Using Attribute-Based Access Control to Enable Attribute-Based Messaging [*]

Rakesh Bobba, Omid Fatemieh, Fariba Khan, Carl A. Gunter, and Himanshu Khurana
University of Illinois Urbana-Champaign[†]

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 paper 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 enterprises. Our implementation can dispatch ABM messages approved by XACML review for an enterprise of at least 60,000 users with only seconds of latency.

1. Introduction

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 [11, 5, 18, 17, 8].

An application that can benefit greatly from integration with an attribute-based system is multi-party e-mail 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 e-mail 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 e-mail 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 send to a large number of recipients just to reach the subset that actually needs to see the message. Though technically such unwanted messages do not qualify as *spam*[1] they tend to waste users' time all the same.

Attribute-Based Messaging (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 1) enhances the relevance of messages to the recipient and 2) allows the sender to send confidential messages knowing that the messages would be delivered only to the intended recipients. The approach also brings the intuitiveness of attributes as enterprise users typically understand the attributes associated with the users of their enterprise. Thus an ABM message with an address consisting of a query that returns the e-mail addresses of the faculty on sabbatical 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 Annual Computer Security Applications Conference (ACSAC '06), Miami Beach, FL, December 2006.

[†] Affiliations of the authors: Bobba, Khurana, National Center for Super Computing Applications (NCSA); Fatemieh, Khan, Gunter, Computer Science Department.

[1] *Spam* is defined as *unsolicited commercial e-mail* by the Federal Trade Commission.

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 e-mail 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 e-mail 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 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 three security and privacy challenges by employing an Attribute-Based Access Control (ABAC) approach integrated into an architecture focused on deployability. We then implement a prototype and conduct experiments that demonstrate the efficiency of our solution. This work is described in the following seven sections. In the second section we outline our approach for a practical access control system. In the third section we discuss how the ABAC approach is suitable for ABM. In the fourth section we describe our architecture for ABM using ABAC and eXtensible Access Control Markup Language (XACML) with off-the-shelf e-mail MUAs and MTAs. In the fifth section we describe our implementation and look at measures of its performance for various types of policies. In the sixth section we outline related work on targeted messaging and practical demonstrations of ABAC. In the seventh section we make conclusions, including limitations of our current work and possible future work.

2. 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' e-mail 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 servers 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). For example, Access Control Lists (ACLs) would not be a good policy model for ABMs since the creation and management of such lists for a potentially large number of attributes would be unwieldy. To address this we turn to ABAC, which has recently proven to be successful in access control for distributed systems [5, 8, 11, 17, 18]. In ABAC a requester is granted access to a collection of services based on a furnished collection of attributes. Translating this to 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) [9], 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.

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 [10] and Sun's standards-compliant implementation of its policy engine (`sunxacml.sourceforge.net`). There are several advantages to XACML for our practical demonstration. 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 [3] as well as research projects [14] indicates the confidence in its deployability and effectiveness.

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 e-mail address of the ABM server. Furthermore, the ABM address is integrity-protected and securely bound to the sender's e-mail account so that it cannot be spoofed or replayed. This approach also provides e-mail 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.

The efficiency of the access control system can best be gauged via prototype implementation and experimentation. With an eye towards rapid prototyping and performance we have employed several commercial of-the-shelf components 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 e-mail 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 principals 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. 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 [20]) that exemplifies the Service Oriented Architecture (SOA) approach [4] 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
$(value_i, \ldots, value_n)$,
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 [19]. 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 $\langle$*attribute, value* $\rangle$ pair $\langle$*position* = *faculty*$\rangle$ if the following expression of $\langle$*attribute, value*$\rangle$ pairs is satisfied: $\langle$*position* = *faculty*$\rangle$ *or* $\langle$*position* = *staff*$\rangle$ *and* $\langle$*designation* = *coordinator*$\rangle$.

