USING ATTRIBUTE-BASED ACCESS CONTROL TO ENABLE ATTRIBUTE-BASED MESSAGING

BY

FARIBA MAHBOOBE KHAN

B.S., Bangladesh University of Engineering and Technology, 2004

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

Attribute Based Messaging (ABM) enables message senders to dynamically create a list of recipients based on their attributes as inferred from an enterprise database. Such targeted messaging can reduce unnecessary communications and enhance privacy, but faces challenges in access control. In this work we explore an approach to ABM based on deriving access control information from the same attribute database exploited by the addressing scheme. We show how to address three key challenges. First, we demonstrate a manageable access control system based on attributes. Second we show how this can be used with existing messaging systems to provide a practical deployment strategy. Third, we show that such a system can be efficient enough to support ABM for mid-size enterprise. Our implementation can dispatch ABM messages approved by XACML review for an enterprise of at least 60,000 users with only seconds of latency.

To Father and Mother.

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my advisors, Carl Gunter and Himanshu Khurana who read my numerous revisions and helped make some sense of the confusion. I would like to thank Rakesh Bobba and Omid Fatemieh, whom I worked with on this project, for all the help and support. This material is based upon work supported by the Office of Naval Research under Award No. ONR N00014-04-1-0562. Thanks to the University of Illinois for providing me with the financial means through Sara and Shohaib Abbassi Fellowship to complete this project. And finally, thanks to my husband, sister, parents, and numerous friends who endured this long process with me, always offering support and love.

Table of Contents

List of Tables

List of Figures

1 Introduction

In access control and policy management, the inherent flexibility and intuitiveness of attributes seem to be of much value and at the same time also offer some challenges. In the project, *Attribute Based Messaging (ABM)* [BFK+06], we explored attributes in two ways. First, we explore their flexibility for a more targeted messaging system. Second, we use attribute based access control to have fine-grained policies on user attributes and corresponding values.

Attribute-based systems are useful in practice because they are flexible, intuitive, and highly deployable. A common example is attribute-based directory searching where the attributes of an employee (e.g., department, location) are used to find the employee. In this example the flexibility comes from the ability to combine $\langle$*attribute, value*$\rangle$ pairs arbitrarily and intuitiveness comes from a common understanding of employee attributes. In general, attribute-based systems are deployable because most attributes associated with an enterprise are already present in various enterprise databases and assigned to enterprise users; e.g., in LDAP directories for the example above. Other examples of attribute-based systems include attribute-based authentication, access control, and trust negotiation [LMW02, BS02, YWS03, WWJ04, DVS05].

An application that can benefit greatly from integration with an attribute-based system is multi-party email messaging in an enterprise. Today, mailing lists are used to provide such messaging and while they enable a single sender to communicate with a large number of recipients, they also lead to email inboxes filled with many messages that do not interest the recipient. This is often caused by the fact that the recipient lists are overly broad. For example, if the University of Illinois wishes to send an email to all of its faculty on sabbatical, it is likely to do this by sending it to all faculty and including a body that indicates that the message only applies to the ones on sabbatical. In principle,

it would be possible to use a database to find out who is on sabbatical and use this to create a mailing list, but this may seem like a hassle given the system staff time required to accomplish it. So it is much easier to *s*pam a large number of recipients just to reach the subset that actually need to see the message. Technically spam is unsolicited commercial messages and spam filters accomplish a very good job of detecting most of it. But enterprise emails sent to large number of people, where they trust the sender and need to open and read the email to realize that it is not related to their interest, can be argued to be more distracting than spam.

ABM is the concept of allowing lists of messaging recipients to be formed dynamically by using an attribute-based recipient address. This approach brings the flexibility of attributes in enabling the sender to send targeted messages, which saves the recipient from receiving unwanted messages thereby enhances message relevance and also enables the sender to send sensitive messages knowing that the messages would be delivered only to the intended recipients thereby enhancing privacy. For example, faculty and students on a disciplinary committee can communicate with each with ensured privacy as the email can sent only to members of that committee. Also faculty and staff on a collaborative project with multiple sub-groups can discuss matters over email by being able to target the actual group members who might be spread across campus. In both of these examples users would not prefer to send the email to a broader group of audience as the content of the email might be sensitive to broadcast.

The approach also brings the intuitiveness of attributes as enterprise users typically understand the attributes associated with the users of their enterprise. So the user is in his natural domain when composing a massage with an address consisting of a query that returns the email addresses of the faculty on sabbatical. This also would save about 6 out of every 7 professors the hassle of deleting a message that does not apply to them. Furthermore, this concept can be applied to any collection of attributes that are available in an enterprise database to which the mailing mechanism can be linked. For instance, it might be possible to send a message to all of the female CS graduate students who have passed their qualifying exams to tell them about a fellowship opportunity

that has these requirements. ABM holds the opportunity to be make more efficient use of recipient time than broadcast messages or even specialized bulletin boards or web pages.

Practical ABM raises some interesting challenges, however. To identify these challenges we first consider a possible ABM development and deployment path. An initial step would be the collection of enterprise attributes and their assignment to users in a database, or perhaps, a single view of such user-attribute assignments from a collection of databases as offered by a data services layer. The next step would be to set up an ABM server and associate it with a domain Mail Transfer Agent (MTA) for compatibility with current SMTP systems much like the mailing list servers of today. This ABM server would be responsible for resolving attribute-based messages to a list of email addresses from the database(s). The next step would be to provide an interface to clients to compose and send messages to attribute-based addresses (*ABM addresses*). It is at this step that we find the interesting challenges of ABM, which primarily have to do with the security and privacy of deployed ABM systems.

First, there is the challenge of finding a manageable way to deal with access control. If anybody can send a message based on any set of attributes, this may increase rather than decrease the number of unwanted communications. It also entails some privacy issues. For instance: who, if anyone, is allowed to send an email message to faculty that make a salary of more than $150,000? If these concerns make it too difficult for an organization to decide on a policy for access to ABM, then ABM will not be useful. Second, there is the challenge of finding a plausible deployment avenue for ABM that allows the clients to send and receive attribute-based messages with restricted access policies via the enterprise messaging system. If each Mail User Agent (MUA) client or enterprise MTA must be modified to incorporate ABM, then this will be too expensive for deployment in the foreseeable future. Third, there is the challenge of making the system usable. The regular user should not feel intimidated by the system. The integration of ABM to current email should be in such a way that user observes minimum increase in effort. Forth, there is the problem of making ABM sufficiently efficient. Since each message address entails an access control

decision and dynamically forming a set of recipients, there is a serious question about whether users will loose patience or MTAs will be overwhelmed.

In this work we address these four security and privacy challenges by employing an Attribute-Based Access Control (ABAC) approach integrated into an architecture focused on deployability and usability. We then implement a prototype and conduct experiments that demonstrate the efficiency of our solution. This work is described in the following six chapters. In the second chapter we outline related work on access control and targeted messaging. In the third chapter we outline our approach for a practical access control system, discuss how the ABAC approach is suitable for ABM and finally describe our architecture for ABM using ABAC and eXtensible Access Control Markup Language (XACML) with off-the-shelf email MUAs and MTAs. In the forth chapter we describe our implementation and look at measures of its performance for various types of policies. In the final two chapters we discuss about different aspects of ABM and make conclusions, including limitations of our current work and possible future work.

