



Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX 76019

SmartAccess: An Intelligent Proactive Role-Based Authorization System

Raman Adaikkalavan and Sharma Chakravarthy

Technical Report CSE-2005-5
2005

SmartAccess: An Intelligent Proactive Role-Based Authorization System

Raman Adaikkalavan and Sharma Chakravarthy
IT Laboratory and Department of Computer Science & Engineering
The University of Texas at Arlington
{adaikkal, sharma}@cse.uta.edu

Abstract

In role-based access control (RBAC), users and objects are *assigned* to one or more roles. Users should be *active* in the role that has the required permissions before making access requests. In other words, users should be aware of the role-permission assignments i.e., what roles are required to perform operations on objects, so that they can activate the required roles. Thus, with the increase in the number of objects and with multiple roles, users often get swamped with role activations and lean towards activating all the assigned roles violating the principle of least privilege (\mathcal{PLP}). In this paper, we introduce *SmartAccess*, a user-friendly access control system that allows users to concentrate on what objects they need, rather than what role should be activated in order for accessing. Furthermore, it provides access control by preserving the \mathcal{PLP} and without any information leak. On the other hand, enforcing or implementing access control in a generalized way requires flexible and powerful suites of techniques. In this paper we analyze two approaches namely integrated and mediated that can be employed for enforcing access control. *SmartAccess* uses an event-based push-pull paradigm and supports the NIST RBAC standard and other extensions.

1 Introduction

Unabated growth of computing systems and their impact on our day-to-day activities along with the colossal amount of data available in enterprises, warrants for smart and effective access control systems. Access control models such as discretionary, mandatory and role-based (RBAC) *protect* data from unauthorized access. RBAC [1, 2, 3], where object accesses (or operations) are controlled by roles (or job functions) in an enterprise rather than a user or group, has established itself as a viable alternative to traditional discretionary and mandatory access control. RBAC has been standardized [3] and is defined in terms of four model components. Although RBAC does not provide a complete solution for all access control issues, with its rich specification it has proven to be cost effective

[4] by reducing the complexity and cost for authorization management of data. Furthermore, it is being extended for handling various constraints [5, 6, 7] such as temporal, control flow dependency and context-aware, so that it can support diverse domains in authorization management of data.

Operating systems, database management systems, and various other systems support RBAC at least in its primitive form, as it reduces the cost and complexity of access control. Current systems [5, 8, 9, 10, 11, 12, 13, 14] enforce RBAC in a binary mode i.e., they either *allow* or *deny* user access requests. Even though employees can be assigned to more than one role in an enterprise, they are *not* required to be active in all the assigned roles at all times in order to preserve the *principle of least privilege* or \mathcal{PLP} (i.e., at any given point in time no additional permissions are made available than required). This introduces some problems while providing role-based authorization as the user's access requests are granted only if they are *active* in a role that has the required permissions. For instance, take a file *analysis.scr* that can be *read* only by role *Project Manager*. Consider an user **Alice** assigned to two roles *Project Manager* and *Software developer*, but has only activated the role *Software Developer*. Thus, when **Alice** tries to *read* the file *analysis.scr* the access control system denies access as **Alice** is not active in *Project Manager*.

When user's request object access, current access control systems check for permissions based on the active roles, authorized roles (when role hierarchies are present) and other constraints (if defined in the enterprise security policy) that are *available* at the time when access requests are made and provide binary replies i.e., *allow* or *deny*. Thus, users are required to know beforehand the role-permission assignments or \mathcal{PA} , which keeps track of the operations that can be performed on objects by roles, so that they can activate the required roles. On the other hand, being aware of \mathcal{PA} or role-permission assignments is cumbersome because of various factors and they include; 1) there can be thousands of objects in enterprises, 2) number of objects is ever increasing, 3) users shift roles due to promotions, demotions, relocations, and so forth, and 4) restructuring of roles in enterprises. Binary decisions i.e., either *allow* or *deny*, alone is not sufficient in real-life situations where users often try to gain access without activating the required roles as they are \mathcal{PA} -unaware. Furthermore, with current systems, users tend to activate all the assigned roles violating the \mathcal{PLP} . Thus, smart access control systems will allow users to concentrate on the *data* or *object* that needs to be accessed, rather than the roles that fetch them those access permissions. However, they should be able to handle information leak and \mathcal{PLP} .