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 $\langle$*attribute,value*$\rangle$ 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 e-mails to achieve the same effect and when the sender chooses the *and* operand she only ends up targeting his e-mail to a narrower set of recipients than she is allowed to. Therefore, at most one access policy is required for each $\langle$*attribute,value*$\rangle$ pair. In practice, there are various ways to reduce the number of policies, some of which are explored in Section 6.

4. Architecture

Figure 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.

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 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 $\langle$*attribute, value*$\rangle$ 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 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).

Messaging (MS) Path. This path is represented by solid lines in Figure 1. Users send ABM messages using any

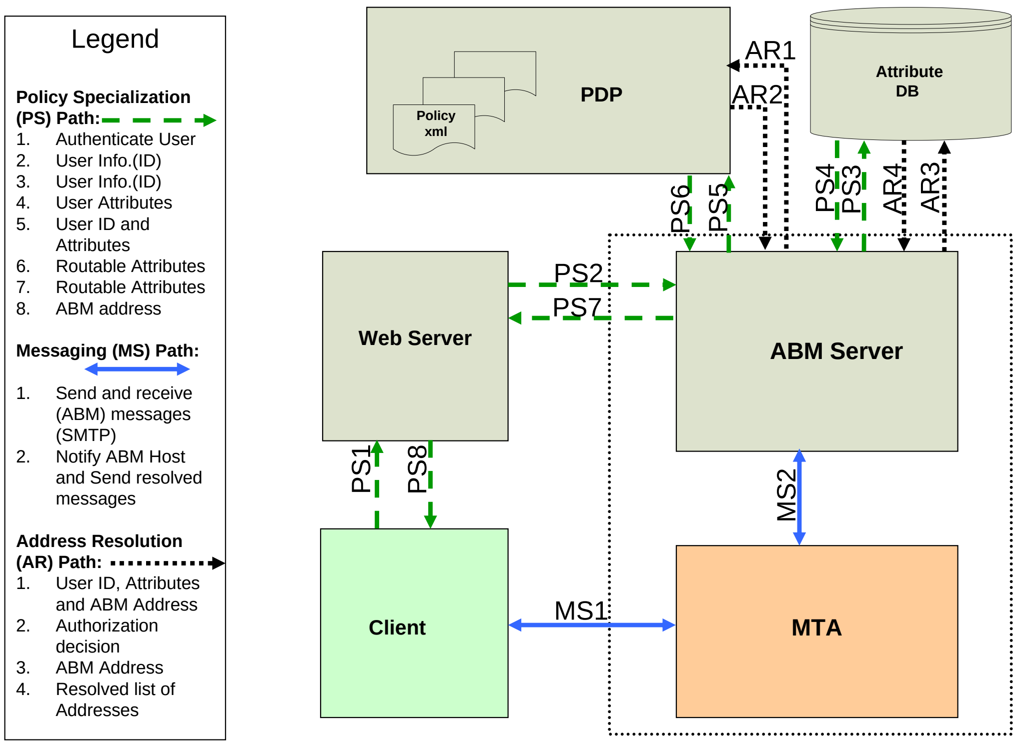

Figure 1. ABM Architecture

standard MUA [2] 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.

Address Resolution (AR) Path. This path refers to the message processing by the ABM server and is represented by dotted lines in Figure 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 *e-mail 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 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.

[2]ABM system can easily be integrated with web-based e-mail in enterpsises that use web-based e-mail system but for generality we assume the presence of an e-mail client like Outlook.

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 *e-mail 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 *e-mail 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.

5. Implementation and Experimental Results

To test that the architectural framework presented in Section 4 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.

5.1. 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.

PDP. As it was described Section 2, 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.

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.

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.

5.2. Test Bed

Studying the components in our system in Figure 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 running on a 2.8 GHz Pentium 4 with 1GB of memory with Windows XP Pro operating system.

5.3. 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 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 e-mail.

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.

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 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.

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 0.0001 to 0.01, 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.

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 (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.

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.

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 1. Address Resolution Time. Number of attributes = 100; number of policies = 568.