2 Related Work

ABM covers different sub areas within security. We discuss three of areas of related work: access control, secure messaging and Enterprise messaging.

2.1 Access Control

We discuss access control models, languages and tools in this section.

2.1.1 Models

Access Control List

Access Control Lists (ACL) associated with an objects are lists of users and groups of users and their access permissions for that object that define how they can access that object. ACLs would not be a good policy model for ABM since the creation and management of such lists for a potentially large number of attributes (resources and users) would be unwieldy. ABM would require a policy set that defines the access control for each user attribute with n users and m attributes per user. This leads to an $O(mn)$ sized matrix. Clearly, this makes ABM with ACLs very hard to manage and less adaptable.

Role-Based Access Control

Role-Based Access Control (RBAC) [FK92, San98, SCFY96] was introduced as an alternative to Mandatory Access Control (MAC) and Discretionary Access Control (DAC). RBAC was motivated by the hierarchical job or role structure within organizations where permissions are essentially part of the role rather than the person assigned to that role. When a person is assigned to a different role his access permissions change accordingly. RBAC manages access by maintaining user-to-role and role-to-permission mappings. This way RBAC

has fewer relationships to maintain compared to user-object-permission tables in ACL. RBAC can model much more complex access policies than traditional ACLs.

Basic RBAC defines sessions where users are assigned roles and the roles are assigned specific permissions for that session. This mapping could vary in a different session. For example, in an organization after office hour roles and permissions would be much more restricted than those during office hour.

RBAC is very expressive and at the same time a very costly and administratively demanding system. RBAC can have constraints that make it very flexible. For example, a user cannot have any other role assigned when he has the role *president*. Also time related constraints such as a user can be assigned role *office staff* from 8 a.m. to 5 p.m. only. Support for these constraints add up to cost of realizing a RBAC system. Thus if used for ABM RBAC could be a potential bottleneck for performance.

Attribute-Based Access Control

In Attribute-Based Access Control (ABAC) access to service or resource is granted based on rules applied on attributes of the requester. ABAC has recently proven to be successful in access control for distributed systems [BS00, BS02, DVS05, LMW02, WWJ04, YMW00, YWS01, YWS03]. ABAC has the strength of modeling RBAC-like access with less management complexity [YT05]. As discussed earlier, RBAC has to maintain user-to-role and role-to-permission mappings. ABAC manages attribute-to-permission mappings only. Attributes are application specific in ABAC. The system doesn't need to depend on predefined set of roles or try to define role in an environment where it is not feasible, (`idsynch.com/docs/beyond-roles.html`).

Defining ABAC for our ABM system, a message sender is granted the permission to send messages to a set of recipients with a collection of attributes based on his own collection of attributes. This approach has two advantages in terms of manageability. First, since the ABM systems extracts attributes from enterprise databases for addressing purposes, using ABAC allows us to derive access control information from the same databases. Second, like Role Based

Access Control (RBAC) [FKR03], ABAC simplifies assignment and revocation of permissions. However, since ABAC uses attributes directly it avoids the need to set up and manage a role administration system that is needed for RBAC.

2.1.2 Languages

KeyNote

KeyNote [BFK99, BIK00] is a framework as well as a language to build trust-management systems. Trust management unifies the notions of security policy, credentials, access control, and authorization. KeyNote Toolkit (`cis.upenn.edu/~keynote/Code/keynote.tar`) is C language open source reference implementation of KeyNote. KeyNote has been used to develop a secure distributed file system [MPI+03] and the OpenBSD's IPsec stack [BIK02]. Apache-SSL can also be configured to use KeyNote.

RT

RT [LMW02, LM03b] is a role-based trust-management framework. It provides a policy language, semantics, deduction engine and strongly typed credentials through vocabulary agreement. RT uses Constraint Datalog [LM03a], which is expressive and also preserves the properties of Datalog. RT started as collaboration with and ABAC project at Network Associates Laboratories about credential-based trust negotiation and was used as the policy language for the project [WL02b, WL02a]. There is a basic implementation of RT written in Java used in this project.

XACML

XACML [LPL+03, God03] is an industry-standard XML access control markup language. It provides the language for administrators to define access control requirements specifically for their applications and resources in terms of policies in XML. XACML is not access control model-specific, but it is flexible enough to support most models and extensible enough to accommodate new models. OASIS Open E-business Standards (`www.oasis-open.org`) supports and maintains XACML. The current standard XACML 2.0 has specific language support

for RBAC, SAML and XML digital signaturea. XACML has both a Java and a C# implementation for their policy engine.

There are several advantages of XACML over KeyNote and RT for our practical demonstration in ABM. First, XACML lends itself very well for ABAC policy specification as the framework supports attributes. Second, the XACML standard has widespread support from industry and standards bodies and this may support adoption. Third, its successful integration in several commercial products [And05] as well as research projects [LPL+03] indicates the confidence in its deployability and effectiveness. And finally XACML is platform independent and easily integrated in heterogeneous systems.

2.1.3 Tools

XACML.NET

XACML.NET (`mvpos.sourceforge.net`) is an XACML implementation in C# for .NET community licensed under Mozilla Public License (MPL) 1.1. It provides almost full support for XACML v1.0 and has started providing support for some features in XACML v1.1. Implementation on XACML.NET v2.0 is also going on. The implementation, however, lacks extensive documentation and support.

Sun's XACML Implementation

Sun's XACML implementation (`sunxacml.sourceforge.net`) is an open source implementation of XACML standard writhen in Java. It provides complete support for XACML 1.2 and work on XACML 2.0 is on progress. The development team closely reviews and updates any changes required to the implementation. It has several policy decision engine modules of varying complexity. It has tools for creating XACML policy-sets and requests pertaining to the standard. Tools for authoring, debugging and visualization are under development.It has an active online blog for support and bug tracking.

Margrave

Margrave is (`cs.brown.edu/research/plt/software/margrave`) a PLT Scheme API for use in analyzing access-control policies written in a subset of XACML. Margrave parses an XACML policy and answers questions regarding the permissions allowed by that policy. It can also compare two policies and mark down the dissimilarity between them [GMMT05, FKMT05]. As a tool Margrave has been proved to be sound and complete [GMMT05] but it does not verify correctness of XACML syntax. It answers questions such as,

> `Can full-time faculty send email to department chairs?`
>
> `Do all changes between` *`policy1`* `and` *`policy2`* `include email permissions to committee chairs?`

Margrave can be used by ABM administrators for XACML verification and policy analysis.

2.2 Secure Messaging

2.2.1 PKI-Based Email Security

Public-key based email security solutions such as PGP (Pretty Good Privacy) [Zim95] and S/MIME (Secure/Multipurpose Internet Mail Extensions) (`ietf.org/html.charters/smime-charter.html`) provide end-to-end messaging security in terms of confidentiality, integrity and authentication. Secure Email List Services (SELS) [KSB05] provides the same for emails through listservs. ABM, on the other hand, focuses towards a different direction. It concentrates on policy-based dynamic lists and access control over such lists.

2.2.2 Role-Based Messaging

Secure role-based messaging uses RBAC for authorizing access to sensitive email content [CLZ04, MBH03]. In this area [CLZ04] allow users to send messages to a given role identified by a special email address. Users that are assigned to that role can then provide their role membership credentials and access the email.