In this paper we provide, *SmartAccess*, a system for providing role-based authorizations. To the best of our knowledge, this is the first paper to provide feedback based access control for NIST RBAC and its extensions. Previous works [15, 16, 17, 18] deal with disclosure/release of policies, automatic trust negotiations, interactive access control to autonomic communications and web services, and so forth, but does not deal with complete NIST RBAC and its extensions that includes role hierarchies, separation of duty relations, temporal-constraints, context-aware constraints, and so forth. *SmartAccess* overcomes the

problems associated with the existing systems and provides interactive access control in a smart and effective way. It allows users to be \mathcal{PA} -unaware, interacts with users without leaking information, preserves \mathcal{PLP} and does not intricate the interaction and entangle the user.

Various approaches have been proposed for access control enforcement. Enforcing RBAC, its extensions and various other constraints in diverse underlying systems in a uniform and transparent way warrants the re-examination of the functionality as well as RBAC enforcement techniques. In this paper we analyze the ways of enforcement depending upon the openness of the underlying system. We describe the two approaches; integrated and mediated (agent-based) and their implications. We show that mediator-based approach is useful for providing access control and show how *SmartAccess* provides role-based (RB) authorizations following a event-based push-pull paradigm utilizing the mediated approach.

This paper is organized as follows. Brief introduction for RBAC is given in Section 2. Issues and problems when RB authorizations needs to be provided is elaborated in Section 3, and Section 4 addresses them. *SmartAccess* system is described in Section 5, and Section 6 discusses implementation. Potential extensions to the approach proposed in this paper are elaborated in Section 7. Related work is presented in Section 8 and conclusions are provided in Section 9.

2 Background: Role-Based Access Control

NIST RBAC Standard [3] is defined in terms of four model components and their restricted combinations:

- **CORE RBAC:** defines relationships between three basic elements (i.e., users, roles, permissions). Permissions consist of objects and associated operations that can be performed on those objects.
- **HIERARCHICAL RBAC:** defines hierarchies between roles. “A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby senior roles acquire the permissions of their juniors, and junior roles acquire the user membership of their seniors” [3].
- **STATIC SEPARATION OF DUTY (SOD) RELATIONS:** used to enforce conflicts of interest policies which may arise as a result of user gaining permissions to conflicting roles. Static SoD relations prevent these conflicts between roles by placing constraints on the assignment of users to roles.
- **DYNAMIC SOD RELATIONS:** these are similar to the static SoD that limits user permissions, but they differ by the context in which the constraints are placed. A user can be assigned to \mathcal{M} (i.e., two or more) mutually exclusive roles, but cannot be active in \mathcal{N} or more mutually exclusive roles at the the same time, where $\mathcal{N} \geq 2$ and $\mathcal{N} \leq \mathcal{M}$.

Standard RBAC alone does not suffice to handle various constraints that are required in diverse domains. For instance, hospitals, pervasive environments or spaces require temporal constraints and context-aware constraints. Furthermore hospitals might require to track the usage of patient records, thus, requiring usage-based or purpose based access control. Thus, RBAC is being extended with various constraints [5, 6, 7] to handle role-based authorizations in various domains.

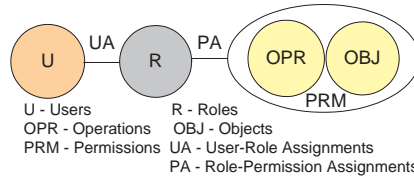


Figure 1: User-Role-Permission in RBAC

3 Issues and Problems

As RBAC reduces the complexity of authorization management and produces a lot of return of investment, more and more applications/systems support RBAC. Figure 1 shows the relationship between the basic element sets of RBAC namely; users, roles and permissions. As shown, users are assigned to roles. Permissions combining operations and objects are assigned to roles in order for the roles to access objects. This abstraction allows users to shift roles seamlessly, and allows roles to be owners of the objects rather than individual users. On the contrary, it requires users to be \mathcal{PA} aware, as users have to activate the required role in order to access objects, which is impractical. Thus, users are forced to activate all the assigned roles violating \mathcal{PCP} .

