

Dynamic, Context-Aware Access Control for Distributed Healthcare Applications

Junzhe Hu and Alfred C. Weaver
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

The rapid worldwide deployment of the Internet and Web is the enabler of a new generation of e-healthcare applications, but the provision of a security architecture that can ensure the privacy and security of sensitive healthcare data is still an open question. Current solutions to this problem (mostly built on static RBAC models) are application-dependent and do not address the intricate security requirements of healthcare applications. The healthcare industry requires flexible, on-demand authentication, extensible context-aware access control, and dynamic authorization enforcement. With on-demand authentication, users are authenticated according to their task-specific situations. Extensible context-aware access control enables administrators to specify more precise and fine-grain authorization policies for any application. Dynamic authorization enforcement makes authorization decisions based upon runtime parameters rather than simply the role of the user. In this paper we describe a dynamic, context-aware security infrastructure that can fulfill the security requirements of healthcare applications and that can also be easily adapted to offer security support for similar enterprise applications.

1. Introduction

Many industries are extending their business operations to operate over the Web, and healthcare is no exception. Although most healthcare participants (e.g., hospitals, private physicians, insurance companies, pharmacies) already implement some type of computerized system to manage their business data, their implementations tend to be proprietary and interact loosely or not at all. A more integrated system that crosses the various healthcare boundaries would clearly be an advantage to the patient; imagine the economy of effort for the patient if he or she could schedule an appointment, retrieve the results of a diagnostic test, refill a prescription, and file an insurance claim from a single, integrated portal, rather than making four separate logins on four separate websites to conduct four separate transactions with four different entities. A web services approach has the potential to enable this single-portal scenario, but only if we can assure the privacy and security of patient data throughout the medical enterprise. To do that, we must first understand the security requirements of modern healthcare.

First, given the new privacy and security requirements of the recent Health Insurance Portability and Accountability

Act [1], healthcare services are turning to stronger authentication methods. Biometric methods (e.g., fingerprints, iris scans, signature and voice recognition) and non-biometric digital techniques (e.g., e-tokens, RFID, key fobs) are rapidly replacing passwords for authentication purposes. A practical and usable healthcare system must accommodate a range of identification technologies and their associated trust levels; see section 3 for details.

Second, even within a single medical enterprise there can be an enormous variety of applications that require making complex access control decisions. For example, in our medical center, a medical student can not view a patient's data unless that access is authorized by the patient's attending physician. Although RBAC (role-based access control) can make authorization decision based on a user's role, it is not sufficiently powerful to enforce rules that are dependent on runtime parameters. Current healthcare applications require a context-aware control infrastructure to enforce these policies.

Third, the security requirements in our healthcare system require very dynamic and flexible policy enforcement. Even in the normal case, there are many intricate user-data relationships that must be managed by the security framework; in addition, we must also accommodate emergency access to data in life-threatening situations (e.g., public health emergency). In our proposed security infrastructure, the policy enforcement mechanism is highly dynamic and independent from any particular application.

2. Related Work

Role-based access control is presented in Sandhu et al.'s seminal paper [2] where the main RBAC components (users, roles, permissions, sessions) are systematically addressed. Using this model, efficiency is gained by associating permissions with roles rather than users. This model greatly simplifies security management for administrators, and many complex security policies can be applied more easily. Bertino et al. [3] presents Temporal-RBAC, which supports periodic role enabling and disabling and temporal dependencies among permissions by introducing time into the access control infrastructure. Covington et al. [4] extends the model beyond time by introducing location and system status as constraints. Moyer and Abamad propose generalized RBAC in [5]; GRBAC leverages and extends the power of traditional

RBAC by incorporating subject roles, object roles, and environment roles into access control decisions.

Georgiadis and Mavridis [6] and Wang [7] both present a team-based access control model that is aware of contextual information associated with activities in applications. Kumar et al. [8] summarizes previous work and formally proposes a context-sensitive RBAC model. Their model enables traditional RBAC to enforce more complicated security policies that are dependent on the context of an attempted operation; however, this model does not provide a mechanism for automatically merging new context types into existing access policies, which limits its application in distributed scenarios. McDaniel [9] gives a more generic view of contexts, suggesting that a context should be defined by its implementation. Neumann and Strembeck [10] and Lei et al. [11] also discuss some research issues in context-related security applications.

