrace concepts that enhance interoperability and promote higher assurance solutions while discarding proprietary, stovepiped solutions.

Confidence in Access Control Decisions

An access control decision is made by using the accurate, timely, and relevant data gathered from authoritative source(s) that are appropriate to the level of risk. Confidence in the access control decision depends upon timeliness, relevance, authority, and quality, reliability, and completeness of information used to compute the decision. Other factors in establishing confidence include identification and authentication processes (e.g., strength of authentication mechanism, identity vetting, credential issuance and proofing, attestation, source Internet Protocol [IP] address). When adopting a risk-based approach to ABAC, the factors discussed above should be taken into account.

Mapping Attributes between Organizations

Organizations may name attributes and attribute values differently. It may be important to implement solutions that provide attribute mapping between enterprise organizations to minimize the need for a special class of attributes called enterprise attributes. Attribute mapping serves as a translation between attributes or attribute values that are named differently. For example, one organization may use the name *Citizenship* and another may use the name *Nationality* to refer to the same set of attribute values.

In practice, cross-organizational ABAC may follow a collaborative approach outlined in the “Object Identification and Policy Assignment” and “Attribute Architecture” bullets of Section 10.2.1.3. This would allow each organization to make local decisions within a framework that provides assurance of appropriate control between organizations. When new policies are created, if policy authors create or designate their own attributes, policies may not be interoperable. Using preagreed attributes will make the policies more uniform and easily understood.

9.3.3 Implementation/Assessment Phase Considerations

In the implementation/ assessment phase (see [Figure 9.8](#)), the organization installs or implements the system, configures and enables system security features, tests the functionality of these features, and finally, obtains a formal authorization to operate the system. Most of the considerations during this phase are focused on optimizing performance and ensuring security features work as expected.

9.3.3.1 Attribute Caching

When an ABAC solution moves from the prototype/ pilot to deployment, attribute caching may be considered to enhance performance. Performance of the ABAC solution can be negatively affected if each access decision requires an across-the-network attribute request. This is especially apparent in low-bandwidth, high-latency environments.

In addition to performance issues regarding attribute caching, the organization may evaluate a trade-off regarding the freshness of attributes and the impact upon security. Attributes that are not refreshed

as often will ultimately be less secure than attributes that are refreshed in real time. For example, a subject's access privileges may have changed since the last refresh, but those updates will not be reflected in their available access privileges until the next refresh.

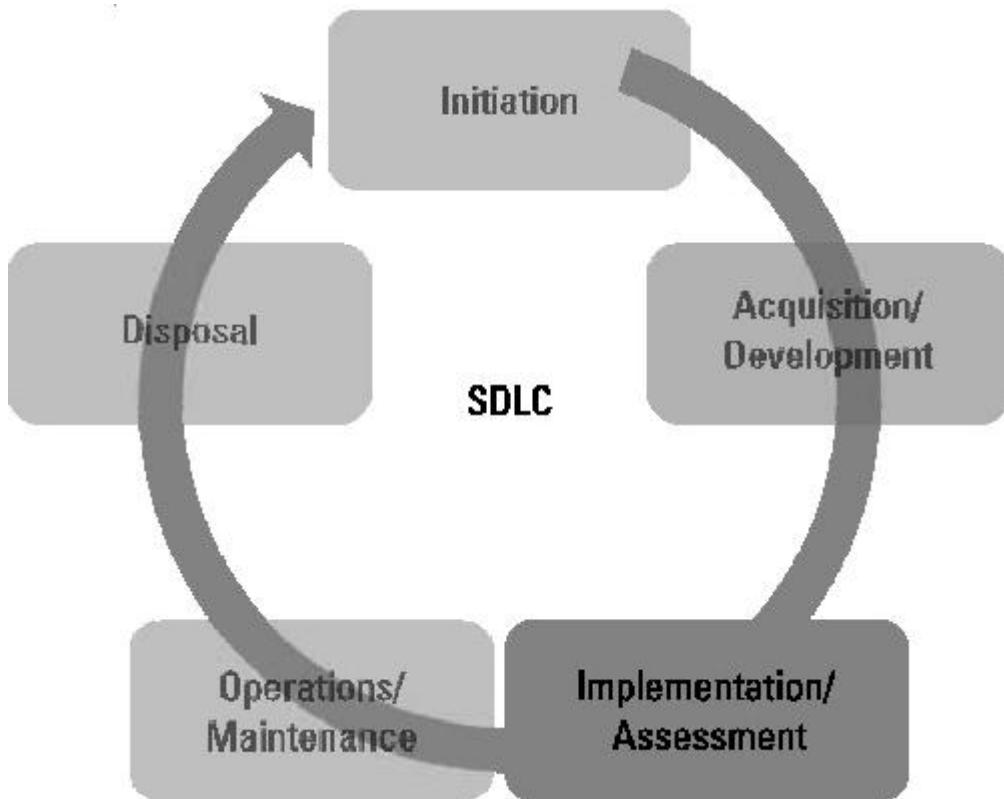


Figure 9.8 Implementation/assessment phase.

Environments with sporadic connectivity will need to cache attributes at the local level. The security ramifications of using cached attributes locally need to be determined within the implementing organization at a policy level and addressed with appropriate technical controls. In these disconnected environments, administrators may employ risk-based analysis as a basis for access decisions, as some attributes at the local (disconnected) level may change or be removed before the system refreshes its attributes. The local (and disconnected system's) possible use of stale cached attributes could introduce a level of risk to the system, because the local system is not making use of the most recently available attributes. Therefore, a risk-based analysis may be warranted as to whether or not to deploy this type of solution.

An example is a deployed ship with only intermittent, nonideal connections to enterprise network fabrics. Because the deployed user population will have only minor changes throughout their transit,

supporting the unanticipated system user is less of a concern. In this case, a bulk download and local storage of subject attributes may be sufficient for most local access control decisions. Therefore, subject attribute data could be stored locally on the ship throughout a deployment, and local applications and services could use the data from the local store without the need to reach to an authoritative enterprise attribute source. While this is one example of a solution to an austere environment problem, it should not be inferred that this is the only solution.

9.3.3.2 Attribute Source Minimization

Minimizing the number of attribute sources used in authorization decisions may improve performance and simplify the overall security management of the ABAC solution. Organizations planning to deploy an ABAC solution will benefit from establishing a close working relationship among all of the organization's stakeholders who will be involved in the solution's deployment.

9.3.3.3 Interface Specifications

To help ensure consistently reliable access to ABAC services, all organizations that participate in information sharing through enterprise ABAC capabilities should fully understand the interface, interaction, and precondition requirements for all types of requests, including attribute and DP requests. It is also important to ensure that as changes occur in the infrastructure and interface requirements, all relying parties are provided notification of updates so they can plan to modify their components accordingly.

9.3.4 Operations/Maintenance Phase Considerations

In the operations/ maintenance phase (see [Figure 9.9](#)), systems and products are in place and operating, enhancements and/ or modifications to the system are developed and tested, and hardware and/ or software is added or replaced. During this phase, the organization should monitor performance of the system to ensure that it is consistent with

preestablished user and security requirements, and needed system modifications are incorporated.

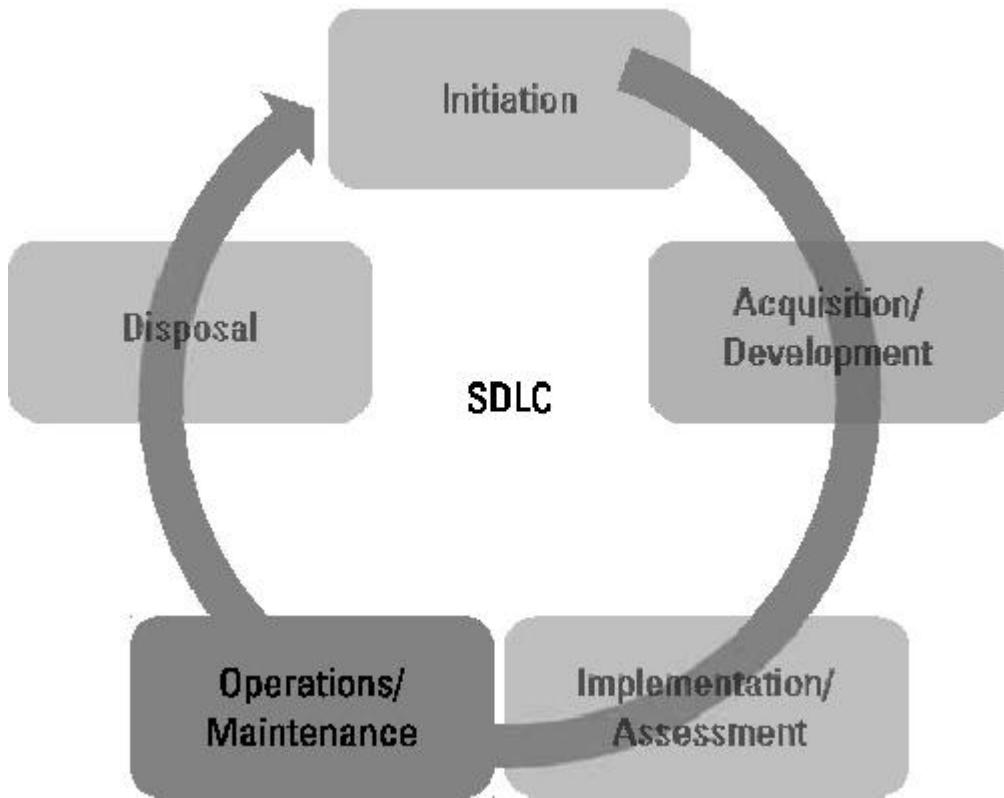


Figure 9.9 Operations/maintenance phase.

9.3.4.1 Availability of Quality Data

As the information needed to render access control decisions, and in some cases the decisions themselves, is externalized from the objects and consumers, access to information and services will become more dependent on an outside service's ability to provide timely and accurate data. It is important that the infrastructure be robust, well-tested, resilient, and scalable to mission needs. This is important to support attribute services, attribute stores, policy stores, policy and attribute generation and validation components, decision engines, and metaattribute repositories and conduits through which this information must pass. If outsourced, service agreements should detail availability, response time, and data quality and integrity requirements. For example, failover, redundancy, and continuity of operations must be considered for data and services that are considered mission-critical. Maintaining high availability of quality data requires that addition, updating, and deleting

of attribute values is performed by trained, authorized individuals, and regularly audited.

9.3.4.2 Agreements

Formal agreements between providers and consumers of attributes and services should meet an appropriate standard of service, quality, availability, protection, and usage. Various laws and regulations establish responsibilities, liabilities, and penalties related to the appropriate protection of information. The agreements should capture these requirements as well as those related to responsibility for data.

Agreements establishing an appropriate level of trust between organizations are important. These agreements would serve to formalize that trust relationship with a series of requirements and, possibly, penalties for nonconformance. APSs and memorandum of understandings (MOUs) /memorandum of agreements (MOAs) for attribute services and authoritative and accountable attribute sources can also serve to translate organizational policy into operational procedures. The purpose, usage, participants, responsibilities, and administration of these services are described in these formal agreements.

9.4 Conclusion

ABAC capabilities will allow an unprecedented amount of flexibility and security while promoting information sharing between diverse organizations. It is vital that these capabilities be developed and deployed using a common foundation of concepts and functional requirements to ensure the greatest level of interoperability possible. ABAC is well suited for large enterprises. An ABAC system can implement existing RBAC policies and support a migration from role-based to a more granular access control policy based on many different characteristics of the individual requester. It supports the external (unexpected) user and provides more efficient administration. However, an ABAC system is more complicated and therefore more costly to implement and maintain than simpler access control systems.

This chapter brings to light numerous considerations including advantages and common pitfalls of ABAC mechanisms aligned to the system development life cycle that must be factored into the planning,

design, development, implementation, and operation phases of ABAC capabilities within a large enterprise.