Using a slightly different approach [MBH03] employs Identity Based Encryption (IBE) for encrypting messages to recipients; *i.e.,* recipient must authenticate themselves to a role administration system and obtain the email decryption keys. These two approaches differ from ABM access control by focusing on the access control rules for *recipients*, whereas we focused on access control rules for *senders*. Of course, they also differ in the use of roles rather than attributes as a foundation for policies.

2.2.3 Adaptive Messaging Policy

The Adaptive Messaging Policy (AMPol) project, of which this paper is a part, has considered some technologies related to ABM. WSEmail is the idea of building messaging systems over a web services foundation. A prototype [LMBG05] of such a system demonstrated messages that could be routed with addresses that are determined dynamically as the message passes through WSEmail MTAs. However, this system does not decide on recipients based on their attributes. A WSEmail-based design [AZHG06] shows how to adapt to recipient policies as part of messaging, but this design does not deal with multiple recipients. Other details on AMPol, including a demonstration of our ABM system, can be found on the AMPol web site (`seclab.cs.uiuc.edu/ampol`). Though ABM and WSEmail both concentrate on adaptive messaging, WSEmail redefines email architecture for more flexibility and ABM focuses on building on existing architecture for further usability.

2.3 Enterprise Messaging

Perhaps the most similar technology to ABM arises in Customer Relationship Management (CRM) systems. CRMs help enterprise to target customers by isolating specific buying patterns and using this to customize the communication with them. The key difference between CRMs and ABM is that in CRMs the communication is from the enterprise to the customer group and so there is no need for access control. Where as in ABM messages are sent by users to other users after access is determined by the attributes of the sender. In

other words, CRM generally uses a monolithic permission given to the owner of the system, whereas ABM provides diverse permissions to a broad user group. Traditional list servers also provide a way to send email messages to a certain group of people. One can imagine driving membership in lists from a database of attributes to provide a form of ABM. For example, SendMail (a popular MTA) can be integrated with LDAP but it lacks a mechanism to control the use of such mailing lists. A key difference between ABM and list servers is the fact that ABM has the potential to route on *involuntary* attributes of recipients rather than relying solely or mainly on voluntary subscriptions. A good potential use of ABM is to provide a way for users to subscribe to lists automatically and voluntarily by collecting a user profile of interests.

3 ABM Design

3.1 Approach for Practical Access Control

An attribute-based messaging system comprises an enterprise attribute database that provides user to attribute mapping functionality, a query language and composition mechanism that enables senders to compose ABM addresses, a bridging mechanism that connects the ABM system with the enterprise messaging system, an ABM server that provides service to all enterprise users and related components, and the access control component. The access control component is needed to ensure that the sender is authorized to send the message to the set of recipients represented by their collective attributes in the composed address. The absence of access control would allow senders free and easy access to all enterprise users' email inboxes and would also violate the privacy of user attributes. Note that this privacy is currently enforced in enterprise databases by allowing only authorized administrators access to them. When attributes are made available to users in the ABM system, it is essential that the privacy of the attributes be enforced via appropriate access control. To do so, the access control system would comprise a policy language that enables administrators to specify policies, a policy engine that acts as the Policy Decision Point (PDP) by evaluating specified policies against a given access request, and a Policy Enforcement Point (PEP) that enforces the decision. In the ABM system the ABM server acts as the PEP. As identified in the Introduction, practical access control for ABM involves addressing the challenges of manageability, deployability, and efficiency.

In order for the access control system to be manageable it must use access control techniques that specify an efficient mapping of permissions to services (*i.e.*, the ability to send messages to a set of recipient with a given collection of

attributes). As discussed, to address this we turn to ABAC, which has recently proven to be successful in access control for distributed systems [BS02, DVS05, LMW02, WWJ04, YWS03].

For server-side deployability on a variety of messaging environments the access control system must employ a usable, standardized policy language and a standards-based implementation of a policy engine. To address this our architecture and prototype are based on XACML [God03] and Sun's standards-compliant implementation of its policy engine (`sunxacml.sourceforge.net`) as discussed in Chapter 2.

For client-side deployability the access control system must enable the sender to compose an attribute-based message that complies with the access policy using almost any existing MUA. To address this we use *policy specialization* techniques where the sender logs into a web server to compose an ABM address using only those attributes that he is allowed to route on; *i.e.*, the composition of an ABM address is limited to attributes based on the access policy for the sender. This ABM address is returned to the sender in a file that he can then attach to his message, which is addressed to a pre-specified email address of the ABM server. Furthermore, the ABM address is integrity-protected and securely bound to the sender's email account so that it cannot be spoofed or replayed. This approach also provides email semantics that users are familiar with in that once they compose and send a message they expect the message to be delivered. Other approaches for addressing this challenge can also be envisioned; *e.g.*, the development of MUA plug-ins that can access enterprise attributes and understand and enforce access policies. However, a major advantage of our approach of setting up a web server is that we avoid the need for developing multiple plug-ins for different MUAs as well as requiring installation of additional software on the client side. Also web-based email client can incorporate all these features in easy-to-use web-based tools. It is more challenging to make to incorporate ABM in day-to-day email clients.

The efficiency of the access control system can only be gauged via prototype implementation and experimentation. With an eye towards rapid prototyping and performance we have employed several commercial of-the-shelf compo-

nents that are well-implemented and standards-compliant including, for example, Sun's XACML policy engine. In our implemented solution a a user accesses a web page to create an ABM address that further makes a request; our policy engine *specializes* the organizational policy to this user, indicating the attributes that the user can use for routing. The user then forms the desired attributes into an address, which is represented using a query language. This query is added to a message as an attachment and sent to a distinguished ABM address at an MTA using the user's standard MUA. The ABM system collects the email from this distinguished inbox and dynamically creates a distribution-list using the attached query and the enterprise attribute database. With an enterprise of 60,000 principles using its existing enterprise database or an XML database view of it, we are able to show that both the XACML decision procedure and the dynamic list creation can be one within seconds in typical cases, and will still have satisfactory performance for emerging XML database representations that integrate heterogeneous enterprise databases.

3.2 ABAC for ABM

In this section we describe how ABAC is employed to provide manageable access control for ABM. All enterprises have attribute data about their users in their databases. For example, a university might have the following attribute data on a user represented as $\langle attribute,\ value \rangle$ pairs:

UserID: user089

Position: Faculty

Designation: Professor

Department: Computer Science

Courses Teaching: CS219, CS 486

Date of Join: 06/24/1988

Annual Salary: $80,000

. . .

This information may not all be available in one centralized database but, instead, might be distributed over multiple databases that are managed by different units of the University. Our ABM system makes use of this information, present in an enterprise's collective databases, abstracted as user attributes to dynamically create recipient lists. To have this attribute information available to the ABM system we envision the use of a data services layer (dubbed *information fabric* by Forrester Research [YGHS06]) that exemplifies the Service Oriented Architecture (SOA) approach [BSJ+05] and presents a view of the attribute data after extracting it from the disparate databases.

To send an attribute based message to a group of recipients a user needs to specify the attributes in a logical expression. For example the expression *((position=faculty) and (salary>$150000))* defines a group that constitute faculty who make a salary of more than $\$150,000$. This expression is referred to as an *ABM address* and, in practice, can be specified using the language of the database (*e.g.* SQL) or via a commonly used query language that can be executed on a variety of database technologies (*e.g.* XQuery (`www.w3.org/XML/Query/`)).