Meanwhile, approaches to applying the RBAC model to applications distributed over the Internet have also been proposed. Taylor and Murty [12] and Joshi et al. [13] describe a security model for authentication and access in distributed systems, while Kang et al. [14] addresses how to separate inter-organizational workflow from concrete organizational-level security enforcement; however, none of these give much consideration to integrating context data with their models. Bonatti and Samarati [15] propose an approach for regulating service access and information disclosure on the Web, which can be adopted in our system when equivalent organizations all have independent access control infrastructures.

There are also some papers on security issues in distributed healthcare systems. Weaver et al. [16,17] propose a federated, secure trust network for distributed healthcare systems, which is the motivation for this paper. Zhang et al. [18] presents a delegation framework that can be used within the security framework of healthcare applications. Wilikens et al. [19] discusses how to apply CBAC to healthcare, but the model is static and not able to handle arbitrary context-dependent authorization policies.

Our authorization model is dynamic. Each data access attempt is verified by our authorization engine, and its rules can be changed on-the-fly by systems administrators in response to changing conditions (e.g., a public health emergency).

3. Dynamic, Context-Aware Security Framework

3.1 Authentication Trust Levels

Authentication is the first step to protect sensitive medical records. We are investigating a number of biometric technologies (e.g., fingerprint, iris scans, signature and voice recognition) as well as other digital techniques (e.g., e-tokens, RFID, key fobs) to determine their reliability. Each authentication technology is assigned a trust level $T()$

based upon its perceived reliability. In the abstract, these trust levels are ordered based upon the number of degrees of freedom inherent to the underlying identification technology.

For example, there is general agreement [20] that $T(\text{retina}) > T(\text{iris}) > T(\text{fingerprint}) > T(\text{password})$. However, in practice, the trust level of any particular product must be determined by experimentation to quantify its false acceptance and false rejection rates (the false acceptance rate is the percentage of authentication attempts by a person other than the enrolled individual which are nevertheless successful; the false rejection rate is the percentage of authentication attempts by the enrolled individual which are nevertheless rejected). In our system, hospital administration defines the trust level of each identification technology based upon its general perception of reliability, its manufacturer's data on false acceptance and false rejection rates, and our own experimental evidence from testing the devices in our lab. The numerical values assigned to the trust levels $T()$ are arbitrary; it is only their relative ordering that is important because administrators will issue a rule such as "access requires authentication at the level of $T(\text{fingerprint})$ or higher."

Our system implements Schneider's concept of "least privilege" [26]—that is, granting the user no more rights than are needed to execute the service requested and for which the user is authorized. This means that a user whose verified identity would normally allow him or her to write data, but who is only asking to read data, is only assigned a read permission. A user may log on using any identification technology physically available on his/her system, and an authentication token is issued that contains a trust level consistent with the identification technology used. If and when the user makes a request that requires a higher trust level, the user is offered the opportunity to re-authenticate using any available method of higher reliability to gain a new trust credential with a higher trust level.

3.2 Context-Aware Access Control

Healthcare systems have complex access rules because of the many actors in the system and their interlocking access privileges, and most of these rules have to be context-aware. A practical healthcare IT system must support hundreds or thousands of users, roles, objects, and permissions.

In this section, we introduce the formal definition of our context-aware access control schema. Let us first define our terminology.

Data Object: A data object is the smallest unit to be accessed and protected in an application. Examples include a prescription, a clinical note, a laboratory test result, or a diagnostic image.

Data Type: A data type is a group of data objects with the same attributes. For instance, all patients' prescriptions in a hospital have a data type of "Prescription". More generally, data types can be combined to define a more complex type. Thus "Electronic Patient Record" is a complex type composed of subsidiary types such as demographics, clinical notes, psychological evaluation, diagnostic tests results, prescriptions, etc.

Data Set: A data set is the set of all data objects within an application.

User Set: A user set is the set of potential entities that will access the data objects in the data set of an application.