References

- [1] Hu, V. C., et al., *Attribute Based Access Control Definition and Consideration*, NIST SP 800-162, Gaithersburg, MD: National Institute of Standards and Technology, January 2014.
- [2] Hu, V. C., and K. Scarfone, *Guidelines for Access Control System Evaluation Metrics*, NIST IR 800-7874, Gaithersburg, MD: National Institute of Standards and Technology, September 2012.
- [3] Hu, V. C., D. F. Ferraiolo, and D. R. Kuhn, *Assessment of Access Control Systems*, NISTIR 7316, Gaithersburg, MD: National Institute of Standards and Technology, September 2006.
- [4] Bowen, P., J. Hash, and M. Wilson, *Information Security Handbook: A Guide for Managers*, NIST Special Publication 800-100, Gaithersburg, MD: National Institute of Standards and Technology, October 2006 (including updates as of March 7, 2007), 178pp. <http://csrc.nist.gov/publications/PubsSPs.html#SP-800-100>.
- [5] Burr, W. E., D. F. Dodson, E. M. Newton, R. A. Perlner, W. T. Polk, S. Gupta, and E. A. Nabbus, *Electronic Authentication Guideline*, NIST Special Publication 800-63-1, Gaithersburg, MD: National Institute of Standards and Technology, December 2011, <http://csrc.nist.gov/publications/PubsSPs.html#SP-800-63--1>.
- [6] Burr, W. E., D. F. Dodson, E. M. Newton, R. A. Perlner, W. T. Polk, S. Gupta, and E. A. Nabbus, *Electronic Authentication Guideline*, NIST Special Publication 800-63-2, Gaithersburg, MD: National Institute of Standards and Technology, August 2013, <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf>.
- [7] Dierks, T., and E. Rescorla, *The Transport Layer Security (TLS) Protocol*, Version 1.2, RFC 5246, Internet Engineering Task Force, Network Working Group, August 2008, <http://www.ietf.org/rfc/rfc5246.txt>.

-
- * Most of content in this chapter is derived from *NIST SP 800-162—Attribute Based Access Control Definition and Consideration*, by Hu, V. C., et al., January 2014.
 - 1. Attributes and rules allow more precision through a larger number of discrete inputs into an access control decision, providing a larger set of possible combinations of those variables to reflect a larger and more definitive set of possible rules, policies, or restrictions on access. ABAC allows a large number

of attributes to be combined to satisfy any access control rule imaginable. As long as the attributes are available to evaluate at the time the access decision is rendered, the rule can be as complex and definitive as it needs to be to satisfy the protection requirements of the object. Thus, fine-grained AC allows access to be more detailed or flexibly partitioned when compared with coarse-grained AC, for example: coarse: *employees* can read file X, fine: *employees* working on project A can read file X, and finer: *employees* working on project A during *office hours* can read file X.

10

ABAC in Commercial Products

10.1 Introduction

ABAC product offerings provide externalized authorization capability to one or more applications and databases. Externalized authorization capability provides access control for application modules or database content outside of the application or database management system (DBMS). In other words, the access control functionality is not an integral part of the application or the DBMS. A review of the overall capabilities of various ABAC products reveal the following four canonical features:

1. *ABAC policy modeling* provides the language and user interface to build policy components and entire policies. This feature thus plays the role of PAP in the XACML Reference Architecture.
2. *ABAC policy evaluation* provides the engine that receives access request, retrieves the appropriate policies from policy stores, attributes through attribute connectors, and either renders a YES or NO access decision or modifies the request appropriately for sending it to the Resource Server. Thus this feature plays the role of PDP.
3. *Attribute retrieval*, which uses software development kits (SDKs) and drivers to connect with various attribute repositories such as LDAP, Active Directory, databases, HR systems, security appliances and mobile device management (MDM) systems. This plays the role of PIP.
4. *Access Mediation*, enabled using a custom PEP for the application, a PEP built using a SDK provided by PDP vendor,

or simply using a gateway or proxy positioned in the network pathway to the application. This plays the role of PEP.

The approach adopted in this chapter for describing various ABAC products is as follows. The first task is to preview the architecture of each product in terms of its constituent modules.

The next task in our approach is to describe the capabilities of each product's module in the context of each of the four canonical feature (listed above). Since each canonical feature is associated with a functional role in the XACML Reference Architecture, the product module's functional role (e.g., PAP or PDP) is also automatically identified as well. In addition to the four canonical features, we also devote a paragraph for describing the overall architecture under the subheading Overall Design Feature(s). One piece of information that is captured under this topic is the level of integration between the various product modules—either loosely coupled or tightly integrated. This information is useful to the reader in order to assess the value of the ABAC product and the kind of system integration tasks it may need for deployment.

In summary, the approach consists of

- Presenting an overview of each product either in terms of its functional architecture.
- Describing the capabilities of each of the product's modules in the context of the four canonical features.
- Briefly highlighting the overall design feature in terms of the nature of integration between product modules, and between modules and the application to be protected.

The overall objective of the approach is to highlight the unique features of each product and technical considerations for its deployment. All information pertaining to each product's modules are obtained from publicly available documents such as product vendor's data sheets, white papers, user/ admin documents, and user blogs, and no proprietary information with respect to product or vendor was used in this chapter.

Out of scope: Each of the commercial ABAC products also comes with one or more administrative modules. The functionality of these administrative modules is not covered in this chapter as the focus is on ABAC deployment features provided by these products. While the

overall design feature in terms of the nature of coupling between modules and with the protected application is also highlighted for each product, configurations relating to performance and availability such as using multiple instances of a product's module in a cluster configuration for high availability, redundancy, and failover recovery are beyond the scope of this chapter.

10.2 Axiomatics Data Access Filter

The traditional access to database contents through a three-tier architecture (consisting of user interface and application and database layers) is now being increasingly replaced with direct access to databases through APIs because of the varied sources such as mobile devices and cloud gateways from which database access is required. Further, the data flows extend beyond traditional enterprise boundaries to outside entities such as business partners, collaborators, and suppliers. Hence Axiomatics's product goal is to provide data-centric security by enabling efficient information sharing. Meeting this goal requires fine-grained access control to database contents driven by a centralized set of access control policies. Axiomatics has chosen the ABAC model to express these policies using the XACML language and their product is called Axiomatics Data Access Filter. In addition to data access filtering, the product also supports policy-driven data masking and unmasking of database contents for sensitive data elements.

10.2.1 Product Architecture and Modules

Axiomatics Data Access Filter (ADAF) [1] applies access policies to database requests to ensure that only authorized individuals are allowed to read and write data, and to further protect this data by dynamically masking or redacting it so that sensitive data is never exposed. The overall architecture of ADAF MD 1.6 version is given in [Figure 10.1](#).

The ADAF product has the following modules (with associated functions):

- *Axiomatic Policy Server* serves as policy editor for creation and modification of XACML policies.

- *SQL Proxy Service* performs database activity monitoring to intercept calls (access requests) to the database. It is also called the SQL enforcement component of ADAF. It uses the SQL filter services to achieve data access filtering and dynamic data masking.
- *SQL Filter Service* evaluates access requests forwarded by SQL Proxy Service and evaluates them against the enterprise ABAC policies encoded in XACML. It is also called the SQL authorization service of the ADAF product.

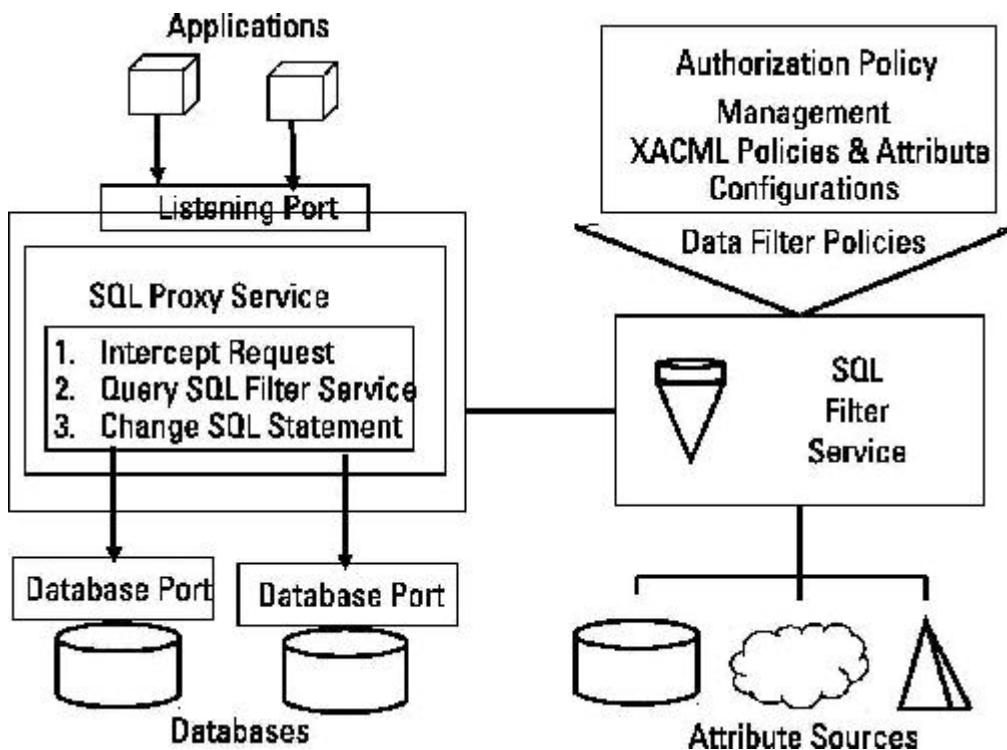


Figure 10.1 Axiomatics Data Access Filter.

10.2.2 Canonical Features in Product Modules

ABAC policy modeling: The Axiomatic Policy Server Version 6 (APS 6) [2] is the module that is used to author ABAC authorization policies and hence plays the role of PAP. It provides a Web-based graphical user interface for XACML policy creation and editing, including support for the REST and JSON profiles of XACML. The support for the concept of attribute namespaces enables segregation of policies by projects as well as enables collaboration between them in policy writing.

ABAC policy evaluation: The SQL Filter Server performs two functions. It has an authorization service that determines the response to

database queries by evaluating them against applicable policies and by querying attribute sources for attribute values required by those policies. It thus plays the role of PDP. As part of the results of access policy evaluation, the SQL Filter Server provides information for the SQL Proxy Server to modify the SQL statement if required.

Attribute retrieval: Recognizing the fact that attributes needed for ABAC policy evaluation can come from multiple sources, Axiomatics provides connectors to connect diverse attribute sources such as directories (LDAP or Active directory) and databases (HR systems or CRM). The connectors thus provide the PIP interfaces.

Access Mediation: The PEP has to be customized for each application and there is no gateway-like solution where a single PEP can provide access control service for multiple applications. The component that plays the PEP role in Axiomatics product is the SQL Proxy Server. This module receives the result of evaluation of access policies from the SQL Filter Server and modifies the SQL statement (if required) before forwarding it to the data resource server. Data access filtering is the main function of the product, and so this SQL Proxy Server is positioned before the data layer to enable it to intercept database SQL queries and forward it to SQL Filter Server. The SQL Proxy Server can provide this service for most common database engines such as Oracle, MS SQL Server, IBM DB2, and Teradata. It can also be configured to apply ABAC policies on all incoming requests or just a specific set of SQL statements.

Overall architecture: In the ADAF product, the SQL Proxy Service has to intercept SQL queries as well as send modified/ changed SQL statements to the DBMS application in order to ensure that the access control policies are not violated while enabling an access request. The logic for the modification is provided by the SQL Filter Service that has to use database schema elements as attributes (specifically as resource attributes) in its access evaluation (in addition to subject and environmental attributes). Due to this dependency, the modules SQL Proxy Service and SQL Filter Service (playing the roles of PEP and PDP, respectively) are tightly coupled/ integrated in Axiomatic's ADAF product. Both these modules are then loosely coupled with the DBMS application. The advantage is that the SQL Proxy Service can provide this service for multiple DBMS applications, thus minimizing the effort needed for integrating a customized PEP for every DBMS application.

The consequence of this architecture is that data access enforcements cannot be as granular as in environments where the PEP is tightly integrated with DBMS application.

10.3 Jericho Systems EnterSpace 9

Jericho System's EnterSpace 9 with DLDS Datasheet product enables enforcement of ABAC-based policies by integrating with a variety of systems. In addition, it provides the capability to mask (encrypt), redact (remove), and anonymize (substitute) on structured documents and files.