A user is permitted to send a message to a given ABM address based on his/her attributes. For example, only a user who has the $\langle$*attribute, value*$\rangle$ pair $\langle$*position = faculty*$\rangle$ or the pairs $\langle$*position = staff*$\rangle$ and $\langle$*designation = coordinator*$\rangle$ (*i.e.,* only faculty or coordinators), might be allowed to send messages to the ABM address (*position = faculty*) (*i.e.,* all faculty). We specify access policies as well as ABM addresses in disjunctive normal form to make them flexible and intuitive. Specifically, access policies take the following form:

$cond \Rightarrow \langle attribute, (value) \rangle$;

i.e., if the condition *cond* is satisfied then

"access" is granted to $\langle (attribute, value), \rangle$

where:

$(value)$ is a set of discrete or enumerated values $(value_i, value_j, \ldots, value_n)$,

$cond = (Term_1)\ or\ (Term_2)\ or \ldots (Term_n)$,

$Term_i = (literal_1)\ and\ (literal_2)\ and \ldots (literal_m)$,

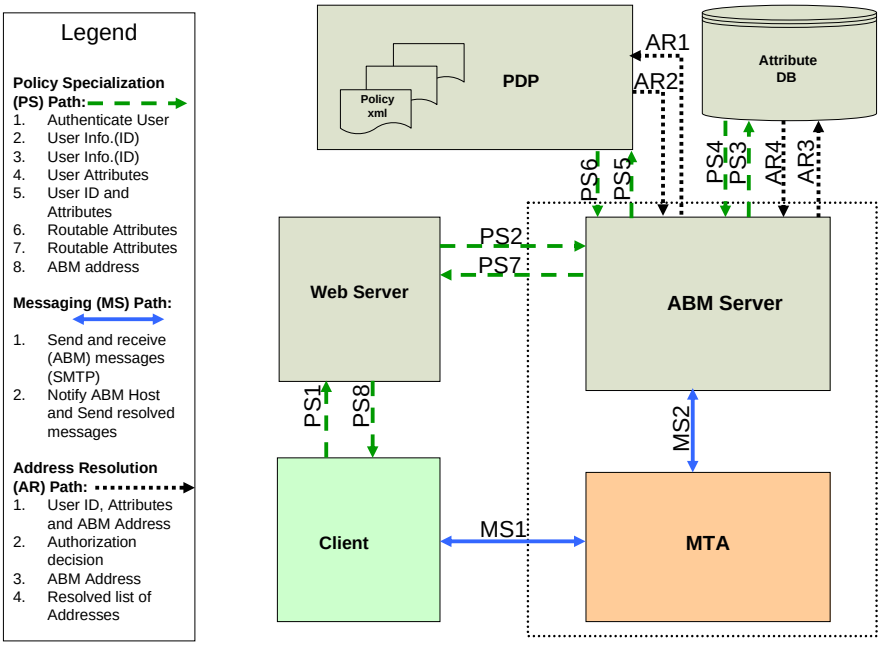
$literal_j$ = *(attribute <arg> value)*, and

arg is one of $=, <, >, \leq$ *or* $\geq$.

Therefore, we argue that the access rules can express a variety of policies and, similarly, an ABM address can specify almost any arbitrary group based on attributes. ABAC policies in ABM have similarities and differences with those of more traditional enterprise services; *e.g.*, file access or web services [YT05]. They are similar in that just like attributes may be mapped to file access permissions in file systems, they would be mapped to the routable attribute. So, the ABAC policy for the above example would grant "access" to the ⟨*attribute, value* ⟩ pair ⟨*position = faculty*⟩ if the following expression of ⟨*attribute, value*⟩ pairs is satisfied: ⟨*position = faculty*⟩ *or* ⟨*position = staff*⟩ *and* ⟨*designation = coordinator*⟩.

They are different because unlike files one can envision granting access to an ABM addresses that combine various attributes in a logical expression. The equivalent notion in file systems would be to have a policy that grants access specifically to text that is common to two given files, which is a level of granularity not seen in practice. Clearly, even in ABM specifying a unique access policy for every possible ABM address is not practical. To address this issue, we take a simplifying, pragmatic approach: a user is allowed to send messages to any combination (using *logical and* and *logical or* operands) of ⟨*attribute,value*⟩ pairs if she can send messages to those pairs individually. This turns out to be a reasonable approach because instead of choosing the *or* operand the sender can easily send out multiple emails to achieve the same effect and when the sender chooses the *and* operand she only ends up targeting his email to a narrower set of recipients than she is allowed to. Therefore, at most one access policy is required for each ⟨*attribute,value*⟩ pair. In practice, there are various ways to reduce the number of policies, some of which are explored in Chapter 5.

Figure 3.1: ABM Architecture



3.3 Architecture

Figure 3.1 illustrates the architecture of our ABM system and its associated access control system, which strongly influences the overall structure. The ABM system comprises a web server to help users compose policy compliant ABM addresses, a PDP along with the access policy, an attribute database, and an ABM server associated with an enterprise MTA that resolves ABM addresses to recipient lists and mediates other components. The message flows in our system can be classified into three functional classes, *viz.,* Policy Specialization Path, Messaging Path and Address Resolution Path. We now describe these flows in detail.

3.3.1 Policy Specialization (PS) Path

This path refers to the message flow in the system when a user logs into the web server to compose policy compliant ABM addresses. These messages are represented by dashed lines in Figure 3.1. In step one the user authenticates herself to the web server. In step two the web server sends the user's information

to the ABM server and requests for a *specialized policy* for the user. In steps three and four the ABM server retrieves user's attributes from the attribute database. In step five the ABM server sends the user's attributes to the PDP and requests a specialized policy. The PDP then evaluates all the policies in a policy file against the user's attributes and returns the specialized policy, *viz.,* a list of ⟨*attribute, value*⟩ pairs that the user can *route on*. The ABM server then returns the specialized policy to the web server in step seven. The user then composes an ABM address and downloads it in step eight. ABM addresses created using the web interface include user's *e-mail id*, are time-stamped, and are integrity protected using a SHA-1 Hash MAC. Messages using freshly composed ABM addresses aren't subject to an access policy check at the ABM Server, in order to reduce the burden on the PDP (*e.g.*, within 24 hours; note that extent of freshness is a system parameter and should be based on the dynamic nature of policy and user attributes).

3.3.2 Messaging (MS) Path

This path is represented by solid lines in Figure 3.1. Users send ABM messages using any standard MUA [1] to a pre-specified e-mail address such as `abm@localdomain.com`, with the ABM address included in the message as an attachment. The enterprise MTA is configured to notify the ABM Server when it receives a message for the pre-specified address. The ABM server after processing the message invokes the enterprise MTA to deliver the message to a list of recipients as specified by the ABM address.

3.3.3 Address Resolution (AR) Path

This path refers to the message processing by the ABM server and is represented by dotted lines in Figure 3.1. The ABM Server, on receiving the (e-mail) message, verifes the Hash MAC on the ABM address, verfies that the *from address* in the message is same as the *email id* included in the ABM address, and queries the attribute database for the sender's attributes. In step one, the ABM server checks with the PDP that the sender is authorized to send the message

[1] ABM system can easily be integrated with web-based e-mail but for generality we assume the presence of an email client like Outlook.

to the ABM address included in the message. In step two, the PDP evaluates the policies for accessing the attributes contained in the ABM address against the sender's attributes and responds in the affirmative only if the user is allowed access to all attributes in the ABM address. The ABM Server then resolves the ABM address to a list of e-mail addresses by querying the attribute database in steps three and four. It then forwards the message to each member in the list via the enterprise MTA.