Definition 1 (Context Type): A context type is defined as a property related to every participant in an application when it is running. In simple cases, context type may be a concrete property familiar in everyday life, such as time or location. In a more complex scenario, context type can also be used to describe an abstract concept such as the authentication trust level that we discussed previously.

Based on context type, every application has its own **Context Set**, which is defined as a set of context types:

$$CS = \{CT_1, CT_2 \dots CT_n\}, 1 \leq i \leq n.$$

By analyzing the system security requirements, application designers determine which context types will be used to specify access. For example, suppose we have a security requirement that a physician can access the data of patients if and only if that physician has the legal status of "attending physician" for that patient. From this requirement, we need a concept to describe if there is an "attending physician" relation between physician A and patient B, so we define a context type "Attending Relation". Although the context set is determined before the application implementation, system administrators can dynamically add new ones when needed.

In order to make this abstract concept of context type usable when authorization decisions need to be made, it is necessary to have each context type evaluated by some **Context Implementation**. A context implementation CI of context type CT is defined as a function with n inputs (context type) that returns an object of type CT.

$$CI: CT_1 \times CT_2 \times \dots \times CT_n \rightarrow CT, n \geq 0.$$

Context types without parameters are called primitive types and we support five types; those with parameters require the user to supply a function or web service that can evaluate them. This concept is elaborated in the next section.

Definition 2 (Context Constraint): We define a context constraint as a regular expression (shown below). Based on this format, our access control schema is capable of specifying any complex context related constraint to describe all kinds of security requirements.

Context Constraint: $= \text{Clause}_1 \cup \text{Clause}_2 \dots \cup \text{Clause}_i$

Clause: $= \text{Condition}_1 \cap \text{Condition}_2 \dots \cap \text{Condition}_j$

Condition: $= \langle CT \rangle \langle OP \rangle \langle VALUE \rangle$, where

$CT \in CS$; OP is a logical operator in the set $\{>, \geq, <, \leq, \neq, =\}$ and this set can be extended to accommodate user-defined operators as well; VALUE is a specific value of CT.

Suppose we have a context set $CS = \{\text{Time, Location, Authentication Level}\}$, and we have a partial security rule such as "patient data can be accessed from within the hospital between 8am and 5pm with a trust level of a password; otherwise a higher level of trust is required." If "in" is a user-defined logical operator and "hospital" is a valid value of context type Location, then this rule could be expressed as:

$$\text{Context Constraint: } = (\text{Time} \geq 08:00 \cap \text{Time} < 17:00 \cap \text{Location in hospital} \cap \text{AuthenticationLevel} \geq T(\text{password})) \cup (\text{AuthenticationLevel} > T(\text{password}))$$

Definition 3 (Authorization Policy): We define an authorization policy as a triple, $AP = (S, P, C)$ where:

S: is the subject in this policy, which could be a user or a role.

P: is the target permission in this policy, which is defined as a pair $\langle M, O \rangle$, where M is an operation mode defined in $\{\text{READ, APPEND, DELETE, UPDATE}\}$ and O is a data object or data type.

C: is a context constraint in this policy. If C is empty then this policy reverts to simple RBAC.

Definition 4 (Data Access): We define data access as a triple, $DA = (U, P, RC)$ where:

U: is a user in a User Set who issues this data access.

P: is the permission this user wants to acquire.

RC (runtime context): is a set of values for every context type in the Context Set. That is, $RC = \{v_1 \text{ of } CT_1, v_2 \text{ of } CT_2, \dots, v_n \text{ of } CT_n\}$, where $\{CT_1, CT_2, \dots, CT_n\}$ is the context set CS of the application.

A data access DA (U, P, RC) is granted only if there exists an authorization policy AP (S, P', C), such that $U \in S$, $P = P'$, and C evaluates to true under RC (that is, when all CTs in constraint C are replaced with their values in RC, then the resulted Boolean expression is true).

3.3 Dynamic Context Evaluation

From the above context-aware access control schema, we can design the basic algorithm to determine whether a data access is authorized or not based upon the context type and its implementation.