10.3.1 Product Architecture and Modules

Jericho's EnterSpace ABAC product [3] is designed to provide ABAC support for different types of applications. It runs on Linux and Windows platforms. The overall architecture of EnterSpace product is given in [Figure 10.2](#) with the following modules.

- *Jericho Authorization Provider (JAzP)* integrates with multiple applications to intercept access requests and forward the request to EnterSpace Decisioning Service (ESDS).

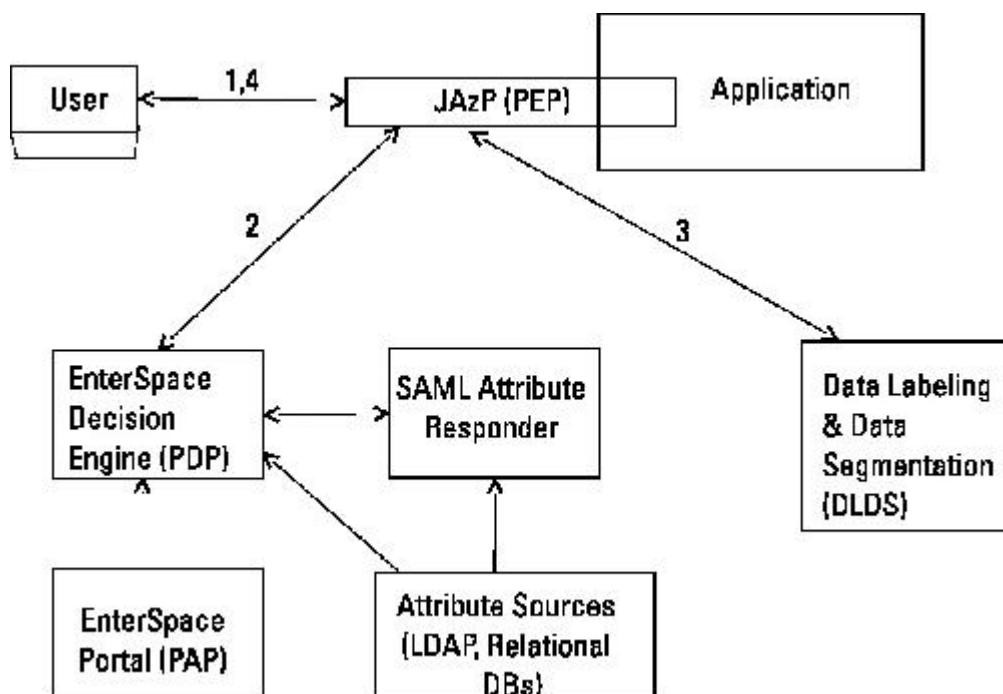


Figure 10.2 Jericho Systems EnterSpace with DLDS.

- *ESDS* has features to play the role of a PAP that can import and debug XACML policies as well as play the role of PDP.
- *SAML Attribute Responder* enables deployment of the OASIS SAML 2.0 standard-based attribute authority.
- *Data labeling and data segmentation* (DLDS) is an enhancement module that enables additional data filtering through masking (encrypting), redacting (removing), and anonymizing (substitution) based on previously assigned security metadata tags (top secret, sensitive but unclassified, etc.) to meet compliance needs such as privacy in data sharing environments.

10.3.2 Canonical Features in Product Modules

ABAC policy modeling: The EnterSpace Portal is the EnterSpace 9's [4] product module (and the policy editor within it) for creating and managing policies and thus plays the role of PAP. It provides two editing modes (simple mode and advanced mode) for performing this function.

- The simple mode is a graphical user interface (GUI) with drag-and-drop widgets for building rules and policies and features for displaying rule expressions in the form of a logic tree. The policy modeling capabilities include (a) the ability to create a resource hierarchy that allows policy inheritance (thus enabling large number of objects to be brought under access protection with minimal specifications (e.g., all files in a folder)) and (b) the ability to use the authentication mode (password or two-factor authentication, etc.) as an attribute in policies (thus enabling different access scope based on the type of authentication used by the user for that session).
- The advanced mode is a text editor in which entire XACML policies/ policy sets can be created/ changed as well as (bulk) imported for latter modification (if necessary).

ABAC policy evaluation: Jericho's EnterSpace Decision Engine is the policy engine that plays the role of PDP.

Attribute retrieval: Just like other ABAC products, Jericho's EnterSpace provides built-in connectors to common attribute sources such as LDAP directories, Active directory, digital certificates, and several different DBMSs for retrieving attributes needed for policy

evaluation. An API is also available for building custom connectors to other attribute sources. In addition, Jericho provides a separate product called SAML Attribute Responder for establishing a standards-based attribute authority that can support queries in the form of SAML Attribute Query, retrieve attributes from several different attributes sources referred above, and provide the EnterSpace Decision engine with attributes in the form of SAML assertions conforming to OASIS SAML v2.0 standard.

Access Mediation: Jericho's JAzP module plays the role of PEP that can be integrated with many different applications such as Microsoft Office Professional Suite (Access, Excel, Powerpoint, Outlook, and Word), Microsoft Sharepoint 2011, Web Services (SOAP/REST), and PKI applications (CRL checking, OCSP and digital signature validation).

Overall Architecture: Because the Jericho architecture requires a close coupling of the PEP and the application, it allows for the creation and enforcement of highly granular ABAC policies. The architecture supports a SAML attribute responder that can supply attributes in standardized SAML format not only for Jericho's EnterSpace Decision Engine (PDP) but also for Jericho's JAzP module (PEP) as well as for the accessing user.

10.4 NextLabs ABAC Solution

The design goal of NextLabs' ABAC solution is to prevent data breaches by enhancing access control to applications and data.

10.4.1 Functional Architecture and Components

NextLabs' ABAC solution has the following components (see [Figure 10.3](#)).

- *Business policy management.* The collection of modules that perform this functional component is called the NextLabs Control Center. The control center has the modules to build policy components and policies as well as deploy the latter and supports a proprietary language called Active Control Policy Language (ACPL) for modeling policies. It also supports the development of a policy component model, whose elements can be used as

building blocks for formulating access control policies. The Control Center Policy Studio–Policy Author and the Control Center Policy Manager are the two modules that come under this component and functionally have the role of PAP.

- *Attribute management.* The primary module in this component is the enrollment manager, which pulls in attributes (playing the PIP role) from a variety of common (e.g., enterprise directory) and custom (e.g., CRM application) attribute sources. In addition, it has tools for performing attribute mapping (to connect properties defined in the policy component model to attribute classes in different attribute source systems) and for building an attribute inventory with normalized attribute names (for attributes from disparate source systems that may have different attribute naming conventions).
- *Business policy evaluation.* The Control Center module that performs business policy evaluation (PDP role) is the policy controller. There are different policy controller versions for different platforms on which the application to be protected resides: in-house server-based, desktop and cloud-hosted, which are called the server policy controller, endpoint policy controller, and policy controller service, respectively. The provisioning of policies that is appropriate for each platform from the central policy repository (called policy pack).
- *User- and data-centric enforcement.* Next Labs entitlement manager is the module that performs the function of enforcing access control for a wide variety of application platforms. For those application platforms where out-of-the-box support is not available, SDKs are provided for different language environments.

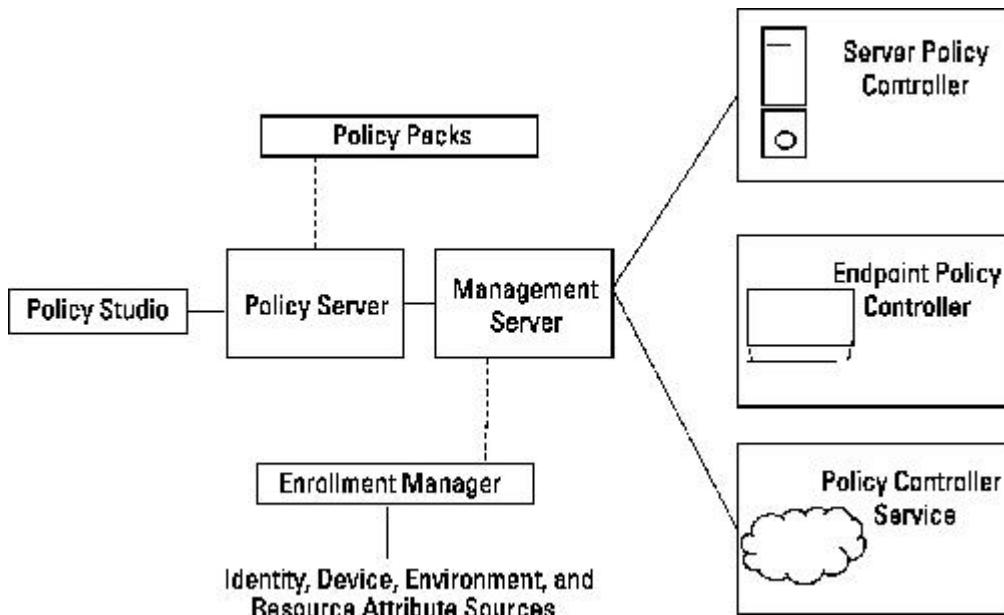


Figure 10.3 NEXTLABS policy modeling and evaluation modules.

10.4.2 Canonical Features in Product Modules

ABAC policy modeling: NextLabs Control Center module suite has its own proprietary language called Active Control Policy Language (ACPL) for modeling ABAC policies and providing capabilities to build a policy component model. The policy component model consists of user-defined user and data properties (e.g., user clearance level, document classification), which can be combined to build a nested subcomponent that in turn can be combined to develop a top-level component. These components can be used as building blocks to develop policies. NextLabs Control Center provides two modules to create and manage ACPL policies. The Control Center Policy Studio—Policy Author module enables policy developers to drag and drop policy components into policy templates using a drag-and-drop graphical user interface. The Control Center Policy Manager module enables delegation of policy administration functions to different operating units in the organization (business units, departments, or teams) and apply workflow rules to manage, deploy, and audit ABAC policies.

ABAC policy evaluation: Policy controller is NextLab's Control Center module that performs policy evaluation and plays the functional role of a PDP. In order to ensure that policy controllers receive policies from legitimate policy stores and that the policy stores send policies to legitimate policy controllers, a feature for bidirectional authentication

between each policy controller and policy server (that creates and manages policies) using digital certificates is provided. Further, to ensure the integrity of the ABAC policies being evaluated, all policies sent from policy servers to policy controllers are encrypted and digitally signed.

Control Center offers flexibility with respect to location of policy controllers. For protecting on-premise server applications (Sharepoint, SAP, file servers), policy controllers are generally colocated to minimize network traversals. Policy controllers can also be installed on endpoint devices for protecting desktop systems and can be configured as a cloud service to provide access decisions for cloud-based applications.

Attribute retrieval: Just like other ABAC products, Control Center provides in-built (out-of-the-box) connectors to common (identity) attribute stores such as LDAP directories, active directory, Microsoft Sharepoint server, HR applications, CRM applications and asset management systems which thus become PIPs.

In addition, Control Center provides two features for managing attributes—attribute mapping and attribute inventory. Recognizing the fact that properties (and components built using properties) defined in the policy component model are the building blocks of the policies and that instantiation of policy rules require values of attributes from different attribute source systems (directories, databases, and application), Control Center's attribute mapping feature helps to establish connection between these properties to attribute classes in those source systems. Also, being cognizant of the fact that the integrators who perform attribute mapping do not own nor are familiar with the internal semantics of those attributes, the tasks expected of the integrators is just to know the location of each attribute store and the logic used to retrieve attributes in order to link the properties to attribute classes.

In many situations, establishing ad hoc connections between currently defined properties of the policy component model and the attribute classes in attribute stores may not sufficient for maintaining robust, adaptive policies. Again recognizing the fact that names, semantics, and formats of attributes in diverse attribute sources, such identity and access management (IAM) systems and different LOB applications (e.g., document libraries in Microsoft Sharepoint server) vary widely, there is the need to define a common structure and normalize attributes from these disparate source systems. Control Center's attribute inventory feature performs this function and in

addition has capability to set up and receive notifications when any additions, changes, or deletions take place in the attribute source systems.