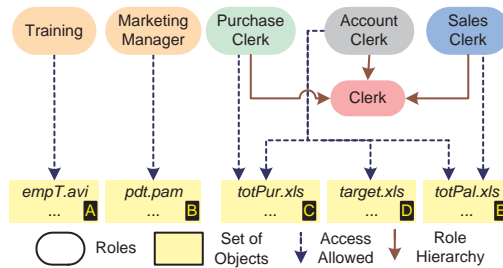


Figure 2: Role-based Access Policy

Figure 2 illustrates the role-based access policy of enterprise “ABC”. As shown, there are six roles where roles *Purchase Clerk*, *Account Clerk* and *Sales*

Clerk are senior to role *Clerk*. For simplicity, objects that can be accessed by the roles are represented as sets ($\mathcal{A}, \dots, \mathcal{E}$). For example, object `pdt.pam` belongs to set \mathcal{B} . Role *Marketing Manager* does not have any junior role and it has permissions to access those objects belonging to set \mathcal{B} . \mathcal{PA} for the role-based access policy shown in Figure 2 is shown¹ in Figure 3. For instance, role *Account Clerk* has permissions to access objects belonging to set \mathcal{C} , \mathcal{D} and \mathcal{E} .

Roles		Objects					
		Training	Marketing Manager	Clerk	Purchase Clerk	Sales Clerk	Account Clerk
A	(empT.avi, ...)	Y	N	N	N	N	N
B	(pdt.pam, ...)	N	Y	N	N	N	N
C	(toPur.xls, ...)	N	N	N	Y	Y	Y
D	(target.xls, ...)	N	N	N	N	N	Y
E	(totPal.xls, ...)	N	N	N	N	N	Y

Y - Access Allowed N - Access Denied

Figure 3: Role-Permission Assignments (\mathcal{PA})

Consider a user *Tom* assigned to roles *Marketing Manager* and *Purchase Clerk*. Even though *Tom* is *assigned* to two roles, he has to be *active* in those roles in order to access the objects. For instance, if *Tom* needs to access object `pdt.pam`, then he has to be active in role *Marketing Manager*. Below shown are just a set of sample requests and there are lot more scenarios that include context-aware constraints, separation of duty relations, etc. Following are the requests from user *Tom* when he is only active in the *Marketing Manager* role; 1) he requests for `pdt.pam` and the access is granted. 2) he requests for `totPur.xls` and the access is denied. Even though he can access the same as he is assigned to role *Purchase Clerk*, he will not be permitted as he is currently not active in the role. Thus, he *needs to know* that `totPur.xls` can be accessed by role *Purchase Clerk* and has to activate the role, which can be cumbersome with thousands of objects in enterprises. 3) he requests for `empT.avi` and the access is denied. This object can be accessed by users in role *Training*. This role is required by all employees in order to complete a training. Even though this role is not assigned to *Tom* he can acquire this role as he might be delegated [19] to activate this role temporarily for the duration of training by some authorized authority. Thus, in addition to assigned and authorized roles, delegated roles must also be checked.

Thus, from the above requests it is evident that user *Tom* should know the roles that need to be activated in order to access *any* object in the underlying system. Being aware of the \mathcal{PA} relationships is cumbersome and impossible with thousands of objects that are available in enterprises. Although the access for requests 2 and 3 is denied directly, it is possible to provide the same indirectly. In addition to the above scenario, with users shifting roles it becomes more harder as the new users assigned to the roles have to be aware of the objects that were created by the previous users in those roles.

¹For brevity, we have not shown the operations that can be performed over the objects

A straightforward manner to address the above issues is by activating *all* the roles that are assigned to the user, but it violates the \mathcal{PLP} and increases the security risks. Current systems that enforce RBAC are non-interactive and have the same problem and they burden the user. Thus, RBAC systems cannot *just* provide binary replies (i.e., either *allow* or *deny*) based on the current active roles, but need to address additional issues. However, granting accesses indirectly and based on feedback requires special attention as it should not leak information and should preserve \mathcal{PLP} . *SmartAccess* provides RBAC and addresses the problems and issues discussed above.

In addition to being smart and interactive, systems enforcing RBAC should be independent of the underlying system and should be able to provide seamless access control. In other words, they should be loosely coupled with the underlying system. For example, the same access control system should be able to handle RB authorizations in operating systems, databases, and so on. *SmartAccess* provides RBAC for the underlying systems, seamlessly, using a mediated or agent-based approach.