3.3.4 Security Analysis

Analyzing the proposed architecture, one can see that the ABM system as described above is open to replay attacks. A malicious user can steal an ABM address, composed by a legitimate user in step PS8, either on the network or from the user's machine and use it to route messages. This attack would be successful, even though ABM addresses are integrity protected with a Hash MAC, because the adversary can spoof the legitimate user's *email id*. So when the ABM server receives the adversary's e-mail message it believes that the sender of the message is the legitimate user (who composed the ABM address used by the adversary). Hence, there is a need for the underlying messaging system to provide the ABM server with an authenticated *email id* of the sender. Toward that end we need to do the following: (1) have the enterprise MTA invoke the ABM server only for messages originating inside the enterprise, (2) require SMTP authentication at the enterprise MTA, and (3) ensure that the *user id* used in SMTP authentication and *from address* of the message being sent are the same. Step one ensures that only enterprise users can use the ABM system and can be achieved using mail filters. Steps two and three ensure that the *from address* in the received e-mail message is authentic. Popular MTAs like SendMail support SMTP authentication and step three can be achieved using mail filters.

3.3.5 Usecase Scenario

We now provide a usecase scenario for ABM. Lets assume a Professor Carlson in University of Illinois gets to know about a fellowship opportunity by Megasoft

for graduate students in computer science. The fellowship has some requirements for the students to be eligible to apply for it. The fellowship is for female graduate students who have passed their PhD qualifiers but are yet to take their preliminary exam. The department maintains a list of graduate students. But there is no such list that could target the audience of the Megasoft fellowship. University of Illinois has 40,000 students, 10,000 of them are graduate student and 500 of them are in computer science. The number of emails each day a graduate student gets related to course-work, academic deadlines, thesis deadlines, seminars, fellowship and job opportunities can range from 10 to 20. Not all of these are of interest for all the students. In this scenario Prof Carlson can use ABM to target this message better so that only concerned students receive it.

First, he logs on into ABM web-client using his userid and password. If this is his first log-in in days, the policy specialization path is completed. Prof. Carlson's user information is sent to the ABM server and the ABM server gets his attributes from the database. His attributes will have information such as, full-time faculty who is in position Professor, courses he is teaching (CS431, CS591), and committees he is on (graduate admission, fellowship). PDP will then check his access based on his attributes and a specialized policy for him will be returned eventually to the client application. The specialized policy will have information on attributes and values he can route on, *e.g.*, he can route on attribute *course* on values CS431 and CS591 only, he can route on attribute *gender of students* on any value. If Prof Carlson used ABM recently and cached his specialized policy (which has not yet expired) he can use that. Prof. Carlson now composes an ABM address that specifies female, post-qualifier, pre-prelim, computer science, graduate students as receivers. It has his email id (`carlson@uiuc.edu`) and is time-stamped. He archives this ABM address in his computer.

Prof Carlson now opens his email client (*e.g.*, Eudora), composes the body of the email, attaches the saved ABM address and sends it to (`abm@uiuc.edu`). University of Illinois MTA sends the email to ABM server.

ABM server, on receipt of the email, downloads the ABM address and upon verifying its freshness and Prof. Carlson's permissions through PDP to send

such message, queries the database for email address of such users and gets 15 email addresses. The email is sent to these students through the enterprise MTA.

4 Implementation and Evaluation

To test that the architectural framework presented in Section 3.3 satisfies the manageability, deployability, and efficiency requirements for ABM, we implemented a prototype ABM system. We used this prototype implementation as a test bed for experimental evaluation. This section provides details on the prototype implementation, experimental setup, and performance results with the aim to show that ABM can satisfy the above-mentioned requirements.

4.1 Implementation

We had to make a number of decisions on the technologies and programming languages to use for the major components of our proposed architecture. These decisions, and the reasoning behind them are briefly discussed in this section.

4.1.1 PDP

As it was described in Section 3.1, we chose to use XACML and Sun's standards-compliant implementation of its policy engine for our implementation. An XACML policy file is stored in conjunction with the PDP. This policy file contains the policies for sending messages based on each $\langle attribute,\ value \rangle$ pair. Our current implementation supports numeric and enumerated attributes.

4.1.2 Database

Our system has been implemented using two different database representations, relational and native XML. We included an XML database representation in our evaluation as we envision data abstracted from heterogeneous enterprise databases to be in XML format. The queries submitted to the XML database are XQueries, and the queries for the relational database are expressed in SQL.

We had to chose a database management system with support for XML and XQuery as well as SQL. We used the recently released community technology preview release of Microsoft SQL Server 2005 (Standard Edition), which provides support for all the above mentioned data models and query languages.

4.1.3 ABM Server

The ABM server is associated with an enterprise MTA. The ABM Server gets automatically invoked when the MTA receives an ABM message targeted for the inbox associated with the ABM Server. This enabled us to use our domain MTA without any modification. We used C# to implement the ABM Server, and used the University of Illinois MTA as the enterprise MTA.

4.2 Test Bed

Studying the components in our system in Figure 3.1, we anticipated that the two major resource consuming components of our system would be the database and the PDP. Based on this assumption, we decided to place them on different machines on the network. Our prototype runs on windows client and server machines. The database was running on a Windows 2003 Server with dual Intel Xeon 3.2GHz processors and 1 GB of memory. PDP, Web server and ABM Server were cloned on two client machine. One is a 2.4 GHz Pentium 4 with 1GB of memory with Windows XP Pro operating system. The other one is a 2.8 GHz Pentium 4 with 1GB of memory with Windows XP Pro operating system connecting to the server through secure channel.

4.3 Experimental Setup and Results

The goals of our experiments were to evaluate the performance of our ABM system both with and without access control. These goals enabled us to demonstrate the feasibility of the system as well as determine the additional costs imposed by the access control component. To evaluate the performance with access control we needed to study the performance on the three paths described in Figure 3.1, namely, policy specialization, messaging, and address resolution.

To evaluate the performance without access control we needed to study the performance on messaging path and address resolution path but without the authorization check. However, since we are using the University of Illinois MTA, the performance on the messaging path is not part of the evaluation of our system, because the University of Illinois MTA will add the same latency to our messages as it would add to any regular email.

To carry out the evaluation we needed to vary three experimental components: (1) the complexity and number of access policies, (2) the number of users and their assignment to a varying number of attributes in the database, and (3) and the complexity of ABM addresses.

4.3.1 Policy Generation

The complexity and number of the access policies affects the time frame of the policy specialization path and the authorization check on the address resolution path. We wrote a probabilistic XACML policy generator using Java, which created uniformly random policies of varying complexity by varying the number of *terms* and *literals* in the conditional clause of each policy (please refer to Section 3.2 for definitions). Specifically, the number of *terms* and number of *literals* in each term were uniformly drawn between one and five, creating relatively simple to reasonably complex policies. The number of policies depend on the number of $\langle attribute, value \rangle$ pairs and we varied the number of attributes between 25 and 125 with an average of 5 values (or value ranges) per attribute for resulting policies ranging from 143 to 674.