Algorithm: RequestPermission (DataAccess da)

```

initialize candidate policy set PS = {}
for every AP in policy set of the application
    if (U in da ∈ S in AP) and (P in da = P in AP)
        put AP into PS
    end if
end for
result = "Reject"
for every AP in PS

```

```

if (EvaluateContexts(C in AP) is true)
    result = "Accept"
    break
else
    result = "Reject"
end if
end for
return result

```

Algorithm: EvaluateContexts(Constraint C)

```

for every clause CL in C
    for every condition CN in CL
        get value of context type CT in CN
        compute CN with this retrieved value
        if (CN = false)
            CL = false
            break
        end if
    end for
    if (CL = true) return true
    else continue
end for
return false

```

It is a crucial step in the two algorithms above to dynamically retrieve the context type value (also called context evaluation). By our definition, a context implementation is actually a function or method call. There are several ways to implement context types such that they can be evaluated dynamically. For example, the application developer can implement a context type in a dynamic link library or as a web service, both of which support dynamic invocation at runtime. In our system, we implement all the required context types as web services to enhance extensibility and interoperability.

When a context implementation is dynamically invoked to retrieve the value of the corresponding context type, we must solve the problem of how to pass parameters to that function (context implementation). Since a web service is an XML-based protocol, we know what kind of parameters this web service needs from its WSDL [21]. In addition, according to our context implementation definition, those required parameters are also context types, so we can get those parameters' values by evaluating their respective context types. By iteration, this eventually reduces to a context type without parameters.

We call a context type CT a *primitive* context type CT_p , if its context implementation

$$CI: CT_1 \times CT_2 \times \dots \times CT_n \rightarrow CT, n = 0.$$

In our system, there are 5 primitive context types:

- Time— when this access request was issued;
- Location—where the access request was issued (e.g., IP address, physical location, category of location such as “mobile” or “inside-hospital”);
- User ID—who sent the request;

- Object Type—what type of data object is being accessed;
- Object ID—which particular data object is being accessed.

Therefore, based on this context implementation hierarchy, any context type CT_n can be dynamically evaluated. Figure 1 shows the dynamic execution path needed to resolve an authorization rule such as: “the requester named by the UserID is granted access if the access time-of-day satisfies the rule implemented by the Time() context implementation (a primitive type), if the UserID authentication token has been validated to have an adequate trust level as evaluated by the TrustLevel() context implementation, and if the UserID (requester) has an “attending physician” relationship with the patient (identified by ObjID) as determined by the Attending() context implementation.

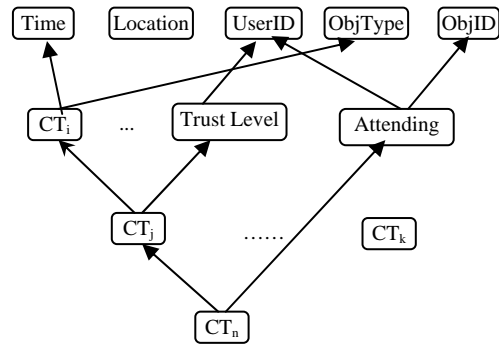


Figure 1. Context Implementation Hierarchy

3.4 Access Policy Specification

In our context-aware access control schema, an access policy specifies which role has what kind of permissions under some contextual constraints. Although access policies are mainly used by the authorization engine to make access control decisions, these policies also need to be exchanged among trust domains. For example, if a pharmacy receives an electronic prescription for a restricted drug (e.g., narcotics), the pharmacy needs to verify that the physician has the appropriate DEA license to write such a prescription. If the DEA authorization number is not provided with the electronic prescription, then the pharmacy will contact the hospital electronically and request the missing information; if the hospital supplies a validated response the prescription is accepted, otherwise it is rejected.

In order to make our access policies understandable to other applications within the same trust domain, specifying the policies in XML format is therefore the first choice for us, because XML is becoming a standard representation for interoperation rules between different applications. Also, there are some XML-based policy languages available for us to use, such as WS-Policy [22], XACML

[23] and SAML [24]. All these policies languages define an XML framework to exchange authentication and authorization information for accessing a web service or web resource, but they are based only on the traditional RBAC model. We must extend that to incorporate our context-aware access control schema.

Since our system is built on Microsoft's .NET Framework, and WS-Policy inherently supports a challenge-response policy exchange, we use WS-Policy as our access control policy specification language and customize it to specify contextual constraints in policies. The customized element tags we use are shown below.