Access mediation: The design goal of Control Center is to have policy evaluation performed in the fastest time possible via a closed coupling to the application for achieving this goal. This is the reason for having different versions of policy controller (the module that plays the role of PDP in the NextLabs Control Center product) for servers, desktops, and cloud services. This goal in turn automatically requires the module that intercepts the access requests on behalf of the application to be tightly coupled to the policy evaluation module (in our case the policy controller) as well. The net result is the availability of SDKs for multiple platforms such as Windows, Linux, and Unix and to provide APIs for common programming languages such as C#, C++, Java, .NET, and web services so as to facilitate integration with a variety of custom and off-the-shelf commercial applications. The same SDKs enable integration with applications that perform custom actions, called obligations, which need to be invoked following a policy evaluation and access enforcement. An example is the need to update the workflow state (and constituent task status) as a consequence of an access event.

Overall architecture: As already mentioned, the Control Center's design goal with respect to policy controller (that plays the PDP role) has resulted in a tight coupling between the modules or executables that play the role of PDP and PEP, as well as both of them with applications. This enables the product to enforce a highly fine level of granularity in access restrictions at the cost of building a one-to-one PEP/PDP implementation for each application.

References

- [1] <https://www.axiomatics.com/solutions/products/authorization-for-databases/197-axiomatics-data-access-filter-adaf.html>.
- [2] <https://www.axiomatics.com/blog/entry/announcing-aps-6-0-the-industry-s-first-web-based-graphical-ui-for-xacml-policy-creation-and-editing.html>.
- [3] <https://www.jerichosystems.com/assets/misc//ES-9-Data-Sheet.pdf>.
- [4] <https://www.jerichosystems.com/products/decisioning9.html>.
- [5] <https://www.nextlabs.com/products/control-center/>.

11

Open Source ABAC Implementations: Architecture and Features

11.1 Introduction

The ABAC model used in the ABAC deployment in various applications ([Chapter 8](#)) and in ABAC commercial products ([Chapter 10](#)) for expressing access control policies is based on first-order logic as its theoretical foundation. Hence the implementation of the model in these proposals and products is in XACML ([Chapter 4](#)), the XML vocabulary used for encoding access control policies expressed in first order logic.

However, there is another class of ABAC model that can express the policy requirements using flat binary and ternary relations involving user (subject) and object (resource) attributes. This type of ABAC model is known by the research community [1] as enumerated ABAC model to distinguish it from the logic-based ABAC model implemented using XACML. The enumerated ABAC model forms the conceptual foundation in all open-source ABAC implementations such as NGAC/ PM Reference Implementation Harmonica [2] and Medidata Policy Machine [3], which are both available in github. In this chapter, only the features of NGAC/ PM Reference Implementation Harmonica (hereafter referred to as NGAC PM) are described with reference to the its underlying model, which we refer to as the NGAC ABAC model. The NGAC ABAC model is covered in [Chapter 5](#) and hence only references are made to that chapter while describing of the features of NGAC PM in the context of its model's capabilities. Further, a detailed description of the NGAC/ PM implementation is available through a technical publication [4], and the NGAC ABAC model is an ANSI standard with detailed specifications outlined in relevant standards documents [5] and [6].

The overall organization of this chapter is as follows:

- NGAC PM: Functional Architecture ([Section 11.2](#));

- NGAC PM: ABAC Model Definition capabilities ([Section 11.3](#));
- NGAC PM: Access Decision Process ([Section 11.4](#));
- NGAC PM: Design and Application Integration ([Section 11.5](#));
- Summary and Analysis ([Section 11.6](#)).

11.2 NGAC PM: Functional Architecture

The NGAC PM functional architecture comprises of the following components [5], as shown in [Figure 11.1](#).

- One or more policy enforcement points (PEPs);
- One or more policy decision points (PDPs);
- Zero or one event processing point (EPP);

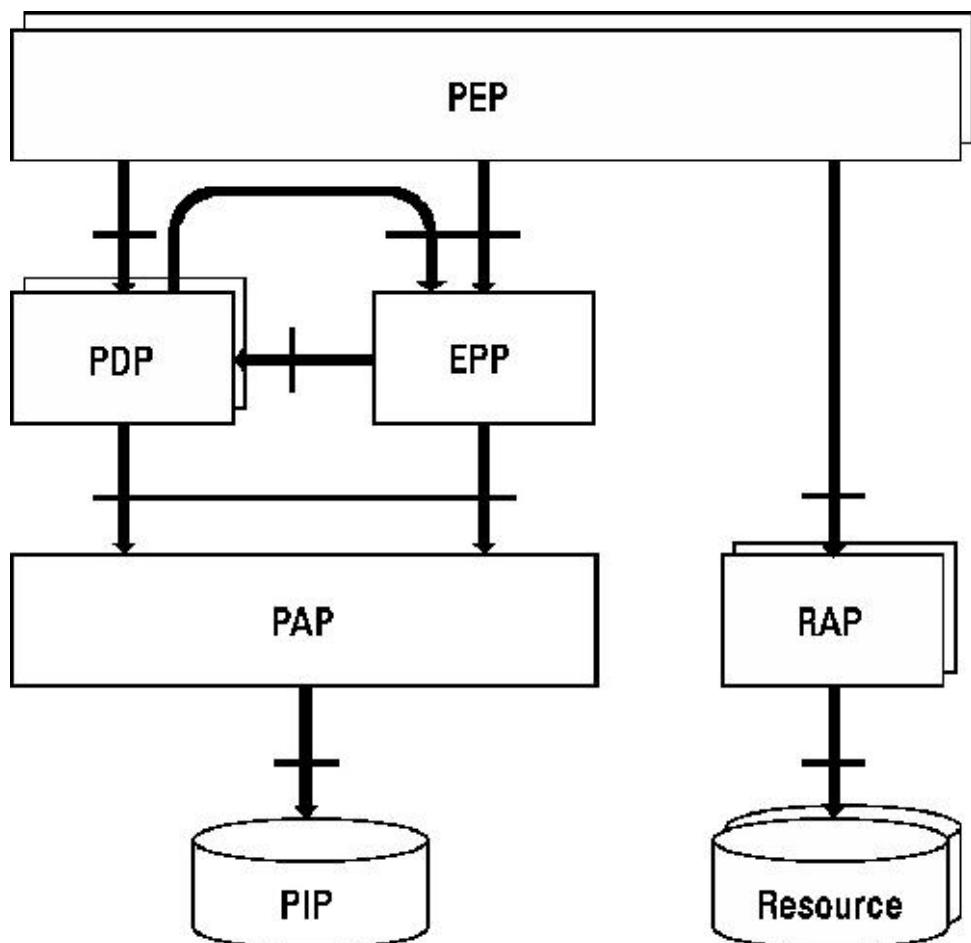


Figure 11.1 NGAC PM functional architecture [4].

- One policy administration point (PAP);
- One policy information point (PIP);
- One or more resource access points (RAPs).

Collectively these components provide the interfaces and protocols needed for:

- Performing administrative operations needed for building the ABAC model instance for supporting applications (i.e., creating, deleting, and modifying basic elements, containers, and relations);
- Enabling users to submit access requests, generate access control decisions, and enforce them for resources managed by the various applications.

Processing of administrative operations is the focus of [Section 11.3](#) while the steps involved in processing a resource access request are discussed in [Section 11.4](#).

11.3 NGAC PM: ABAC Model Definition Capabilities

This section describes the role of various NGAC PM architecture's functional components in carrying out administrative operations needed for creation of a NGAC ABAC model instance. Examples of these administrative operations include creation and modification of various policy elements of the NGAC ABAC model such as basic elements, containers, and relations (refer to [Chapter 5](#) for definitions and semantics of these NGAC ABAC model policy elements). The NGAC PM PEP is the component that provides the API for all types of client applications to interact with NGAC PM, whether they are applications submitting administrative operations (e.g., policy manager application) or submitting a resource access request. More than one PEP can exist to service applications. All access requests to protected resources must pass through PEP and hence it should be nonbypassable.

The PEP passes an administrative access request to PDP that determines whether the access request complies with policy and renders, in most instances, either a grant or deny decision. This is the common function carried out by PDP for both administrative access request and resource access request. In addition, in the case of a situation where an administrative access request is granted, the PDP also performs the necessary access on policy elements stored in PIP through PAP and supplies the result to the requesting PEP along with the decision. A single PAP manages all access to the contents of the PIP. A PAP provides read, modify, and write access to the content of the PIP (i.e., the policy store or policy configuration) based on the semantics of the administrative routines defined

in the NGAC PM. It also ensures that access to the PIP is serialized. The PIP contains the data structures that define the policy elements and the relationships between the policy elements (NGAC ABAC model) that collectively constitute the access control policy enforced by NGAC PM. All changes to the policy are triggered by administrative access request submitted to PEP, which then is passed through PDP to PAP. The latter carries out the request against the NGAC ABAC model stored in PIP. This model can be encoded in a set theoretic or graph theoretic notation and can be defined using a conventional language such as Java or a scripting language such as Python.

11.4 NGAC PM: Access Decision Process

As already stated, the same NGAC PM architecture's functional components (involved in processing administrative access requests) are involved in processing and carrying out a resource access request. A typical resource access request originating from a client application includes the identity of the requesting process (generated by the application's executable), the requested operation, the arguments of the operation (including the targeted resource(s)), and optional data, is intercepted by PEP and passed on to PDP. If the PDP grants the access request, it also sends the uniform resource identifier (URI) for the physical resource in question. This enables the PEP to carry out the requested access by using the URI to identify the appropriate RAP, issue the appropriate commands against it, and return the results to the application.

The PDP obtains the information needed for processing the resource access request from the PIP via the PAP. Along with supplying the necessary details to the requesting PEP for locating and accessing the resource as stated earlier, the PDP also generates an event describing the access and conveys it to EPP. A single EPP is responsible for comparing events against event patterns that have been defined in obligations residing at PIP. For each event pattern that is matched with an event, the EPP uses a PDP to make an access request decision on the associated event response (i.e., the sequence of administrative actions defined for each obligation) and to carry out the response if the definer of the obligation holds sufficient authorization. The sequence of administrative actions in an event response is processed as a transaction by PDP (via the PAP) (i.e., the response is carried out completely or no part of the response is carried out). Since a

PDP may be used by several PEPs, the EPP may use a dedicated PDP to ensure that the event response is carried out with a high degree of certainty.

Recall that PEP uses a RAP to reach the protected resources after it obtains the grant authorization from PDP along with the URI of the physical resource. Multiple RAPs can exist, but each protected resource is accessible only through a single RAP. The PEP issues a command containing its identifier, the location of the physical resource (found in URI), the operation, and any required data to the RAP. The RAP returns the data and status information to the PEP. The RAP does not allow access to resources to any component other than a PEP.

The NGAC PM uses a NGAC ABAC model whose internal representation is set-theoretic. The same is true for Medidata PM [3], whose underlying ABAC model is also specified using a set-theoretic notation. It has been found that all implementations using set-theoretic notations for computation of access decision (i.e., whether a given user has a specific privilege on a given object) are not computationally efficient with respect to the total dimension of an enumerated ABAC model (i.e., the total number of basic elements, containers, and relations). Therefore, a graphic-theoretic representation of enumerated ABAC models as well as an efficient algorithm for access decision computation was developed in the Python language [7]. This algorithm has since been implemented in Java in the NGAC PM reference implementation. This algorithm has also been used not only for accession decision computation but also for other functions, such as policy review, which serves the purpose visualizing all objects (and access modes) that are accessible to a given user or a set of users.

The mapping from the earlier set-theoretic representation in NGAC ABAC model to graph-theoretic representation is made as follows [7].

- Each of the members of the basic element or container is mapped to a graph node. Five primary types of nodes are considered: user (u), object (o), user attribute (ua), object attribute (oa), and policy class (pc).
- The relations (assignment relation and association (privilege) relation) are represented as directed edges. Object attribute (oa) nodes can have edges only to other oa or pc nodes. However, no oa node may point to an object (o) node. User (u) nodes are sources with edges only to ua nodes. However, user attribute (ua) nodes have edges to ua , oa , or pc nodes. Policy class nodes are sinks.
- Cycles and self-loops are prohibited. The edges joining user attribute (ua) nodes to object attribute (oa) nodes are labeled with a

set of one or more allowed operations (ops) (e.g., read or write). All other edges that we have referred to earlier (u to ua , o to oa , ua to ua , oa to oa , ua to pc , oa to pc —which are representations of assignment relations in graphic-theoretic notation)—are unlabeled.