4.3.2 Database Population

The distribution of attributes in the user population affects the number of recipients a given ABM address resolves to, which, in turn, affects the time frame of the address resolution path. Users were assigned an attribute based on the incidence probability of that attribute. For example, if an attribute has an incidence probability of 0.1 then 10% of the user population is assigned that attribute. For our test database, most of the attributes (80%), had a probability of incidence that ranged from 0.01 to 0.0001, 10% had a probability of incidence that

was between 0.5 and 0.9 and the remaining 10% had the probability close to 1. This distribution allowed a big range in the number of recipients per message, and, intuitively, this distribution also reflects organizations where all the users have some common attributes and rest of the attributes are sparsely distributed in the population. The schema below illustrates the way user's attributes data was stored in the relational database.

Relational Database Schema (assuming X variables in the system):

[userid] Primary Key, nvarchar (20)
[passwd] nvarchar (40)
[attr0] int
[attr1] nvarchar(128)
. . .
[attrX] int

For storing data in the native XML format we created a relational table, which consists of three columns. The third column contains the attribute information stored in XML format. The following schema illustrates this better.

[userid] Primary Key, nvarchar (20)
[passwd] nvarchar (40)
[attributes] XML(AttributeSchema)

AtributeSchema associates an XML Schema (`www.w3.org/XML/Schema`) with the XML values in that column.

4.3.3 ABM Address Generation

The complexity of an ABM address affects the performance on the address resolution path by affecting both the number of recipients it resolves to and the database query resolution time. Similar to our approach for policy generation we varied the number of *terms* for a given address query between one and five

Relational Database			
DB Size (No. of Users)	Avg. List Size	Address Resolution Time Mean 95% Conf. Interval (ms)	
		With Access Control	Without Access Control
60K	422	(167, 322)	(83, 244)
45K	302	(134, 294)	(65, 206)
30K	220	(117, 179)	(61, 116)
15K	145	(115, 147)	(50, 72)
XML Database			
DB Size (No. of Users)	Avg. List Size	Address Resolution Time Mean 95% Conf. Interval (ms)	
		With Access Control	Without Access Control
60K	745	(4682, 6062)	(4628, 5970)
45K	472	(3969, 4711)	(3599, 4436)
30K	317	(2640, 3217)	(2581, 3151)
15K	171	(2341, 2857)	(2067, 2624)

Table 4.1: Address Resolution Time. Number of attributes = 100; number of policies = 568.

(chosen randomly) and the number of *literals* in each term between one and three (also chosen randomly). Each literal was randomly assigned an attribute from the routable list of attributes of the message sender. The same set of ABM addresses were used to evaluate the system both with and without access control.

4.3.4 Performance Measurements on the Address Resolution Path

The performance on this path is translated to the latency between the time an ABM message is received by the ABM Server until the time the message is sent out to the MTA for distribution.

For the case with access control this latency includes the time for: (1) checking the integrity of the ABM address via HMAC verification (2) consulting the PDP for authorization (in our experiments we do an authorization check on all messages irrespective of the freshness of the composed ABM address) (3) retrieving the list of the recipients specified by the ABM address from the database, and (4) re-composing the message with the list of recipients. For the case without access control only the third and fourth latency components were included.

Relational Database			
DB Size (No. of Users)	Avg. List Size	Address Resolution Time Mean 95% Conf. Interval (ms)	
		With Access Control	Without Access Control
60K	351	(180, 359)	(97, 237)
45K	296	(153, 395)	(82, 263)
30K	168	(137, 277)	(64, 136)
15K	77	(90, 126)	(32, 46)
XML Database			
DB Size (No. of Users)	Avg. List Size	Address Resolution Time Mean 95% Conf. Interval (ms)	
		With Access Control	Without Access Control
60K	798	(6144, 7964)	(5726, 7306)
45K	469	(3970, 4750)	(3589, 4455)
30K	291	(2783, 3260)	(2593, 3157)
15K	174	(2307, 2849)	(2038, 2628)

Table 4.2: Address Resolution Time(Secure Channel). Number of attributes = 100; number of policies = 568.

We performed our tests using databases of user size ranging from 15,000 to 60,000. Each of the experiments was performed on a sample of 100 users chosen uniformly at random from the corresponding databases. Table 4.1 summarize our results. The Average List Size field in the tables refer to the average number of recipients that the ABM addresses resolved to. The ABM addresses used had 2.5 *terms* on average and each *term* had 2.5 *literals* on average. There were 100 attributes in the system and 568 policies. There were 2.5 *terms* on average per policy and 2.5 *literals* on average per *term*. It is worth mentioning that since the databases were probabilistically filled, users were randomly selected, and the queries were also probabilistically generated, we had no direct control on the average list sizes. For enterprise that have geographically dispersed locations,the ABM server and the database will not have a complete trustworthy route for communication. For this case, address resolution experiments were repeated (Table 4.2) with the ABM server communicating with the database over secure channel for the third step (retrieving the list of the recipients specified by the ABM address from the database).

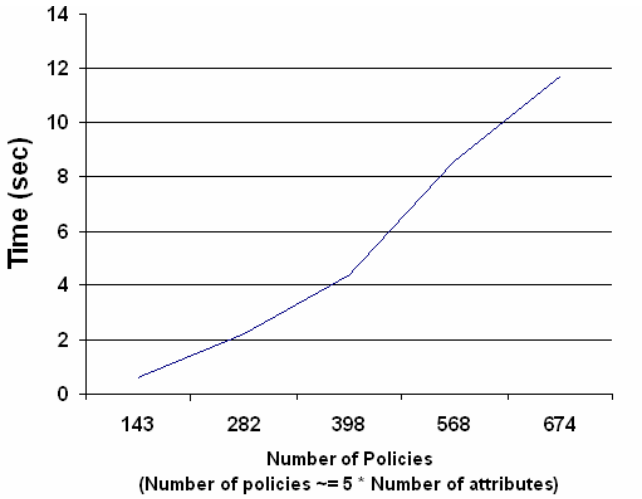


Figure 4.1: Policy Specialization Time

4.3.5 Performance Measurements on the Policy Specialization Path

The performance in this path is translated to the latency a user would see from the time she attempts to log in to the system until the time her *specialized policy* is revealed to her. This time includes: (1) a database lookup for retrieving a user's attributes and (2) a policy decision time for determining the routable attributes.

We studied the policy specialization time with regard to complexity of the policies and the results capturing the latencies are summarized in Figure 4.1. Each policy had 2.5 *terms* on average and each term 2.5 *literals* on average. Each of the experiments was averaged over 100 runs. The database used for these experiments was a relational database with 60,000 users, which was filled using the distribution described above. In each of the runs the policy specialization is performed with respect to a user chosen uniformly at random from the database.

4.4 Analysis of Results

4.4.1 Feasibility Without Access Control

As shown in Table 4.1, within domain the average latency added to an e-mail message by the ABM system (address resolution latency) without access control

is under 250ms using a relational database. It is under six seconds using an XML database. The implemented system thus can process 240 requests per minute using a relational database and 10 requests per minute using an XML database. Though the address resolution takes longer when using an XML database, we can expect that to decrease in the future as XML technology matures. Over a secure channel (Table 4.2) the system can process 230 requests per minute using a relational database and 8.2 requests per minute using an XML database.