4 Role-Based Authorizations

User access requests should not be denied directly and should not be solely based on the active roles as discussed in section 3.

Whenever user *Tom* requests for `pdt.pam` he is granted access for the same as he is active in role *Marketing Manager* that has the required permissions. When he requests for `totPur.xls` the system checks his active roles (i.e., *Marketing Manager* in this case) or authorized roles (i.e., permissions inherited due to role hierarchy) for permissions. As role *Marketing Manager* does not have the required permission the access control system makes additional decisions. It checks if any of his assigned roles have the required permissions. In other words, the system pulls all the assigned roles from the RBAC server and checks for permissions. In our example, he is assigned to role *Purchase Clerk* that has the necessary permissions. Thus, the system notifies the user in a secure way that he should activate the role *Purchase Clerk* in order to access the object. If the user activates the role and requests again, then he can be granted the required access.

On the other hand, there can be some other object requests to which he has indirect permissions. As mentioned in section 3 he can request for an object `empT.avi` and can have indirect access permissions because of role delegation. When there is a role delegation, then the access control system checks the origin authentication and integrity of the delegation and grants the required access. Let us assume that the permission for accessing `empT.avi` is delegated to *Tom* by some authorized authority. In this case, the role delegation is verified and if it is valid then user *Tom* is given the required access. Apart from the above requests there can also be other requests that require the user to satisfy some constraints. For instance, let us assume there exists a context constraint that requires any user who needs to use object `pdt.pam` should be accessing it from

a secured network. Thus, when Tom requests for `pdt.pam` from an unsecured network the system denies his request even though he is active in *Marketing Manager*, and can notify that he needs to be in a secure network.

Confidentiality or information disclosure is one of the three main principles of information security that requires systems to prevent the disclosure of information by unauthorized accesses. Thus, whenever a user makes a access request, the following steps are performed to prevent information disclosure; i) check for access based on current credentials i.e., active roles, authorized roles, etc; if allowed goto step v; else goto next step; ii) check if the operation requested on the object can be performed by any of the user's assigned roles; if allowed send a feedback/request to the user in a secure way to activate the role that has the permissions (i.e., step iv); else goto next step. iii) similar to the above, check for the delegated roles, constraints that need to be satisfied, etc.; if allowed goto step iv; else deny access; iv) send a feedback/request to user in a secure way. v) allow the user access. In step ii, the system requests the user to activate the role *only if* the user is *assigned* to that role thereby avoiding any information disclosure. As the feedbacks/requests are made, it allows the users to concentrate on what data that needs to be accessed preserving \mathcal{PLP} .

Thus, an interactive and smart push-pull approach, where the user identity is pulled from the RBAC server for checking and appropriate messages are pushed to the user in a secure manner, is more favorable while enforcing RBAC. *SmartAccess* discussed in the next section provides RBAC using a smart push-pull approach.

5 SmartAccess

SmartAccess provides interactive role-based authorizations utilizing a smart push-pull approach. It takes a mediated-based approach and can provide access control to diverse underlying systems. Figure 4 shows the architecture of *SmartAccess*. As shown, user requests from the underlying system is sent to the *SmartAccess* system, where the role checking module checks for the access requests utilizing the RBAC server and authorization rule server and the corresponding actions are performed. All the modules are explained in detail in the following subsections.

5.1 User Request/Response Handler

Input and output of access control systems are well defined where the former is the user access requests and the latter is the allow or deny of those requests. Access requests contain the user who made the request, corresponding role and the object that needs to be accessed. In addition, there may be other domain-based information such as time of request, context of request, delegations and so forth. In *SmartAccess* we represent the basic element sets (i.e., users, roles, objects, etc.) as follows:

1. UI represents user identity i.e., user names, user applications or so forth.

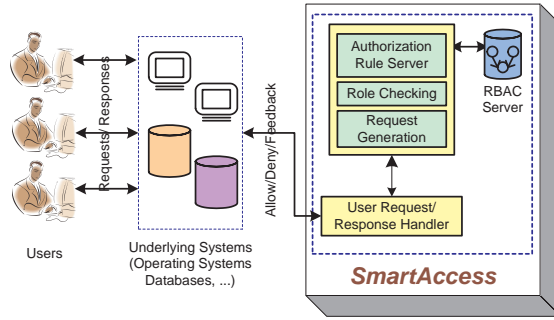


Figure 4: *SmartAccess* RB Authorizations

2. \mathcal{R} represents job functions (roles) in an enterprise.
3. \mathcal{P} represents the operations that can be carried out on objects (permissions), where OBJ represents the objects and OP represents the operations that are permitted.
4. \mathcal{REQ} represents user access requests.
5. \mathcal{ARS} represents user's active roles set i.e., all the roles in which a user is active. Similarly, \mathcal{ASRS} represent assigned roles set and \mathcal{DRS} represents delegated roles set.