- All nodes should have a path (created by the sequence of directed edges) to at least one pc node (without using ua to oa edges—representation of association or privilege relations). This ensures that all users, objects, user attributes, and object attributes are covered by at least one policy class.

The various connectivity restrictions discussed above results in a type of graph called directed acyclic graph (DAG) which in fact can be divided into two DAGs: a user DAG (with u and ua nodes) and an object DAG (with o and oa nodes), as shown in [Figure 11.2](#).

The set of u nodes act as sources for the user DAG and the set of o nodes act as sources for the object DAG. The directed set of ua to oa edges bridge the two DAGs and hence are called bridge edges. Bridge edges are the only ones labeled with operations (ops). The tail node of a bridge edge is a ua node and the head node of a bridge edge is oa node. The bridge edges represent the association (privilege) relations of the NGAC ABAC model and thus are the focal points in arriving at an access decision (user privilege determination).

Mapping the NGAC ABAC model's structural relationship into the graphical representation, one can construct an arbitrary graph called NGAC access control graph. One such graph used in [7] with symbolic identifiers (such as $u1$, $o2$, $ua4$, $pc1$, etc.) is shown in [Figure 11.3](#).

The process of arriving at an access control decision for two sample access requests is illustrated through a path-finding process using the above graph. Readers desiring the algorithmic steps for deriving the privilege that is sought can consult [7].

Access control request 1: Is the user $u1$ allowed read access (operation) on object $o2$?

1. The first step is to identify all association (privilege) relations that contains the operation read, or r , which in the graphical representation represents a set of bridge edges. The only two bridge edges found in the above NGAC access control graph of [Figure 11.3](#), which contains r operation are $ua1-oa1$ and $ua2-oa4$. The only way these bridge edges will have relevance for access control decisions is that for at least one bridge edge, the tail node of the

bridge is reachable from the target user u1 (or in other words the tail ua node contains u1) and the head node of the bridge is reachable from the target object o2 (the head oa node contains o2). We find that this is true for both our bridge edges. The tail node in ua1-oa1 bridge edge is ua1 and is reachable from u1 (through the single directed edge u1-ua1), while the corresponding head node is oa1 and is reachable from o2 (through o2-oa2 and oa2-oa1 directed edges). The tail and head node of the other bridge edge ua2-oa4 also satisfies this condition. These two bridge edges (ua1-oa1 and ua2-oa4) now constitute what is termed as the set of active bridge edges.

2. The next step after determining the set of active bridge edges is to determine the policy class coverage for the target object o2. What this means is that the required policy classes for this object o2 should be a proper subset of the covered policy classes by the head nodes of the (privilege granting) active bridge edges. The former can be computed by determining the set of pc nodes reachable from o2. In our access control graph, pc1 and pc2 are the set of pc nodes reachable from o2 (through the path (a sequence of directed edges) o2-oa5-oa4-pc1 and o2-oa2-oa1-pc2), respectively. Thus the set {pc1,pc2} forms the set of required policy classes for the object o2. To determine the set of covered policy classes, we have to look at the head nodes of the active bridges. These are the nodes: oa1 and oa4. The pc node pc1 is reachable from oa4 while the pc node pc2 is reachable from oa1. Hence the set {pc1, pc2} forms the set of required policy classes. Since the required policy class set is a proper subset of a covered policy class set, the second condition for access to object o2 is satisfied.

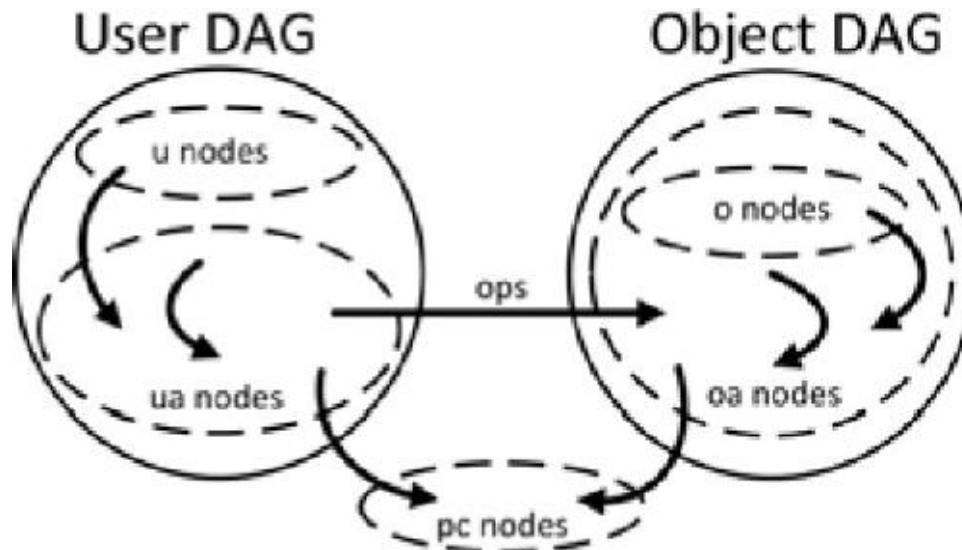


Figure 11.2 Allowed edge relationships in the graphical representation of NGAC ABAC Model [7].

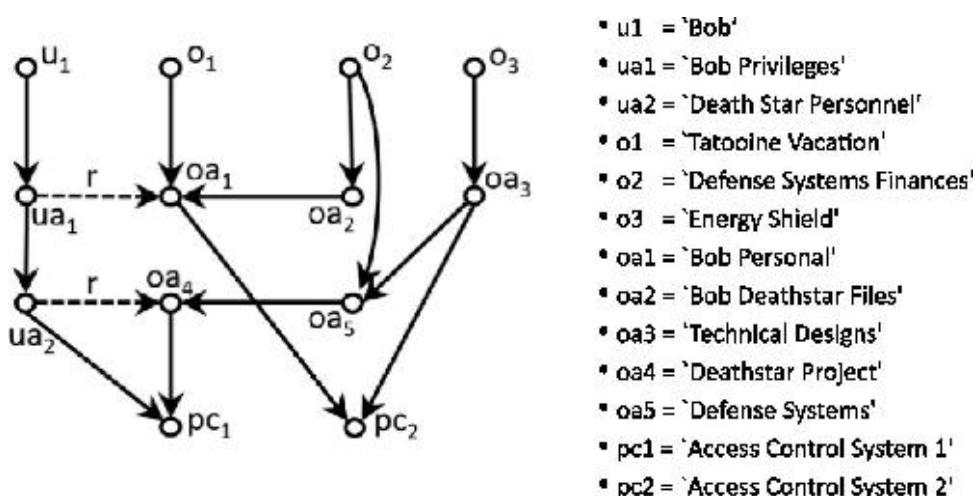


Figure 11.3 NGAC access control graph [7].

Access control request 2: Is the user u1 allowed read access (operation) on object o3?

1. As in the previous scenario, we find that the two bridge edges found in the above NGAC access control graph that contains the r operation are ua1-oa1 and ua2-oa4. However, in this case, the head node oa1 is not reachable from o3 and only the head node oa4 is reachable from o3. Hence the only active bridge edge is ua2-oa4.
2. The only policy class covered by the head node of the active bridge is pc1 since pc1 is directly reachable from oa4. However, the required policy classes for o3 are both pc1 and pc2 (through the

paths o3-*oa*3-*oa*5-*oa*4-*pc*1 and o3-*oa*3-*pc*2, respectively). Since the second set is not a proper subset of the former, u1 cannot read o3.

Policy review question: What are the sets of accessible objects for a user u1 (Bob)?

Using the assignment of names to symbols in [Figure 11.3](#), it is possible to answer the set of accessible objects for Bob. By repeated application of the process enumerated above for answering the access control requests to all objects of interest, it is possible to obtain a visual representation of all accessible objects for Bob as shown in [Figure 11.4](#) below:

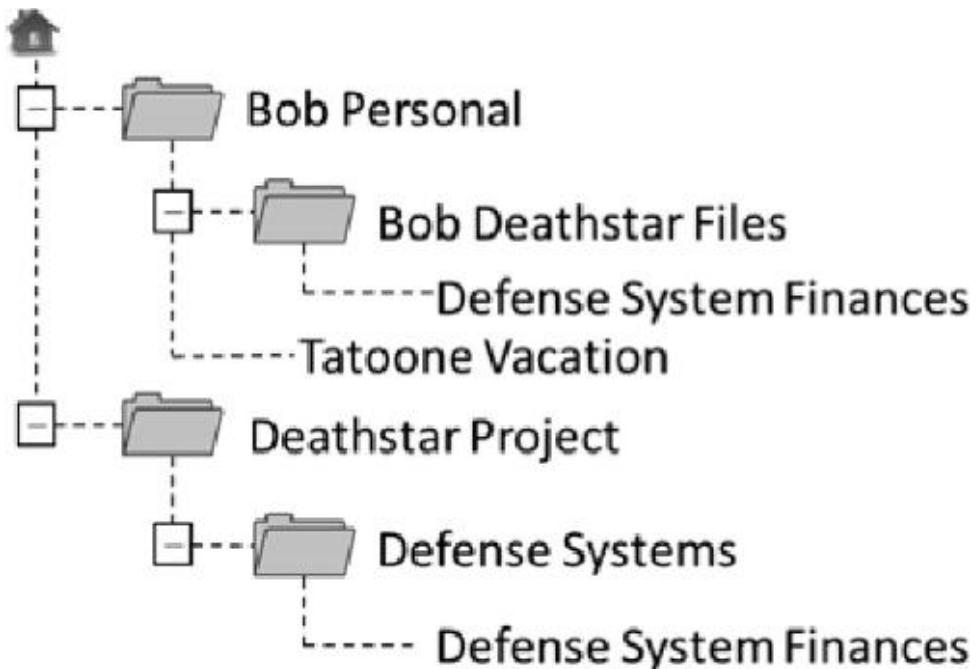


Figure 11.4 Policy review: set of accessible objects for Bob [7].

11.5 NGAC PM: Design and Application Integration

Each of the components in the NGAC PM architecture have well-defined interfaces. Therefore, NGAC PM can be implemented either as a monolithic application running on a single server and sharing the same execution space or it can be implemented as a distributed application with its components communicating with each other through a network protocol.

Applications that want to deploy NGAC PM to gain access to protected resources must make use of the application programming interface (API) provided by the NGAC PM's PEP component. The name for collection of all components of NGAC PM shown in the functional architecture in [Figure 11.1](#), apart from RAP and protected resources, is PM environment. As

shown in [Figure 11.5](#), the PEP API is the only means available for an application to interact with the PM environment and obtain access to protected resources. Those applications that have been developed from the ground up to leverage NGAC PM's PEP API are called PM-aware applications. Alternatively, existing applications developed without considering integration with NGAC PM as part of the original design can be adapted later on for the NGAC PM by intercepting access requests at key points in the code and converting them to calls on the PEP interface for eventual mediation by the PDP.

The following are some of the resources that have been demonstrated for access protection by integrating their corresponding client applications with NGAC PM:

- Files under a file management system;
- Records (with access protection at individual field level);
- Workflow data;
- Open office resources;
- Internal email;
- Other auxiliary resources such as clipboard objects.