For any authorization policy $AP = (S, P, C)$

- `<wsa:DataType>` specifies the data object or data type of permission P in an AP;
- `<wsa:AccessType>` specifies the operation mode of permission P in an AP;
- `<wsa:Permission>` specifies permission P in an AP;
- `<wsse:SubjectToken>` specifies the security token issued to S in an AP from the authentication engine;
- `<wsse:ContextToken>` specifies one context condition in C of an AP;
- `<wsse:ContextType>` specifies which context type is used in one context condition in C of an AP;

As an illustration, consider this example of the access rule that "the attending physician can only delete a patient's record when he logs in with an authentication trust level higher than password." The corresponding authorization policy will look like this.

```

<wsp:Policy>
  <wsp:AppliesTo>
    <wsa:EndpointReference>
      <wsa:DataType>PatientRecord
    </wsa:DataType>
    <wsa:AccessType>Delete
    </wsa:AccessType>
    <wsa:Permission>DeletePatientRecord
    </wsa:Permission>
    </wsa:EndpointReference>
  </wsp:AppliesTo>
  <wsse:SubjectToken wsp:Usage="Required">
    <wsse:TokenType>Attending Physician
    </wsse:TokenType>
  </wsse:SubjectToken>
  <wsp:OneOrMore wsp:Usage="Required">
    <wsp:All>
      <wsse:ContextToken
        wsp:Usage="wsp:GreaterThan"
        wsp:Preference="T(password)">
        <wsse:ContextType>Trust Level
        </wsse:ContextType>
      </wsse:ContextToken>
    </wsp:All>
  </wsp:OneOrMore>
</wsp:Policy>

```

Permission Block (bracketed around the first three blocks)
 Subject (bracketed around the second block)
 Constraint (bracketed around the third block)

4. Implementation of the Security Infrastructure

Figure 2 depicts the essential system architecture of our proposed distributed healthcare IT system. There are three main components: authentication engine, authorization engine, and context service. The authentication engine is responsible for issuing authentication tokens to users and establishing the authentication level (trust level) for that token. The authorization engine monitors all data access requests coming from the web service interface. If the issuer of the data request has permission to access that data (as determined by applying the authorization rules to the issuer's identity and any context-dependent information), then access is granted and results are returned to the issuer. The context service, using our extensions to the RBAC model, records all pre-defined context types, evaluates them dynamically, and reports its results to the authorization engine.

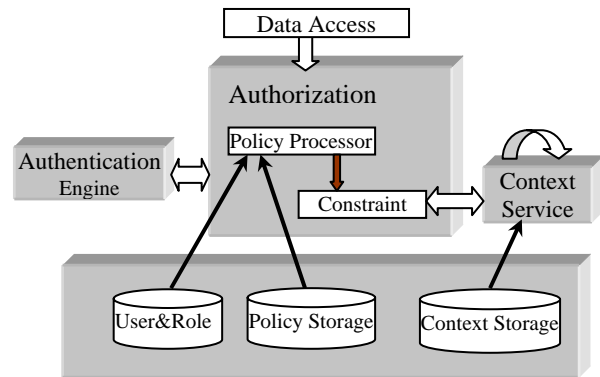


Figure 2. Security Infrastructure

4.1 Authentication Engine

The authentication engine (also called the Secure Token Service) is interrogated whenever a user needs to establish identity. Its main function is to validate credential information supplied by the user against the database of enrolled identities. In the simplest case, the user supplies a username and a password, the authentication engine verifies correctness, and if correct issues an authentication token containing the user's name and an indication of how identity was established (e.g., via password). The engine also records the trust level $T()$ associated with the identification technology used (in this case, the trust level is $T(\text{password})$). The engine then digitally signs the authentication token. In a more complicated scenario, the user may present a username and a biometric credential (fingerprint, iris, signature, voice); the biometric data template is validated against the username's stored enrollment data and an authorization token is issued if the data matches. Each authentication token is valid for a limited period of time; the validity period varies with the authentication methodology and its length is determined by hospital administrators.