A typical request i.e., $req \in \mathcal{REQ}$, consists of the user identity UI , object OBJ and operation OP . For example, consider user Tom's (i.e., UI) request for reading (i.e., OP) `pdt.pam` (i.e., OBJ). This request is received by this module and is sent to the *Role Checking* module where the access permissions are checked. If the user does not have the access permission, this module sends *feedback/requests* to the user in a secure way. Figure 5 illustrates the functioning of the user access request handler.

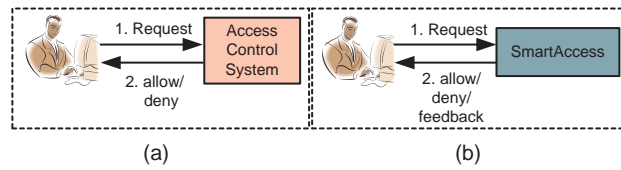


Figure 5: User Access Request/Response

5.2 RBAC Server

Enterprises that utilize RBAC for controlling accesses define their security policies in terms of RBAC elements such as UI , \mathcal{R} and \mathcal{P} and are maintained in the

RBAC server. In other words, the server maintains all the users, roles, objects, permissions in the enterprise. For instance, lists that are maintained in the server for the role-based access policy defined in Figure 2 are shown in Figure 6. We assume three enterprise users and the corresponding list is shown in Figure 6(a). As illustrated in Figure 2 there are six roles and the corresponding list is shown in Figure 6(b).

Users	Roles	User	Assigned Roles
Tom	Training	Tom	Purchase Clerk
Jim	Marketing Manager	Tom	Marketing Manager
Jane	Purchase Clerk		
	Account Clerk		
	Sales Clerk		
	Clerk		
		User	Active Roles
		Tom	Marketing Manager

(a) (b) (c) (d)

Figure 6: User, Roles and Relationships

In addition, the server also maintains the relationships between RBAC elements, for example, user-role assignments ($\mathcal{U}\mathcal{I}$ and \mathcal{R}) have an M:N (i.e., many-to-many) relationship. Similarly, role-permission assignments represent the relationship between (\mathcal{R} and \mathcal{P}), role hierarchies represent the relationship between (\mathcal{R} and \mathcal{R}), and so forth. Figure 6(c) shows the user-role assignments and Figure 6(d) shows the active roles of users. Role-permission relationships are illustrated in Figure 7. Similarly, *constraints* such as separation of duty relations, temporal, context-aware, and others are placed on the relationships and are maintained in the server.

Role	Permission	
	Objects	Operation
Training	empT.avi	r
Marketing Manager	Pdt.pam	rwX
Purchase Clerk	totPur.xls	rwX
Account Clerk	totPur.xls	r
	target.xls	rwX
	totPal.xls	r
Sales Clerk	totPal	RwX
Clerk		

Figure 7: Role-Permission Relationships

5.3 Role Checking

Whenever an access request is received by the request handler it is propagated to this module, where the access decisions are taken. In a system that enforce RBAC, user requests can be categorized into either *administrative* requests or *access* requests. While the former deals with activating roles, deactivate roles, and so forth, latter deals with the object accesses. Even though administrative requests such as role activations can be made smart by asking the user to satisfy more constraints in order to activate the requested role, object access requests are more critical for the smart decision making as discussed in Section 3.

5.3.1 Object Access Request Handler

Users object access requests are handled using the algorithm provided below. First, we will analyze the input and output of the algorithm. The input consists of the session id² SID , object in the underlying system OBJ and the operation that needs to be performed OP . Furthermore, requests can have other free attributes A_i ($i = 1 \dots n$) that are required to satisfy other constraints defined in the enterprise security policy. Thus, an user's *object* access request is similar to: $\{SID, OBJ, OP, [A_1 \dots A_n]\}$. Once the input is received it is processed and the algorithm takes one of the following actions; i) allow access, ii) deny access, or iii) request for role activations or other constraint satisfaction.