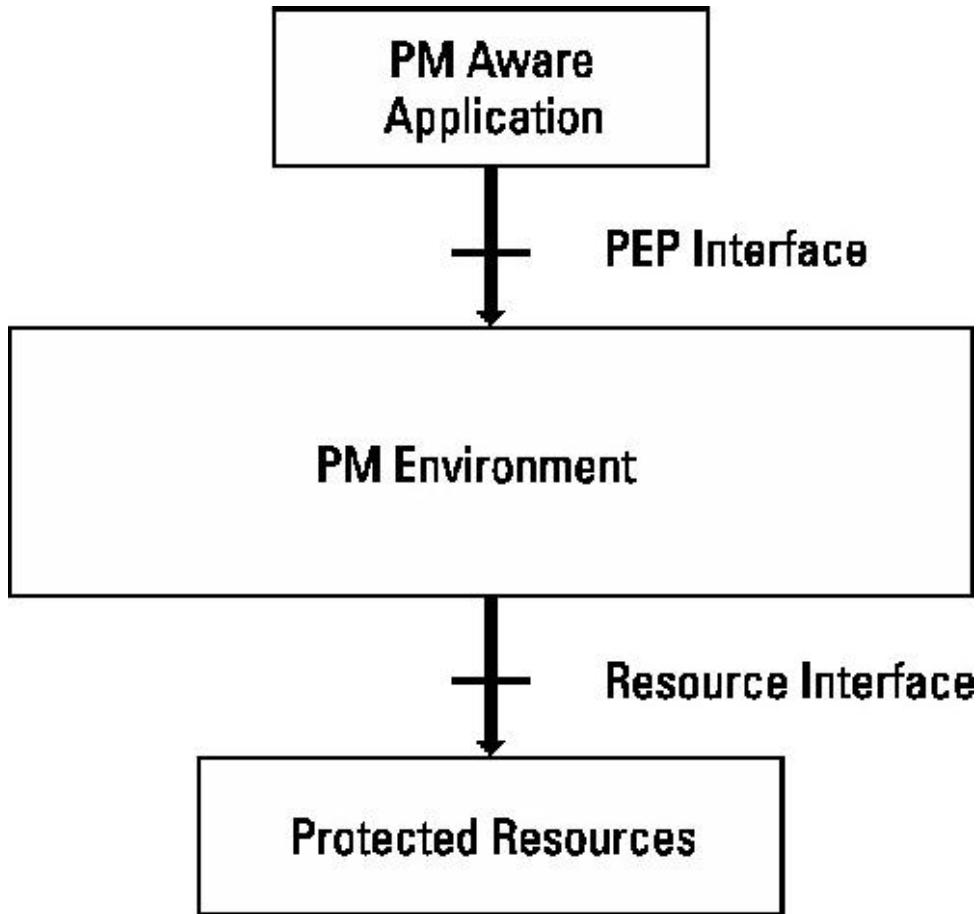


Figure 11.5 Application’s perspective of NGAC PM environment [4].

11.6 Summary and Analysis

Current open-source ABAC products use the model described in [Chapter 5](#) (i.e., NGAC). The feature that distinguishes this type of model (i.e., enumerated ABAC model) from XACML-based ABAC models was highlighted. This was followed in [Section 11.2](#) by an overview of the functional components of NGAC PM architecture—an open-source product that has the NGAC ABAC model (an enumerated ABAC model) as its foundational model. The role played by these functional components in policy formulation (ABAC model definition) and in processing resource access requests were discussed in [Sections 11.3](#) and [11.4](#), respectively. Since an enumerated ABAC model such as NGAC ABAC model can be described using set-theoretic as well as graphical approaches, the use of the latter approach to illustrate the access decision process and policy review process was also shown in [Section 11.4](#) since the latest implementation of NGAC PM deploys the graph-theoretic approach to develop the NGAC

access control graph. The NGAC PM design and integration capabilities were highlighted in [Section 11.5](#).

The reader can recognize that the functional architecture of the open-source ABAC product (i.e., NGAC PM) in this chapter contains many components that are common with commercial ABAC products (which implement the XACML functional architecture) discussed in [Chapter 10](#). Specifically, there are four common components: PEP, PDP, PAP, and PIP. Therefore, it is critical to examine whether the functional roles of these components in the two classes of ABAC products (i.e., commercial ABAC products and open-source ABAC products) and by coincidence the two classes of ABAC models (i.e., logic-based and enumerated) are also the same or different. Taking up the case of the PEP component, we find that the primary use of this component is to integrate applications seeking controlled access to resources. In this context, it is useful to identify that the open-source ABAC product we have discussed uses the same set of interfaces for policy management (administrative access requests) as well as for resource access request. Thus, for this class of application, both a policy management application as well as a business process application are integrated with a PEP API, while for a commercial ABAC product, it is usually the business process application. Still, conceptually the functionality of a PEP is identical in both classes of products (i.e., the interception of access request). However, we find that for the other three components, the scope of functions is broader in the NGAC PM implementation compared to those found for those components in the XACML functional architecture. The differences in the functions of these components are brought out in [Table 11.1](#) below.

Now that the differences in functionality between components in XACML functional architecture and NGAC PM architecture have been highlighted, it will be of value to the reader to highlight the differences in the sequence of operations involved in the access decision process between NGAC PM and XACML functional architecture. Despite dissimilar nature of operations, the overall process itself can be identified to have three distinct phases as was outlined in [\[8\]](#). These processes are

- Loading policies (access control information) from disk to memory;
- Finding applicable policies (pruning of access control database);
- Use of pruned information to perform access decision.

Table 11.1
Functions of Components in XACML and NGAC PM Architectures

| XACML Functional Architecture (Found in Commercial ABAC Element products) | NGAC PM Implementation (Representative Open-Source ABAC Product) | |
|--|---|---|
| PAP | <p>1. Perform administrative operations to create, modify, and delete XACML rules, policies, and policy sets.</p> | <p>1. Perform administrative operations to create, modify, and delete authorization information in the form of elements, containers, and relations in PIP. 2. Perform search and retrieval of authorization information from PIP for rendering an access decision in response to a request from PDP.</p> |
| PDP | <p>1. Receive resource access request from PEP, retrieve necessary policies from policy store (not specified in XACML), retrieve additional attributes needed from various attribute sources through PIP, and convey the results of access decision to PEP.</p> | <p>1. Receive resource access request from PEP, retrieve authorization information pertaining to the request from the PIP via the PAP, and convey the results of the access decision to PEP 2. If the resource access request is granted, also obtain the information necessary to locate the resource and communicate the locator for the resource to the PEP along with the decision result. 3. Receive administrative access request from PEP, evaluate the request, and render the decision. In addition, if the decision is “grant,” carry out the administrative access request on the PIP (policy store) through PAP.</p> |
| PIP | <p>1. Receives attribute requests from PDP, pulls in attributes from multiple attribute sources, and returns them to PDP.</p> | <p>1. Used as authorization information store, and holds information such as elements, containers, and relations.</p> |

Table 11.2
Access Decision Process in Logic-Based and Enumerated ABAC Models

| Accession Decision Process | XACML ABAC Model (Logic-Based) | NGAC PM Model (Enumerated) |
|--|---|---|
| Loading policies (access control information) from disk to memory | <p>1. Loading time is dependent on the number of policies and caching algorithms used.</p> <p>2. Irrespective of the number of policies, the policy structure encoding in XML needs to be converted to an in-memory representation (e.g., DOM tree) during loading of the model data from disk to memory.</p> | <p>1. The NGAC ABAC model can be represented as a DAG and hence minimal processing is required to convert it into an in-memory structure while loading the model data from disk to memory.</p> <p>2. The economy and simplicity of graph-theoretic representation enables the entire access control information (ABAC model instance) to be memory-resident, making processing more efficient because it avoids using time-consuming I/Os for fetching information either from the cache or secondary storage and transferring it to the main memory.</p> |
| Finding applicable policies (pruning of the access control database) | <p>1. Applicable policies among the memory-loaded policies are identified by matching the attributes (user, object, environment, and action) in the access request to corresponding entities in the policy target. The time for matching is dictated by the complexity of logical expressions in the target.</p> <p>2. Use of proxy policies (containing just the target portion without logical policy expressions) for speeding up the matching may require the loading of full policy data for the</p> | <p>1. In the NGAC ABAC model, identifying the set of applicable access control information translates to finding the subgraph consisting of nodes (and their connecting edges) reachable from the nodes representing the user and objects in the access request. The time for this operation is dependent on the connectedness of the access control graph.</p> |

| | |
|---|---|
| | identified applicable policies. Loading time is dependent on the number of applicable policies and the type of caching algorithms used. |
| Use of pruned information (i.e., evaluating applicable policies) to perform access decision | <p>1. The access decision is computed by evaluating logical expressions in the applicable policies, using attribute values provided by context handler or PIP sources, and applying combining algorithms if multiple rules or policies are used in the access decision.</p> <p>2. Performance determinants are complexity of logical expressions, number of rules requiring dynamic attributes, relative stability of contents of policy store, and the types of caching algorithms used.</p> <p>1. The existence of valid privilege is determined by (a) reachability from user and object nodes in the access request to tail and head nodes, respectively, of the edges that contain privilege assignments, and (b) the set of policy classes covering the object attributes in the privilege assignments (i.e., given by the set of policy class nodes reachable from the head nodes of the privilege assignment edges) being the superset of policy classes covering the objects in the access request (given by the set of policy class nodes reachable from the nodes representing the accessed objects).</p> <p>2. The processing algorithm for determining the existence of valid privilege has the complexity equal to the cardinality of subgraph (number of nodes + edges) consisting of nodes (and their connecting edges) reachable from the nodes representing the user and objects in the access request.</p> |

It can be reasonably expected that if the nature of operations under the common phases listed above can be described for XACML and NGAC PM implementations, the reader can make objective conclusions about the relative efficiency of the whole access decision process between these implementations and thus between logic-based and enumerated ABAC model representations as well. The operations under the three common

phases for XACML and NGAC PM model representations are given in [Table 11.2](#).

Although the differences in operations in the three common phases in the accession decision process do not provide the reader with a complete idea of performance scalability of the two model representations in terms of parameters such as number of users, objects, and governing policies, they do highlight certain distinguishing characteristics. One characteristic is the fact that in the graph-theoretic representation of the enumerated ABAC model, the entire access control data (represented by the access control graph) can be loaded and can permanently reside in the memory providing the advantage of in-memory processing. However, efficiency in policy evaluation can be achieved in logic-based representations using measures such as caching of all policies and selective caching of attributes whose values are relatively static, instead of retrieving policies and attributes from storage locations during every policy evaluation.

References

- [1] Biswas, P., R. Sandhu, and R. Krishnan, “Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy,” In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, ABAC ‘16, pp. 1–12, New York: ACM, 2016.
- [2] NIST Policy Machine Versions 1.5 and 1.6—Harmonia [website], <https://github.com/PM-Master>.
- [3] Medidata Solutions Worldwide, Medidata Policy Machine code on github, version 1.1.0. www.github.com/mdsol/the_policy_machine, 2016.
- [4] Ferraiolo, D., S. Gavrila, and W. Jansen, *Policy Machine: Features, Architecture, and Specifications*, Technical Report NISTIR 7987 Revision 1, National Institute of Standards and Technology, October 2015.
- [5] ANSI. American National Standard for Information Technology—Next Generation Access Control—Functional Architecture (NGAC-FA), 2013.
- [6] ANSI. American National Standard for Information Technology—Next Generation Access Control—Generic Operations and Data Structure (NGAC-GOADS), 2016.
- [7] Mell, P., J. M. Shook, and S. Gavrila, “Restricting Insider Access Through Efficient Implementation of Multi-Policy Access Control Systems,” in *Proceedings of the 2016 International Workshop on Managing Insider Security Threats*, pp. 13–22, New York: ACM, 2016.
- [8] Ferraiolo, D., R. Chandramouli, V. Hu, and R. Kuhn. “A Comparison of Attribute Based Access Control (ABAC) Standards for Data Service Applications.” *NIST*

Special Publication 800-178, <http://dx.doi.org/10.6028/NIST.SP.800-178>.

About the Authors

Dr. Vincent C. Hu is a computer scientist in the Computer Security Division of the National Institute of Standards and Technology (NIST). Since 1985, Dr. Hu has experience in the design, development and research of information systems, both in commercial and government sector, and in areas such as air traffic control, e-commerce, and communication. Dr. Hu joined NIST in 1998, and his research focus including access control policies, policy verification, distributed systems, PKI, intrusion detection, formal models and quantum computing. He has written more than 50 conference and journal publications in the areas of computer security, and is the lead author of the “Attribute-Based Access Control Definition and Consideration,” which is the first ABAC definition document published by the U.S. government. He also designed and developed the Access Control Policy Tool (ACPT) for access control policy verification and testing. Dr. Hu holds an M.S. in computer science from Old Dominion University in Norfolk, Virginia, and a Ph.D. in computer science from University of Idaho in Moscow, Idaho.