4.2 Authorization Engine

The authorization engine allows system administrators to specify context-aware authorization policies at policy design time, and then guarantees their enforcement at run-time. Alternatively, since all contexts are evaluated dynamically, administrators can add a new authorization rule or modify an existing one at any time. According to our schema, hospital administration specifies an access control policy by describing its three elements: subject, permission, and constraint. Subjects are easily defined by the user's position (role or user account) within the organization. Individuals may have multiple roles—Dr. Smith may be a physician and a radiologist and a mammographer.

Historically, permissions are pre-defined and associated with roles at policy design or code initialization time. As previously indicated, this is too restrictive for our healthcare environment—roles change, permissions change, and context data change on the basis of real-time circumstances. Thus our authentication engine handles permissions in a very modular way. In our system, a software module processes each data type, and there are five default operations on each module: admin (create or modify operations available on a module), read, update, create, and delete. For example, to handle access to our diagnosis data we design a module that can implement those default operations on that data and likewise dynamically define the default permissions. If more complicated transactions (i.e., more than simple combinations of the five default operations) are needed, we are free to define new permissions. By describing permissions as operations on modules, our engine is capable of managing arbitrary authorization policies.

4.3 Context Service

The context service manages a repository for all context definitions and their corresponding implementations in our security architecture. When the authorization engine encounters a context type in a context condition, it calls the context service to evaluate that context type. Context implementation is provided as a web service, so context evaluation is the process of invoking those web services dynamically. When a complex context type is evaluated, the context service will iteratively ask itself for evaluation inputs.

5. Prototype of a Distributed Healthcare System

We discuss below the application of our prototype dynamic access control infrastructure in a practical healthcare IT system.

5.1 Application Overview

The Ceberus web portal, shown in figure 3(a), is the main access point for all users such as patients, physicians, staff, etc. The portal provides information for the (initially anonymous) user, but also provides dynamic function tabs

on the web page for users to explore the site based upon their particular role. For example, if a user, authenticated as a patient, accesses the site then the web portal produces tabs for upcoming healthcare events such as future appointments, billing information, prescription status, etc. When a physician logs in, all his/her patients' names are displayed; selecting a particular one displays the patient's diagnosis, lab results, diagnostic imagery, and related information (see figure 3(b)), and this data is invisible to others who were not able to authenticate their role as a physician involved in this patient's care. Our system exposes different views of the healthcare IT database depending upon the (authenticated) role of the viewer. This is the first security layer and is built upon the traditional RBAC model.



Figure 3(a). Ceberus Medical Data Portal Homepage

Order	Quantity	Description	Lab Date
1	100	Redtop	1/20/00 2:30:00 PM
2	10	Redtop	1/20/00 2:45:00 PM
3	10	Redtop	1/20/00 4:12:00 PM

Area	Volume	Position	Thickness	View	Contrast	Collim	Magnification	
125	140	3.5	20	20x	20x	1.75	5.2	2.2
80	140	3.5	20	20x	20x	1.5x		
125	80	4.0	20	20x			2.5	3.1

Figure 3(b). Ceberus Display of a Patient's Lab Data and Diagnostic Imagery

Behind this first security layer lies the core access control module of our system. Only system administrators can be authorized to see these security-related pages that define user-role assignments, access rules, and context definitions. This module implements our dynamic distributed access control infrastructure and all system access control policies are produced here. Administrators define context types first, then specify the conditions associated with particular permissions based upon these context definitions. After the user passes the first RBAC-based security layer, the subsequent data access operations and transactions are all

subject to the scrutiny of this context-aware security layer before they are processed by the backend database. Figure 4 illustrates this system.