4.4.2 Feasibility With Access Control.

As shown in Table 4.1 and Table 4.2, the average latency added to an e-mail message by the ABM system (address resolution latency) with access control is under 325ms when using a relational database and under seven seconds when using an XML database. Adding security to the system added on atmost 100ms additional latency when using a relational database and 400ms latency when using an XML database. Thus, the ABM system with security can process 185 requests per minute using a relational database and 8.5 requests per minute using an XML database. While over a secure connection it can process 150 requests per minute using a relational database and 7.5 requests per minute using an XML database. The discrepancy in latency added by security when using a relational database vs. an XML database is due to the fact that the authorization check involves one database look up and one access validation and on average an XML database look up took 350ms more than relational database lookup. Access validation through PDP takes around 60ms and gives us a throughput of 1000 validations per minute.

As expected, Figure 4.1 shows that the policy specialization time increases with the number of policies in the system. The number of policies in the system is directly proportional to the number of attributes in the system. In particular, it is equal to *number of attributes* $\times$ *average number of values/sub-ranges per attribute*. The number of values/sub-ranges per attribute was randomly drawn between 1 and 10. So we can conclude that the policy specialization time is directly proportional to the number of attributes in the system. Our experiments showed that for policy specialization, database access time remains virtually

constant regardless of the number of attributes in the system. This value is about 40ms for relational and 400ms for XML databases. This is due to the fact that each policy specialization includes a single lookup on the primary key of the database. So the observed increase in the policy specialization is due to the increase in the policy evaluation time, not the database lookup time.

Arguably, the latencies of 12 seconds might be beyond the level of patience of most of the users and also impact the scalability of the system. However, we have to keep in mind that specialized policy need not be computed every time a user wants to send a message. The ABM system could periodically, say once a week or once a month, compute the the specialized policy for all users and cache it. Re-computation between the periods will only be necessary if there is a change in the policy or users' attributes. Therefore, we conclude that even with security included the performance of the ABM system remains reasonable.

5 Discussion

In this section we discuss some of the issues that are important for usability of ABM.

5.1 Policy Administration

Specifying and managing polices can potentially be a significant burden in the deployment of our ABAC based ABM system. Even having only one access policy, for each ⟨*attribute, value*⟩ pair can lead to a large set of access policies to be managed by an enterprise policy administrator. In practice, however, most attributes do not need a separate access policy for every possible value. For example, some attributes like *address* may not need a policy for every single value as it may not be possible to even enumerate all values. For some attributes it might be possible to encode policies for all possible values of the attribute into a generic form. For example, a policy to send a message to students in a given course might be that the sender must be teaching the course. So there is no need to write a separate policy for each ⟨*course, value*⟩ pair as policies for all values of attribute *course* follow the same pattern and hence can be written as one policy. The logical form of such a policy is shown below.

$\langle request.teaching = variable_x \rangle$

$\Rightarrow \langle course, variable_x \rangle$,

where

request.teaching is requester's teaching

attribute value and

$variable_x$ is a variable that refers to

the course attribute value in the access request.

Some attributes in an enterprise might need only one access policy for each disjoint subset of possible values. For example an attribute like *Age* whose possible values are from (17,120) might need a policy only for disjoint sub-ranges like (17,30], (30,65] and (65,120). In general, we observe that any attribute that has infinite or uncountable set as the range of values and whose values cannot be grouped together in any meaningful way will have only one policy. While any attribute that divides the population into disjoint sets might need a policy for every $\langle$*attribute, value*$\rangle$ pair. We analyzed attributes in three units of University of Illinois with the above observations in mind found that only 20% of them need a unique policy for each value while for 50% of them a single policy per attribute is sufficient.

Furthermore, a single enterprise policy administrator does not necessarily need to specify and manage policies for all attributes in an enterprise. Policy administrators in each unit can be responsible for specifying and managing policies for attributes originating from their unit, thereby enabling distributed administration of access policies.

5.2 User Interface

End users cannot be expected to write database queries or logical expressions. An effective user interface for composing ABM addresses is crucial for the ABM system to be adopted. Similarly, policy administrators will benefit from a user interface for specifying policies. Though we do not address these needs in this work, user interfaces that closely satisfy the requirements are those found in web directories and catalog searches. Moreover, recent advances in natural language query interfaces such as NaLix [LYJ05, LYJ06], that enable translation of queries in English into queries in XQuery can further improve the usability of ABM system.

6 Conclusion

We have demonstrated a simple and manageable access control model for ABM based on ABAC that accommodates a useful collection of ABM applications. We have shown that this access control system can be embedded in an architecture that can be deployed in virtually any enterprise messaging system. Finally we have shown that this architecture can be implemented efficiently for mid-size enterprises and we have given a profile of policy parameters that affect its efficiency.

There are a number of interesting questions and open opportunities for ABM with ABAC. Two of these will particularly interest us for future research: inter-domain operation of ABM and more expressive ABAC policy languages. While we have shown how to architect and deploy ABM for enterprises, it is much trickier to do this when multiple enterprises are involved. For example, suppose we wish to send a message to all of the doctors in a given county. This cannot be done with a single database or even the collection of databases of a single enterprise. There is some need to map the attribute 'doctor' across multiple domains. This problem arises with virtually any interdomain authorization challenge so the problem is only illustrative, but it is perhaps more tractable for ABM than for interdomain authorization in general. Clearly some techniques are required to map attributes. We have a design for such a system assuming such a mapping is possible, but it needs to be developed and studied in the way we have approached the enterprise systems in this paper. Our ABAC policy language (implemented as a subset of XACML) is rudimentary. We choose it because it was clearly useful and yielded non-trivial questions about processing and performance. However, one can certainly imagine ABAC based ABM systems benefiting from a more theoretical analysis of policy language expressibility such as that undertaken by [LMW02, WWJ04] for distributed systems.

At the same time, it is not clear how complex a policy language should be; perhaps expressiveness is less important than the ease of maintaining policies. After all, existing systems do not offer ABM at all, so even basic functions are a step forward. Complex policies that lead to unintentional user errors would dampen enthusiasm for deployment. Nevertheless, there are a variety of interesting theoretical questions that can be considered in this area.

A Sample Policy

Figure A.1 shows the structure of our policy file specified using XACML. A policy denoted as a *Rule* in the figure consists of *Target* information and *Condition* information. *Target* information includes subject attributes in the *Condition* (not shown in figure for clarity), resource information, and the action on the resource for for which this policy applies, in our case *route*. *Condition* specifies the access condition in disjunctive normal form where the outermost *Condition Function* is *or* and inner *FunctionIds* are *and*.