David F. Ferraiolo is the manager of the Secure Systems and Applications group of the Computer Security Division at the National Institute of Standards and Technology, with a focus on cloud computing security, virtualization, access control, mobile device security, and identity and access management. He has over 30 years of experience in computer and communications security, serving both the U.S. government and private industries. During his 25 years of employment at NIST, he has conducted extensive research in various areas of access control and authorization management, including formal model development, reference and prototype implementation, product demonstration development and evaluation, and is given credit as the originator of numerous commercially available security mechanisms. He is a coauthor of a book on role-based access control, is the author or coauthor of more than 40 papers, and the principle inventor on two U.S. provisional patents. He received a U.S. Department of Commerce gold

medal, an IEEE Innovation in Societal Infrastructure award, and an Excellence in Technology Transfer award from the Federal Laboratory Consortium for research, and has served on the editorial boards of numerous standardization efforts, to include the Common Criteria (ISO 15408), Role-Based Access Control (ANSI/INCITS 359), Next Generation Access Control (ANSI/INCITS 499 and 526). His degrees are in computer science and mathematics from SUNY Albany.

Dr. Ramaswamy Chandramouli has 32 years of professional experience in areas spanning information systems design, development, and implementation, and computer security research. His experience in the design, development and implementation of information systems spans both the commercial and government sectors in diverse areas such as international banking, healthcare, energy, and transportation for the first 12 years. The remaining 20 years to the current time has been devoted to research, prototyping, testing, and publications in the area of computer security, covering areas such as role-based and attribute-based access control models, model-based test development, smart card interface specifications and conformance test methods, development of security profiles, DNS security, identity management and authentication protocols, and security of virtualization and cloud computing infrastructures.

Rick Kuhn is a computer scientist in the Computer Security Division of the National Institute of Standards and Technology. He has authored two books and more than 150 papers on information security, empirical studies of software failure, and combinatorial methods in software testing, and is a senior member of the Institute of Electrical and Electronics Engineers (IEEE). He received the 2018 IEEE Innovation in Societal Infrastructure award, shared with D. Ferraiolo and R. Sandhu, “for advancing the foundations and practice of information security through creation, development, and technology transfer of role-based access control (RBAC),” in addition to a gold medal for scientific/technical achievement for RBAC research from the U.S. Department of Commerce and the Excellence in Technology Transfer award from the Federal Laboratory Consortium. His work in combinatorial test methods also received a U.S. Department of Commerce Silver award for scientific/technical achievement and an Excellence in Technology Transfer award. Previously he served as program manager for the Committee on Applications and Technology of

the Information Infrastructure Task Force, and also as manager of the Software Quality Group at NIST. Before joining NIST in 1984, he worked as a software developer with NCR Corporation and the Johns Hopkins University Applied Physics Laboratory. He received an M.S. in computer science from the University of Maryland College Park and an M.B.A. from the College of William & Mary.

Index

- Access control, 1–3
- Attribute-based access control
 - academic contributions, 3–5
 - Bell and LaPadula model, 8
 - introduction to, 2, 13–15
 - military concerns, 5–7
 - model and policies, 21–23
 - role-based, 11–13
 - TCSEC, 9–11
 - terminology, 19–21
- Access control auditing, 168
- Access control function (AF), 134
- Access control list (ACL), 4, 24, 62, 215–17
- Access control mechanism (ACM), 202–4
 - distribution, 207–8
 - enterprise ABAC, 220–21
- Access control policies
 - XACML, 83–89
 - Web services, 176–81
- Active Control Policy Language (ACPL), 242
- Access control policy tool (ACPT), 128–29
- Access control safety, 115–17
- Access request processing, XACML, 85–89
- Access rights (AR), 20, 104–5
- Acquisition/ development phase, 221–29
- Administrative access, NGAC, 107, 109–10
- Administrative access rights (AAR), 20
- Administrative policies
 - NGAC, 105–6
 - XACML, 83–89
- Agreements, enterprise, 231–32
- Assignments, NGAC, 99–102
- Associations, NGAC, 99–102
- Attribute-at-rest security, 144
- Attribute-based access control (ABAC), 2

applications, 40–42
architecture, 36–38
attribute-centric, 57–58
complexities, 50–52
dynamic roles, 52–55
emergence of, 13–15
enumerated, 38, 39–40
hierarchical group, 42–44
hybrid designs, 46–49
introduction to, 2, 33–36
logical-formula, 38–39
XACML deployment, 71–76

Attribute-centric structure, 57–58

Attribute comparison expression (ATE), 73–76

Attribute consideration

- ABAC systems, 133–34
- elements, 134–36
- evaluation, 152–53
- framework, 146–51
- preparation, 136–41
- readiness, 145–46
- security, 143–45
- veracity, 141–43

Attribute elements, NGAC, 98–99

Attribute evaluation scheme (AES), 135–36, 152–53

Attribute-in-transit security, 144–45

Attribute management, enterprise, 205–6

Attribute practice statement (APS), 153, 219

Attribute provider (AP), 134

Attribute support, XACML, 80–81

Attribute retrieval, 235–36

Attribute trustworthiness, 141–42

Attribute value accuracy, 141, 42–43

Authentication

- distributed systems, 159
- enterprise systems, 225–26

Authorization, distributed systems, 159

Authorization request and response, 67–68

Axiomatics Data Access Filter (ADAF), 237–39

Backup, 146

Bell and LaPadula security model, 8

BigData access control, 160–67

Business Process Execution Language (BPEL), 169, 181–86

Cache, 146
Cloud distributed systems (DS), 168
Content attribute, 167–68
Context handler (CH), 68–71
Coverage and confinement checks (CCC), 122–24
CS ABAC policy (CSP), 164
Cyclic inheritance, 116

Data flow model, XACML, 68–71
Data providing Web service (DPWS), 175–81
Data service operations, NGAC, 106–7
Decision function, NGAC, 103–4
Delegation, XACML, 81–89
Denials, NGAC, 102–3
Digital policy (DP), 204–5, 224, 227
Directed acyclic graph (DAG), 252
Discretionary access control (DAC), 4, 21, 22, 23–24, 133
Distributed systems (DS)
 access control challenges, 158–60
 analysis of, 168–69
 BigData, 160–67
 implementation, 167–68
 introduction to, 157–58
Dynamic policy class, 113–14
Dynamic roles, 52–55
Dynamic separation of duty (DSD), 30, 118

Enterprise ABAC
 access control, 207–8
 acquisition/ development, 221–29
 attribute management, 205–6
 implementation/assessment, 229–30
 initiation phase, 210–21
 operations/ maintenance, 230–32
 overview, 202–4, 209–10
 policy, 204–5
Enterprise, defined, 202
Enterprise Dynamic Access Control (EDAC), 48, 52–55
Enumerated ABAC, 38, 39–40, 45–46, 247. *See also* NGAC/ PM
Environment condition preparation, 139–40
Environment condition, 220
eTRON operating system, 56
Event processing point (RAP), NGAC, 107–9
Extensible Access Control Markup Language (XACML), 15, 61–62

ABAC deployment, 71–91
attribute support, 80–81
data flow model, 68–71
delegation, 81–89
drivers, 62–63
functional features, 236–39
implementation, 79–89
policy language model, 64–67
policy types, 83–89
request/ response context, 67–68
terminology map, 63–64
Web services, 172–81, 183–85

Extensible stylesheet language transformation (XSLT), 177
eXtensible Markup Language (XML), 175

Federated attribute definition (FAD), 164
Ferraiolo-Kuhn model, 11–12
Filtering class decision point (FCDP), 175–81
Filtering definition generator (FDG), 177
Finite state machine (FSM), 112–15, 118–22
Formal methods, validation, 129
Fragmented data, 159
General attribute framework (GAF), 146–51
Generic attributes (GA), 148

Harrison, Russo, Ullman (HRU) model, 23
Hierarchical Group and Attribute-Based Access Control (HGABAC), 42–44
Historical policy class, 114–15
Hybrid designs, 46–49

Identity management, 225
Implementation phase
 distributed systems, 167–68
 life-cycle issues, 229–30
 policy verification testing, 125–27
INCITS 359–2004, 13
Initiation phase, enterprise ABAC, 210–21
Interoperability, 224

Jericho System’s EnterSpace 9, 239–41

Label-based access control, 39, 45–46
Life-cycle issues
 acquisition/development, 221–29
 enterprise concepts, 202–8

enterprise considerations, 209–21
implementation /assessment, 229–30
introduction to, 201–2
questions/ maintenance, 230–32

Log, 146

Logical-formula ABAC, 38–39, 40–42. *See also* Yuan and Tong policy model

Logic-based model. *See* XACML ABAC

Mandatory access control (MAC), 9, 10, 21, 25
attribute consideration, 133
Chinese Wall policy, 27–29
multilevel security, 25–27
role-based access control, 29–30, 55

Many-sorted first order logic (MSFOL), 129

Mapping attributes, 228–29

Metadata, 140–41

Military concerns, 5–7

Minimum attribute assignment, 226–27

Model verification, 118–22

MS ABAC policy (MSP), 164

Multilevel security (MLS), 22, 25–27

Multi-policy considerations, 117

Multiterminal binary decision diagrams, 128

Named protection domain (NPD), 11

National Identity Exchange Federation (NIEF), 148

Natural language policies (NLP), 147, 149
enterprise system, 204–5, 214, 221–22, 227

Next Generation Access Control (NGAC), 15, 39, 45, 46
access decisions, 103–4
access right delegation, 104–5
administrative access, 107, 109–10
administrative operations, 105–6
assignments/ associations, 99–102
data service operations, 106–7
introduction to, 97–98
obligations, 103
policy/attribute elements, 98–99
prohibitions (denials), 102–3
resource access, 107–9

NextLabs, 241–45

NGAC PM
access decisions, 250–54
administrative operations, 249–50

functional architecture, 248–49
integration interfaces, 255–56
summary of, 25659

Node-to-node communication, 159

Nonperson entity (NPE), 225

Object attribute assignment (OAA), 44

Object attribute preparation, 138–39

Object containers, NGAC, 98–99

Object group (OG), 43–44

Object GROUP attribute assignment (OGAA), 44

Obligations, NGAC, 103

OMB 6–16/OMB 7–16, 149–51

Operations in access control, 20

Operations/ maintenance phase, 230–32

Physical-Logical Security Interoperability Alliance (PSIA), 56–57

Policy administration point (PAP)
enterprise ABAC, 207–8
NGAC, 107–9
Web services, 172–74
XACML, 68–70, 79, 89, 134

Policy classes, 112–15

Policy decision externalization, 228

Policy decision point (PDP)
enterprise ABAC, 207–8
NGAC, 107–10, 248–51
Web services, 172–74, 192–94
XACML, 68–70, 76–77, 78, 79, 89–90, 134

Policy elements, NGAC, 98–99

Policy enforcement, 22–23

Policy enforcement point (PEP)
enterprise ABAC, 207–8
NGAC, 107–9, 248–51
Web services, 172–74, 192–94
XACML, 68–70, 76, 79, 134

Policy evaluation, XACML, 77–78

Policy information point (PIP)
enterprise ABAC, 207–8
NGAC, 107–9, 248–51
XACML, 68–70, 76–77, 79, 90, 134

Policy language model, XACML, 64–67

Policy model, 6, 7

Policy verifications, 111–12. *See also* Verification approaches; Verification tools

Preparation, attribute evaluation scheme, 136–41
Privacy, enterprise system, 223–24
Privilege blocking, 116
Privilege conflict, 116
Privilege leakage, 116
Prohibitions, NGAC, 102–3
Property confinement checking, 124–25
PSIA Physical-Logical Access Interoperability (PLAI), 56–57
Published filtering description (PFD), 175–81