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) recomposing the message with the list of recipients. For the case without access control only the third and fourth latency components were included.

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 1 summarizes our results. The Average List Size field in the table refers 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.

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

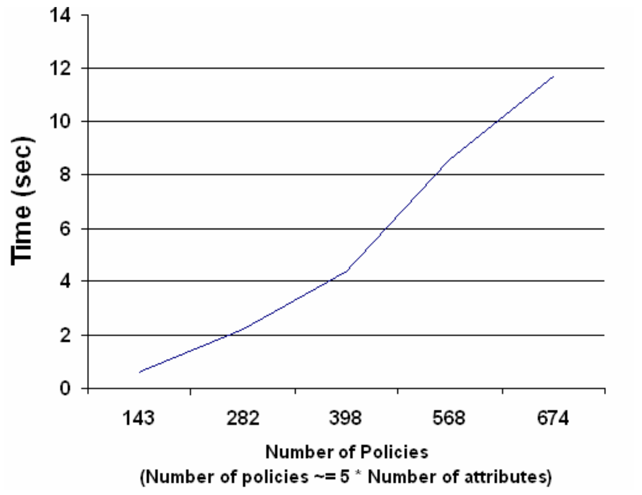

Figure 2. Policy Specialization Time

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 2. 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.

5.4. Analysis of Results

Feasibility Without Access Control. As shown in Table 1, 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.

Feasibility With Access Control. As shown in Table 1, the average latency added to an e-mail message by the ABM system (address resolution latency) with access control is under $350ms$ when using a relational database and under seven seconds when using an XML database. Adding security to the system added at most $100ms$ additional latency when using a relational database and $450ms$ latency when using an XML database. Thus, on average the ABM system with security can process 190 requests per minute using a relational database and 8.5 requests per minute using an XML database. The discrepancy in latency added by security when using a relational database vs. an XML databases 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, via the PDP, by itself takes around $60ms$ and gives us a throughput of 1000 validations per minute.

As expected, Figure 2 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.

6. Discussion

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

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 $\langle$*attribute, value*$\rangle$ 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.

⟨*request.teaching* = $variable\_x$⟩
$\Rightarrow \langle course, variable\_x \rangle$,
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 subranges 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 ⟨*attribute, value*⟩ 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.

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 [12, 13], that enable translation of queries in English into queries in XQuery can further improve the usability of ABM system.

Privacy Considerations. Another issue that needs attention when deploying a system like ABM is privacy of sender and recipient e-mail addresses and of the ABM address itself. For instance, should the senders be allowed to know the list of recipients of the message sent to a particular ABM address? Are receivers entitled to know why they received a particular message or the ABM address that was used to target the message? When the attributes used to target a message are sensitive allowing senders to know the list of recipients would compromise the privacy of the recipients. Similarly letting the recipients of a message know the ABM address used to target the message might leak sensitive information if they could learn who else received the message.

If a sensitive attribute, for example *medical condition*, is used in an ABM address to target messages then 1) the ABM address using the sensitive attribute, 2) the list of recipients (e-mail addresses) targeted by the ABM address and 3) the sender's e-mail address should all be considered sensitive and there should be policies governing the release of such information. For instance, senders may be allowed to know only those recipients that are not targeted by the sensitive attribute. Recipients may be allowed to know only their attributes that were included in the ABM address rather than the (entire) ABM address. If a sender targeting messages based on sensitive attributes is not allowed to know the recipient list, it might be desirable to reciprocally not let the recipients know who the sender is.

7. Related Work

We discuss four areas of related work: targeted messaging systems, secure role-based messaging, WSEmail, and attribute based access control.

Perhaps the most similar technology to ABM arises in Customer Relationship Management (CRM) systems. CRMs help enterprises 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 e-mail 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 integerated 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.