Algorithm 1 handles the object access request. In the first step it retrieves all the roles (ARS) that are active in the session SID that have the permission for accessing the requested object OBJ . Second, it retrieves all the roles ($ASRS$) that are assigned to the user who is the session owner and that have the required object access permission. If both the retrieved role sets (ARS and $ASRS$) are empty then the user does not have the required permission and the access is denied. When ARS is not empty then additional constraints that are required by the enterprise is checked; if all the constraints are satisfied then access is allowed; else the constraint satisfaction request is generated. When ARS is empty and $ASRS$ is not empty, then it indicates that the user has indirect permission to access the OBJ . In other words he has to activate certain roles in order to access the OBJ , thus, a role request is generated and sent to the user. However, request generation and sending it to the user should address other issues that are explained in the next section.

Let us consider the access requests by user Tom from Section 3. When the user is requesting for `totPur.xls`, the input to the algorithm is $\{SID, totPur.xls, OP\}$ ³. Role set ARS will be empty as role *Marketing Manager* does not have the permission. Role set $ASRS$ will have the role *Purchase Clerk* as it has the required permission. Now, the algorithm generates the requesting the user to activate *Purchase Clerk* in order for accessing. Even though there can be thousands of objects in enterprises the user need not know the role-permission assignments i.e., \mathcal{PA} unaware, as the algorithm provides feedback via requests rather than just binary taking decisions.

5.3.2 Interactions in Other Components of RBAC

Algorithm 1 handles the user's object access request when the enterprise security policy does not have role hierarchies, separation of duty relations, etc. When enterprise security policy contains these then additional condition checking has to be performed in the Algorithm 1.

For instance, when role hierarchies are used by the enterprise it can be handled as shown below;

²A user can have multiple sessions, but a session is associated with only one user and can have many active roles [3].

³For brevity, we ignore the values for SID and OP

Algorithm 1: User's Object Access Request Handler

INPUT: $\{SID, OBJ, OP, [A_1 \dots A_n]\}$
OUTPUT: $ALLOW, DENY, REQS$

//Getting Active Role Sets of SID from RBAC server.
Retrieve ARS of SID that has permission for OBJ, OP ;
//Getting Assigned Role Sets of the user using SID from RBAC server.
Retrieve $ASRS$ of SID that has permission for OBJ, OP ;

if (ARS is NOT EMPTY) **then**
 if ($A_1 \dots A_n$ is satisfied) **then**
 | Allow object access;
 else if ($ConsSatiReq$ already generated) **then**
 | // $ConsSatiReq$ is the constraint satisfaction request.
 | Deny object access;
 else
 | Generate a $ConsSatiReq$ and send to the user in a secure way;
 end
else if ($ASRS$ is NOT EMPTY) **then**
 if $RoleRequest$ already generated **then**
 | Deny object access;
 else
 | Generate a $RoleRequest$ and send to the user in a secure way;
 end
else
 | Deny object access;
end

- retrieve all the *authorized* role sets of *SID* from RBAC server that has the permissions for accessing *OBJ*.
- if authorized role sets is not empty then check for free attributes (if any) and take appropriate actions.
- if authorized role set is empty then generate requests (if required)

Similar to the above, additional conditions can be added so that the algorithm can support other components of RBAC and provide feedback.

5.3.3 Information Leak and \mathcal{PLP}

As Algorithm 1 provides feedback to the user in order to allow the user to access objects, it should not leak any information. As the requests generated in the algorithm are based on the roles that are retrieved (i.e., *ARS* and *ASRS*), which in turn are assigned to the user, it does not ask the user to activate any unassigned role and thus do not leak any information. Further more, as the request generations are based on the *assigned* roles that have the permission \mathcal{P} for accessing the object *OBJ* it does not intricate the interaction and entangle the user. Thus, by providing the feedback to the user via role/constraint satisfaction requests it allows the user to be \mathcal{PA} unaware and thus preserving \mathcal{PLP} .