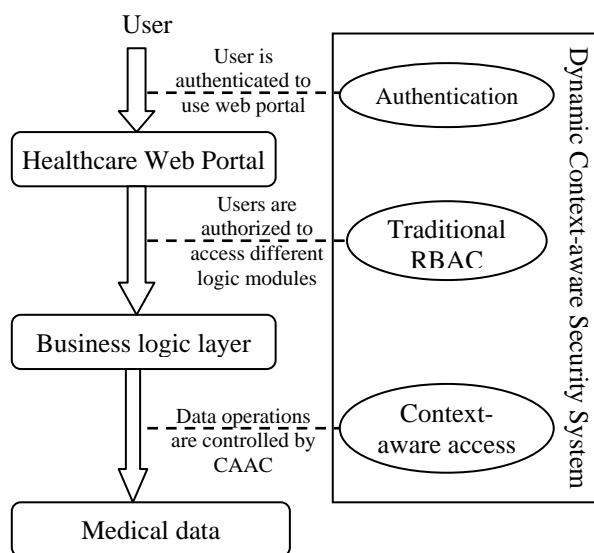


Figure 4. Access Control Layers

5.2 Applications of Context-Aware Access Control

This architecture supports the access control requirements that we need for healthcare. One such requirement is that a patient can view his/her own medical data, but no other person's. The administrator can define a context type "Owner of Medical Data" and a developer implements it as a web service called by "IsOwner(UserID, ObjectID)" which evaluates the target object to determine if it is owned by the target user and returns the resulting Boolean. Both UserID and ObjectID are embedded context types.

Another requirement is that a technologist can only insert and view diagnostic images related to the technologist's specialty (e.g., cardiology, urology). Here the administrator defines a context type "ImageSpecialty" and the developer designs a web service "IsSpecialty(UserID, ObjectID)" that returns a Boolean indicating whether the user's specialty matches the image's specialty.

A more complex example is illustrated by the requirement that to update a patient's record, a physician must be that particular patient's attending physician and that physician must have been authenticated at a trust level of a fingerprint or higher. Here the UserID is that of the access requestor, hence the UserID identifies the physician. This logic could be stated as "IsAttending(UserID, PatientID) & AuthenticationLevel(UserID) >= T(fingerprint)".

5.3 Other Security Concerns

Although the main focus of our security infrastructure is access control, we must also address security issues raised by our distributed architecture. Most context types are implemented as web services located within the enterprise, but in some cases third parties outside our control will

implement contexts. To guarantee that only authorized sites can access those services, we use Web Service Enhancements (WSE) [25] to protect them. WSE is a set of tools that allows developers to build web services that take advantage of advanced web service specifications (e.g., WS-Security [20]). Using WSE, every access to a web service can be signed with a server certificate that is trusted by all web services. The context implementation web services can therefore identify and reject any unsigned (and thus unauthorized) access.

Because our system is fully distributed, all communication is purely text-based using XML. In our current system all transmissions are secured using Secure Sockets Layer (SSL) that provides encryption for all messages; future versions will utilize WS-SecureConversation.

6. Conclusions

In this paper we described a dynamic context-aware access control infrastructure that extends the traditional RBAC model to gain many advantages from its context-aware capability. Our research motivation comes from the complicated access control requirements inherent to current healthcare data management. Traditional RBAC is not able to specify a sufficiently fine-grained authorization policy or specify constraints that should be applied to an access policy. Earlier extensions of RBAC to create CBAC improved access control, but that model is static with poor flexibility and extensibility. Our new access control infrastructure is dynamic and distributed with these advantages:

- Our access control model extends traditional RBAC by associating access permissions with context-related constraints. Every constraint is evaluated dynamically against the current context of the access request. Therefore, our new model is capable of making authorization decisions based upon context information in addition to roles.
- Our context-aware access control is applied dynamically. At design time, administrators have great flexibility to specify complex context-aware authorization policies. At run-time, our authorization engine can enforce any context-aware policy automatically because it is not statically bound to any application.
- Context implementation is separated from the main business logic of target applications. Since every context type definition and context implementation is independent of the specification of the access rules, any change to them has no effect on other parts of the system. Thus our security infrastructure is flexible and permits easy extensibility.

Acknowledgements

We gratefully acknowledge the multiple discussions and intellectual contributions of the other members of our

research group (Xiaohui Chen, James Van Dyke, Andrew Marshall, Vincent Noël) and the support of our research sponsor, David Ladd, at Microsoft Research.