In order to overcome inconsistencies (*e.g.*, different policies in a policy set evaluating to conflicting results for a given access request) in policy specification due to human error we use the *combining algorithm* feature provided in the XACML framework. A *combining algorithm* tells the policy engine how results from different applicable policies for a given access request might be combined into a single result. An example of a simple *combining algorithm* provided by XACML policy engine is *deny-overrides*, which simply tells the engine that if *deny* is the result of one of the conflicting rules then return *deny*. Sun's XACML implementation can be extended to include more complex *combining algorithms*. The combining algorithm, which in our case is *ordered-permit-overrides*, is specified at the top and the default policy, which evaluates to *deny*, is at specified at the end.

```
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
PolicyId="GeneratedPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:ordered-permit-overrides">…

 <Rule RuleId="RouteOnSalary" Effect="Permit">
   <Target>...
     <Resource>
       <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Salary</AttributeValue> ...
       </ResourceMatch>
     </Resource>  ...
     <Action>
       <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Route</AttributeValue>  ...
       </ActionMatch>
     </Action>  ...
   </Target>
   <Condition FunctionId=="urn:oasis:names:tc:xacml:1.0:function:or">…
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">…
            <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="Designation" /> …
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Manager</AttributeValue>
     </Apply>
     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">…
       <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string" AttributeId="Department" />…
         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Payroll</AttributeValue>
     </Apply>
    </Apply>
       …
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
       …
    </Apply>
   </Condition>
 </Rule>…
 …

<Rule RuleId="FinalRule" Effect="Deny" />
```

Figure A.1: Sample Policy

References

[And05] XACML references. Technical Report v1.54, OASIS, May 2005.

[AZHG06] Raja Afandi, Jianqing Zhang, Munawar Hafiz, and Carl A. Gunter. AMPol: Adaptive Messaging Policy. In *European Conference on Web Services (ECOWS '06)*, Zurich, Switzerland, December 2006. IEEE.

[BFK99] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). In *Proceedings of the 6th International Workshop on Security Protocols*, pages 59–63, London, UK, 1999. Springer-Verlag.

[BFK+06] Rakesh Bobba, Omid Fatemieh, Fariba Khan, Carl Gunter, and Himanshu Khurana. Enabling attribute-based messaging using attribute-based access control. In *Annual Computer Security Applications Conference (ACSAC) '06*, Miami, FL, December 2006. ACSA. Accepted.

[BIK00] Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Trust management and network layer security protocols. In *Proceedings of the 7th International Workshop on Security Protocols*, pages 103–118, London, UK, 2000. Springer-Verlag.

[BIK02] Matt Blaze, John Ioannidis, and Angelos D. Keromytis. Trust management for IPsec. *ACM Transactions Information and System Security*, 5(2):95–118, 2002.

[BS00] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2000. ACM.

[BS02] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10(3):241–271, 2002.

[BSJ+05] Norbert Bieberstein, Rawn Shah, Keith Jones, Sanjay Bose, and Marc Fiammante. *Service-Oriented Architecture COMPASS: Business Value, Planning, and Enterprise Roadmap.* Pearson Education, 2005.

[CLZ04] David Chadwick, Graeme Lunt, and Gansen Zhao. Secure Role-based Messaging. In *CMS '04: Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security,Windermere, UK*, pages 263–275, 2004.

[DVS05] E. Damiani, S. Vimercati, and P. Samarati. New Paradigms for Access Control in Open Environments. In *5th IEEE International Symposium on Signal Processing and Information, Athens*, December 2005.

[FK92] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[FKMT05] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, New York, NY, USA, 2005. ACM Press.

[FKR03] D.F. Ferraiolo, D.R. Kuhn, and R.Chandramouli. *Role Based Access Control*. Artech House, 2003.

[GMMT05] Michael Matthew Greenberg, Casey Marks, Leo Alexander Meyerovich, and Michael Carl Tschantz. The soundness and completeness of margrave with respect to a subset of xacml. Technical Report CS-05-05, Department of Computer Science, Brown University, April 2005.

[God03] eXtensible Access Control Markup Language (XACML). Technical Report v1.1, OASIS, August 2003.

[KSB05] Himanshu Khurana, Adam Slagell, and Rafael Bonilla. Sels: a secure e-mail list service. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 306–313, New York, USA, 2005. ACM.

[LM03a] N. Li and J. Mitchell. Datalog with constraints: A foundation for trust management languages, 2003.

[LM03b] Ninghui Li and John Mitchell. Rt: A role-based trust-management framework, 2003.

[LMBG05] Kevin D. Lux, Michael J. May, Nayan L. Bhattad, and Carl A. Gunter. WSEmail: Secure internet messaging based on web services. In *International Conference on Web Services (ICWS '05)*, Orlando FL, July 2005. IEEE.

[LMW02] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *IEEE Symposium on Security and Privacy, Oakland*, May 2002.

[LPL+03] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using XACML for access control in distributed systems. In *XMLSEC '03: ACM workshop on XML security, Virginia*, pages 25–37. ACM, 2003.

[LYJ05] Yunyao Li, Huahai Yang, and H.V. Jagadish. Nalix: an interactive natural language interface for querying xml. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, Baltimore MD, June 2005.

[LYJ06] Yunyao Li, Huahai Yang, and H.V. Jagadish. Constructing a generic natural language interface for an xml database. In *International Conference on Extending Database Technology (EDBT 2006)*, Munich Germany, March 2006.

[MBH03] Marco Casassa Mont, Pete Bramhall, and Keith Harrison. A Flexible Role-based Secure Messaging Service: Exploiting IBE Technology for Privacy in Health Care. In *DEXA '03: 14th International Workshop on Database and Expert Systems Applications*, page 432. IEEE, 2003.

[MPI+03] S. Miltchev, V. Prevelakis, S. Ioannidis, J. Ioannidis, A. Keromytis, and J. Smith. Secure and flexible global file sharing, 2003.

[San98] R. S. Sandhu. Role-based access control. In M. Zerkowitz, editor, *Advances in Computers*, volume 48. Academic Press, 1998.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[WL02a] William H. Winsborough and Ninghui Li. Protecting sensitive attributes in automated trust negotiation. In *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 41–51, New York, NY, USA, 2002. ACM Press.

[WL02b] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *POLICY*, pages 92–103. IEEE Computer Society, 2002.

[WWJ04] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *FMSE '04: ACM workshop on Formal methods in security engineering, Washington DC*, pages 45–55. ACM, 2004.

[YGHS06] Noel Yuhanna, Mike Gilpin, Lindsey Hogan, and Andrew Sahalie. Information fabric: Enterprise data virtualization. White Paper, Forrester Research Inc., January 2006.

[YMW00] Ting Yu, Xiaosong Ma, and Marianne Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 210–219, New York, NY, USA, 2000. ACM.

[YT05] Eric Yuan and Jin Tong. Attributed Based Access Control (ABAC) for Web Services. In *ICWS'05: IEEE International Conference on Web Services, Orlando*, page 569. IEEE, July 2005.

[YWS01] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM Conference on Computer and Communications Security*, pages 146–155, 2001.

[YWS03] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.

[Zim95] Philip R. Zimmermann. *The official PGP user's guide*. MIT Press, Cambridge, MA, USA, 1995.

Author's Biography

Fariba Mahboobe Khan received her B.S. in Computer Science and Engineering from Bangladesh University of Engineering and Technology in 2004. After graduation she spent a short time as a lecturer in Ahsanullah University of Science and Technology. She is expecting her M.S. on Oct 2006 from University of Illinois at Urbana-Champaign (UIUC). Currently she is a PhD student in Computer Science at University of Illinois at Urbana-Champaign and working as a research assistant with Professor Carl Gunter in Illinois Security Lab. She also works closely and has been an intern with Dr. Himanshu Khurana in National Center for Super Computing Applications (NCSA). She has been awarded with Sarah and Shohaib Abbassi Fellowship on 2004-05 and 2005-06 academic year by the department of Computer Science. This year she had a publication in *Annual Computer Security Applications Conference (ACSAC)* and a peer-reviewed abstract presentation *1st Midwest Security Workshop (MSW)*.