Secure role-based messaging uses RBAC for authorizing access to sensitive e-mail content [7, 16]. In this area [7] allows users to send messages to a given role identified by a special e-mail address. Users that are assigned to that role can then provide their role membership credentials and access the e-mail. Using a slightly different approach [16] employs Identity Based Encryption (IBE) for encrypting messages to recipients; *i.e.,* recipient must authenticate themselves to a role administration system and obtain the e-mail decryption keys. These two approaches differ from ABM access control as described in this paper 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.

The Adaptive Messaging Policy (AMPol) project, of which this paper is a part, has considered some technologies related to ABM [15, 2, 1]. WSEmail is the idea of building messaging systems over a web services foundation. A prototype [15] 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 [2] shows how to adapt to recipient policies as part of messaging, but this design does not deal with multiple recipients. Other details on AMPol, can be found on the AMPol web site (`seclab.cs.uiuc.edu/ampol`).

Early works on ABAC [5, 17, 19, 18] use it for trust negotiation and credential based access control in a distributed system with multiple administrative domains. Our ABM study shows how ABAC is also valuable for enterprise applications and uses attributes assimilated from backend databases. Also, access control in ABM is different from access control in traditional systems and services because the resource (*i.e., an ABM address*) is somewhat different than a resource in these traditional systems. Most of the research on ABAC provides insights on theory and expressiveness for applications but do not discuss implementation of the proposed designs and practical studies on applications. Some works [14, 19, 18] have led to implementations, but no performance data is available. At the same time performance of access control systems in becoming important in recent application such as location based access control [6]. In this work we demonstrate the practicality of ABAC for a novel enterprise application (ABM) in a mid-size enterprise as evidenced by our performance evaluation.

8. 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: interdomain 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 [11, 17] 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.

Acknowledgements

We would like to thank Noam Artz, Mike Berry, and anonymous reviewers for their helpful comments. This material is based upon work supported by the ONR N00014-04-1-0562 and N00014-02-1-0715. This work also benefited from partial support by NSF CCR02-08996, CNS05-09268, and CNS05-24695, a grant from MacArthur Foundation and the Sohaib and Sara Abbasi Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of ONR, NSF or MacArthur Foundation.

References

[1] R. N. Afandi, J. Zhang, and C. A. Gunter. AMPol-Q: Adaptive Middleware Policy to Support QoS. In *International Conference on Service Oriented Computing(ICSOC)*, Chicago, IL, December 2006.

[2] R. N. Afandi, J. Zhang, M. Hafiz, and C. A. Gunter. AMPol: Adaptive Messaging Policy. In *European Conference on Web Services(ECOWS '06)*, Zurich, Switzerland, December 2006. IEEE.

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

[4] N. Bieberstein, R. Shah, K. Jones, S. Bose, and M. Fiammante. *Service-Oriented Architecture COMPASS: Business Value, Planning, and Enterprise Roadmap*. Pearson Education, 2005.

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

[6] K. Borders, X. Zhao, and A. Prakash. CPOL: high-performance policy evaluation. In *CCS '05: 12th ACM Conference on Computer and Communications Security, Virginia*, pages 147–157. ACM Press, 2005.

[7] D. Chadwick, G. Lunt, and G. 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.

[8] E. Damiani, S. D. C. di 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.

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

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

[11] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *IEEE Symposium on Security and Privacy, Oakland*, May 2002.

[12] Y. Li, H. Yang, and H. 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.

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

[14] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. 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.

[15] K. D. Lux, M. J. May, N. L. Bhattad, and C. A. Gunter. WSEmail: Secure internet messaging based on web services. In *International Conference on Web Services (ICWS '05)*, Orlando FL, July 2005. IEEE.

[16] M. C. Mont, P. Bramhall, and K. 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.

[17] L. Wang, D. Wijesekera, and S. 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.

[18] T. Yu, M. Winslett, and K. 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.

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

[20] N. Yuhanna, M. Gilpin, L. Hogan, and A. Sahalie. Information fabric: Enterprise data virtualization. White Paper, Forrester Research Inc., January 2006.