References

- [1] Health Care Portability and Accountability Act, Public Law 104-191, <http://aspe.hhs.gov/admsimp/pl104191.htm>
- [2] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman, "Role based Access Control Models," *IEEE Computer*, volume 2, February 1996, pp. 38-47.
- [3] Elisa Bertino, Piero Andrea Bonatti and Elena Ferrari, "TRBAC: A Temporal Role-Based Access Control Model," *ACM Transactions on Information and System Security*, Volume 4, No. 3, August 2001, pp. 191-223.
- [4] Michael J. Covington, Wende Long and Srividhya Srinivasan, "Secure Context-Aware Applications Using Environment Roles," *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, May 2001, Chantilly, Virginia, USA.
- [5] M. J. Moyer and M. Abamad, "Generalized Role-Based Access Control," *21st International Conference on Distributed Computing Systems*, April 16-19, 2001, Atlanta, GA, USA.
- [6] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos and Roshnan K. Thomas, "Flexible Team-Based Access Control Using Contexts," *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, May 2001, Chantilly, Virginia, USA.
- [7] Weigang Wang, "Team-and-Role-Based Organizational Context and Access Control for Cooperative Hypermedia Environments," *Proceedings of the Tenth ACM Conference on Hypertext and Hypermedia*, February 1999, Darmstadt, Germany.
- [8] Arun Kumar, Neeran Karnik, and Girish Chafle, "Context Sensitivity in Role-based Access Control," *ACM SIGOPS Operating Systems Review*, Volume 36, Issue 3, July 2002.
- [9] Patrick McDaniel, "On Context in Authorization Policy," *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, June 2003, Como, Italy.
- [10] Gustaf Neumann and Mark Strembeck, "An Approach to Engineer and Enforce Context Constraints in an RBAC Environment," *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, June 2003, Como, Italy.
- [11] Hui Lei, Daby M. Sow, John S. Davis II, Guruduth Banavar and Maria R. Ebling, "The Design and Applications of a Context Service," *ACM SIGMOBILE Mobile Computing and Communications Review*, Volume 6, Issue 4, October 2002.
- [12] Kerry Taylor and James Murty, "Implementing Role Based Access Control for Federated Information Systems on the Web," *Proceedings of the Australasian Information Security Workshop*, January 2003, Adelaide, Australia.
- [13] James B. D. Joshi, Walid G. Aref, Arif Ghafoor and Eugene H. Spafford, "Security Models for Web-Based Applications," *Communications of the ACM*, Vol. 44, No. 2, February 2001, pp. 38-72.
- [14] Myong H. Kang, Joon S. Park and Judith N. Froscher, "Access Control Mechanisms for Inter-Organizational Workflow," *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, May 2001, Chantilly, Virginia, USA.
- [15] Piero Bonatti and Pierangela Samarati, "Regulating Service Access and Information Release on the Web," *Proceedings of the 7th ACM Conference on Computer and Communications Security*, November 2000, Athens, Greece.
- [16] Alfred C. Weaver, Samuel J. Dwyer III, Andrew M. Snyder, et al., "Federated, Secure Trust Networks for Distributed Healthcare IT Services," *IEEE International Conference on Industrial Informatics*, August 2003, Banff, Alberta, Canada.
- [17] Andrew M. Snyder and Alfred C. Weaver, "The e-logistics of Securing Distributed Medical Data," *IEEE International Conference on Industrial Informatics*, Banff, Alberta, Canada, August 20-25, 2003.
- [18] Longhua Zhang, Gail-Joon Ahn and Bei-Tseng Chu, "A Role-Based Delegation Framework for Healthcare Information Systems," *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, June 2002, Monterey, California, USA.
- [19] Marc Wilikens, Simone Feriti, Alberto Sanna and Marcelo Masera, "A Context-Related Authorization and Access Control Method Based on RBAC: A case study from the health care domain," *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, June 2002, Monterey, California, USA.
- [20] National Center for State Courts, Biometrics Comparison Chart, <http://ctl.ncsc.dni.us/biomet%20web/BMCompare.html>
- [21] WSDL: Web Service Description Language. <http://www.w3.org/tr/wsdl>
- [22] WS-Policy, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp>
- [23] XACML (eXtensible Access Control Markup Language), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- [24] SAML (Security Assertion Markup Language), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
- [25] Web Service Enhancements (WSE) version 2.0, <http://msdn.microsoft.com/webservices/building/wse/default.aspx>
- [26] Fred B. Schneider, "Least Privilege and More," *IEEE Security and Privacy*, Sept/Oct 2003, 1(3), pp. 55-59.