RAND report, 5–6, 8
RBAC96, 12–13
Readiness, attribute evaluation scheme, 136, 145–46
Reference monitor, 6–7
Refresh, 145
Request and response, XACML
 authorization, 67–68
 formulation, 76–77
Resource access, NGAC, 107–9
Resource access point (RAP), NGAC, 107–9
Resource access rights (RAR), 20
RESTful services, 169
REST profile, XACML, 71
Review of capabilities, 40
Role-based access control (RBAC), 11–13, 22
 complexities, 49–50
 dynamic roles, 52–55
 hybrid models, 48–49
 role-centric structure, 55–57
 standard model, 29–30
Role-centric structure, 55–57
Rule coverage checking, 122–24

SARE elements, 195–97
Safety, access control, 115–17
Security agreement (SA), 163
Security Assertion Markup Language (SAML), 143
Security, attribute evaluation scheme, 136, 143–45
Security kernel, 6, 7
Separation of duty (SoD), 10, 29–30, 115–17, 118
Service-oriented architecture (SOA), 40
SOAP (Simple Object Access Protocol), 169, 175
SQL Filter Service, 237–39
SQL Proxy Service, 237–39

Standalone workflow processes, 188
challenges, 189–91
integrated approach, 191–95, 197–98
loosely coupled approach, 195–98

Standardization, attributes, 227

Static policy class, 112–13, 119

Static separation of duty (SSD), 30, 171

Subject attribute preparation, 137–38

Synchronization, 146

System development life cycle (SDLC), 209–11, 221. *See also* Life-cycle issues

Temporal RBAC, 49

TRON operating system, 56

Trust chain, enterprise ABAC, 215–17

Trusted Computer Security Evaluation Criteria (TCSEC), 9–11

Trust CS list (TCSL), 163

Trust establishment, DS, 167

Universal Description, Discovery, and Integration (UDDI), 169

User attribute assignment (UAA), 43–44

User authentication, 19

User authorization, 19

User container, 98

User group (UG), 43–44

User group attribute assignment (UGAA), 44

Verification approaches, 117–26
model verification, 118–22

Verification tools, 128–30

Veracity, attribute evaluation scheme, 136, 141–43

Web services

ABAC challenges, 186–87

ABAC suitability, 170–71

background, 169–70

environment requirements, 187–88

environments with workflows, 181–86

environments without workflows, 171–81

Web Services Business Process Execution Language (WS-BPEL), 181–86

Web Services Description Language (WSDL), 169, 175

Workflow, defined, 181

Workflows, environments with, 181–88

WS-I Basic Profile, 169

XML Schema Definition Language (XSD), 169

Yuan and Tong policy model, 40–42

Table of Contents

| | |
|---|----|
| Dedication | 2 |
| Title Page | 3 |
| Copyright Page | 4 |
| Contents | 5 |
| Preface | 12 |
| Acknowledgments | 15 |
| Intended Audience | 16 |
| 1. Introduction | 17 |
| 1.1 Overview | 17 |
| 1.2 Evolution and Brief History of Access Control | 19 |
| 1.2.1 Academic Contributions | 19 |
| 1.2.2 Military Concerns | 22 |
| 1.2.3 Bell and LaPadula Security Model | 25 |
| 1.2.4 Trusted Computer Security Evaluation Criteria | 26 |
| 1.2.5 Discontent | 27 |
| 1.2.6 Role-based Access Control | 29 |
| 1.2.7 Emergence of ABAC | 32 |
| References | 35 |
| 2. Access Control Models and Approaches | 38 |
| 2.1 Introduction | 38 |
| 2.2 Terminology | 38 |
| 2.3 Access Control Models and Policies | 40 |
| 2.4 Policy Enforcement | 42 |
| 2.5 Discretionary Access Control | 42 |
| 2.6 Mandatory Access Control Models | 44 |
| 2.6.1 Multilevel Security | 45 |
| 2.6.2 Chinese Wall Policy and Model | 47 |
| 2.6.3 Role-Based Access Control | 49 |
| References | 51 |
| 3. ABAC Models and Approaches | 53 |

| | |
|---|------------|
| 3.1 Introduction | 53 |
| 3.2 ABAC Architectures and Functional Components | 56 |
| 3.3 Logical-Formula and Enumerated ABAC Policy Models | 59 |
| 3.4 ABAC Model—Applications Primatives | 61 |
| 3.5 Hierarchical Group and Attribute-Based Access Control | 63 |
| 3.6 Label-Based ABAC Model with Enumerated Authorization Policy | 66 |
| 3.7 Hybrid Designs Combining Attributes with Roles | 67 |
| 3.8 ABAC and RBAC Hybrid Models | 70 |
| 3.9 Complexities of RBAC Role Structures | 71 |
| 3.10 Complexities of ABAC Rule Sets | 73 |
| 3.11 Dynamic Roles | 75 |
| 3.12 Role Centric Structure | 79 |
| 3.13 Attribute-Centric Structure | 81 |
| 3.14 Conclusion | 82 |
| References | 82 |
| 4. ABAC Deployment Using XACML | 85 |
| 4.1 Introduction | 85 |
| 4.2 Business and Technical Drivers for XACML | 85 |
| 4.3 XACML Standard—Components and Their Interactions | 87 |
| 4.3.1 XACML Policy Language Model | 88 |
| 4.3.2 XACML Context (Request and Response) | 92 |
| 4.3.3 XACML Framework (Data Flow Model) | 93 |
| 4.4 ABAC Deployment Using XACML | 96 |
| 4.4.1 Access Policy Formulation and Encoding | 97 |
| 4.4.2 Request/ Response Formulation | 102 |
| 4.4.3 Policy Evaluation and Access Decision | 103 |
| 4.5 Implementation of XACML Framework | 105 |
| 4.5.1 Attribute Support and Management | 106 |
| 4.5.2 Delegation | 107 |
| 4.6 Review and Analysis | 117 |
| References | 119 |
| Appendix 4.A | 120 |
| 5. Next Generation Access Control | 126 |

| | |
|---|------------|
| 5.1 Introduction | 126 |
| 5.2 Policy and Attribute Elements | 127 |
| 5.3 Relations | 128 |
| 5.3.1 Assignments and Associations | 128 |
| 5.3.2 Prohibitions (Denials) | 131 |
| 5.3.3 Obligations | 132 |
| 5.4 NGAC Decision Function | 133 |
| 5.5 Delegation of Access Rights | 134 |
| 5.6 NGAC Administrative Commands and Routines | 134 |
| 5.7 Arbitrary Data Service Operations | 136 |
| 5.8 NGAC Functional Architecture | 137 |
| 5.8.1 Resource Access | 138 |
| 5.8.2 Administrative Access | 139 |
| 5.9 Conclusion | 140 |
| References | 141 |
| 6. ABAC Policy Verifications and Testing | 142 |
| 6.1 Introduction | 142 |
| 6.2 ABAC Policy Classes | 143 |
| 6.2.1 Static Policy Class | 143 |
| 6.2.2 Dynamic Policy Class | 144 |
| 6.2.3 Historical Policy Class | 145 |
| 6.3 Access Control Safety and Faults | 146 |
| 6.4 Verification Approaches | 149 |
| 6.4.1 Model Verification | 150 |
| 6.4.2 Coverage and Confinements Semantic Faults | 155 |
| 6.4.3 Property Confinement Checking | 156 |
| 6.4.4 Implementation Test | 158 |
| 6.5 Implementation Considerations | 158 |
| 6.6 Verification Tools | 160 |
| 6.6.1 Multiterminal Binary Decision Diagrams | 160 |
| 6.6.2 ACPT | 161 |
| 6.6.3 Formal Methods | 161 |
| 6.7 Conclusion | 162 |
| References | 163 |

| | |
|---|-----|
| 7. Attribute Consideration | 165 |
| 7.1 Introduction | 165 |
| 7.2 ABAC Attributes | 165 |
| 7.3 Consideration Elements | 166 |
| 7.4 Preparation Consideration | 168 |
| 7.4.1 Subject Attribute Preparation | 169 |
| 7.4.2 Object Attribute Preparation | 170 |
| 7.4.3 Environment Condition Preparation | 172 |
| 7.4.4 Metadata | 173 |
| 7.5 Veracity Consideration | 174 |
| 7.5.1 Attribute Trustworthiness | 175 |
| 7.5.2 Attribute Value Accuracy | 176 |
| 7.6 Security Consideration | 177 |
| 7.6.1 Attribute-at-Rest | 177 |
| 7.6.2 Attribute-in-Transit | 178 |
| 7.7 Readiness Consideration | 179 |
| 7.7.1 Refresh | 179 |
| 7.7.2 Synchronization | 180 |
| 7.7.3 Cache | 180 |
| 7.7.4 Backup | 181 |
| 7.7.5 Log | 181 |
| 7.8 An Example of a General Attribute Framework | 181 |
| 7.9 Attribute Evaluation Scheme | 188 |
| 7.9.1 AES Examples | 188 |
| 7.9.2 Attribute Practice Statement | 190 |
| 7.10 Conclusion | 190 |
| References | 192 |
| 8. Deployments in Application Architectures | 193 |
| 8.1 Introduction | 193 |
| 8.2 ABAC for Distributed Systems | 193 |
| 8.2.1 Access Control Challenges of Distributed Systems | 194 |
| 8.2.2 BigData Access Control as a Distributed System Access Control Example | 196 |
| 8.2.3 Implementation Considerations | 204 |

| | |
|---|------------|
| 8.2.4 Analysis and Conclusions | 206 |
| 8.3 ABAC for Web Services | 206 |
| 8.3.1 Web Services— A Brief Background | 207 |
| 8.3.2 ABAC Suitability for Web Service Environments | 207 |
| 8.3.3 ABAC for Web Service Environments without Workflows | 209 |
| 8.3.4 ABAC for Web Service Environments with Workflows | 220 |
| 8.3.5 Combined Challenges in Using ABAC for Web Service Environments (with and without Workflows) | 225 |
| 8.3.6 Web Services Environment—Summary of Requirements | 227 |
| 8.4 ABAC for Stand-Alone Workflow Processes | 228 |
| 8.4.1 Challenges and Requirements for ABAC Configuration for Stand-Alone Workflow Processes | 229 |
| 8.4.2 ABAC Deployment for Stand-Alone Workflow Processes: Integrated Approach | 231 |
| 8.4.3 ABAC Deployment for Stand-Alone Workflow Processes: Loosely Coupled Approach | 236 |
| 8.4.4 Analysis and Conclusions | 238 |
| References | 239 |
| 9. ABAC Life-Cycle Issues: Considerations | 242 |
| 9.1 Introduction | 242 |
| 9.2 Enterprise ABAC Concepts | 243 |
| 9.2.1 Enterprise ABAC Policy | 244 |
| 9.2.2 Attribute Management in Enterprise ABAC | 246 |
| 9.2.3 Access Control Mechanism Distribution in Enterprise ABAC | 247 |
| 9.3 ABAC Enterprise Considerations | 250 |
| 9.3.1 Initiation Phase Considerations | 252 |
| 9.3.2 Acquisition/Development Phase Considerations | 263 |
| 9.3.3 Implementation/Assessment Phase Considerations | 273 |
| 9.3.4 Operations/Maintenance Phase Considerations | 275 |
| 9.4 Conclusion | 277 |
| References | 278 |

| | |
|--|------------|
| 10. ABAC in Commercial Products | 280 |
| 10.1 Introduction | 280 |
| 10.2 Axiomatics Data Access Filter | 282 |
| 10.2.1 Product Architecture and Modules | 282 |
| 10.2.2 Canonical Features in Product Modules | 283 |
| 10.3 Jericho Systems EnterSpace 9 | 285 |
| 10.3.1 Product Architecture and Modules | 285 |
| 10.3.2 Canonical Features in Product Modules | 286 |
| 10.4 NextLabs ABAC Solution | 287 |
| 10.4.1 Functional Architecture and Components | 287 |
| 10.4.2 Canonical Features in Product Modules | 289 |
| References | 291 |
| 11. Open Source ABAC Implementations: Architecture and Features | 292 |
| 11.1 Introduction | 292 |
| 11.2 NGAC PM: Functional Architecture | 293 |
| 11.3 NGAC PM: ABAC Model Definition Capabilities | 294 |
| 11.4 NGAC PM: Access Decision Process | 295 |
| 11.5 NGAC PM: Design and Application Integration | 300 |
| 11.6 Summary and Analysis | 302 |
| References | 307 |
| About the Authors | 309 |
| Index | 312 |