5.4 Requests Generation

Requests can be either role-requests or constraint satisfaction requests. When any one of the roles from *ASRS* has to be activated or any other constraints have to be satisfied by the user in order to allow an access request, requests are generated and sent to the user. Requests that are generated should be sent to the user in a secure way. Although there are many ways, digital signatures [20] will be advantageous as it provides both data integrity and data origin authentication. We provide a mechanism of how the request is sent using digital signatures, but it can also be done in a more secure way using other mechanisms. Digital signatures are similar to handwritten signatures as a single entity can sign some data using a private key, but any number of entities can read the signature using a public key and verify its accuracy. In *SmartAccess* data containing role name, etc. are created as a digital signature with the private key of the server and pushed to the user. Once received, users can verify the signature using the server's public key for authenticity and integrity. Once verified it is at the user's discretion whether to activate the role and make another object access request. On the other hand, when the user is delegated with the role, he or she needs to use it when requesting access for objects. In this case, the user can send a role-delegate digital signature consisting of user identity and the delegated role to the *SmartAccess* server which was obtained from the role authentication/delegation server.

Request flag: Request generation creates the signature and sends it to the user. In some cases the user might not want to activate a role or satisfy a constraint or he might not have a role-delegate certificate, in which case the user issues the same request again. Thus, when the same user request is received again, the role checking module should not check for access if the requested credentials are not updated after the request is sent. In this case, requests are not regenerated in order to avoid cycles, and is carried out by setting an access flag for that user object access request.

5.5 Authorization Rule Server

RBAC server explained in Section 5.2 maintains all the lists that correspond to the basic elements of RBAC. Enterprise security policies are maintained in terms of authorization rules in our system. Authorization rules, that are used in *SmartAccess* are similar to the *active* or Event-Condition-Action rules [21] that were used to make the underlying system active. Thus, whenever an access request is made in the underlying system, it triggers an event along with the parameters. This event triggers a rule, which checks for the conditions and takes the appropriate actions. Role checking is performed as part of the condition part of the authorization rule and the action part corresponds to the replies that should be provided to the user, namely; allow, deny or feedback/request. Thus, these rules can be either integrated into the underlying system or can be maintained outside the system. By maintaining them outside the system, the same set of rules can be used to enforce access control in diverse systems. Below, we provide the approaches for maintaining the rules and triggering the events when a access request is received.

5.5.1 Approaches for Role-Based Authorizations

Systems that enforce RBAC should be generalized and be able to provide access control to diverse underlying systems (i.e., operating systems, databases, etc.) in a uniform and transparent way. Although the system that provide access control act as the reference monitor it should be loosely coupled with the underlying system. Below we analyze two kind of approaches that can be exploited for providing role-based authorizations.

Integrated Role-Based Authorization: With the integrated approach the underlying system should be open to modification to incorporate access control modules (or authorization rules). In other words, this approach assumes that the kernel of the underlying system is understood so that it can be modified. There are many advantages to this approach and they are summarized as follows; *i)* flexibility to fine tune the access control modules resulting in good performance, *ii)* only the minimum amount of code that is required can be added, *iii)* additional functionalities can be easily incorporated, and *iv)* applications can be easily modularized and maintained. There are many disadvantages to this approach and they are; *i)* requires access to the internals of the underlying

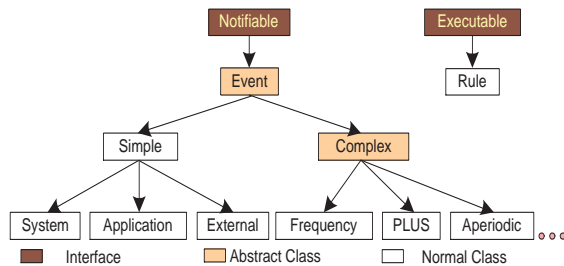


Figure 8: Class Hierarchies

system including the source code, *ii*) it is not cost and time effective, and *iii*) it is not generalized as it is tightly coupled and customized.

Mediated Role-Based Authorization: As opposed to the integrated approach, in the mediated approach the internals of the underlying system need not be accessed. In this approach, it is assumed that the underlying approach provides some hooks which can be exploited for triggering events and invoking authorization rules. When this approach is employed, the mediator or agent acts as the access control reference monitor for the underlying system. Thus, conditions are evaluated in the mediator when there is a user request. The main advantages of the mediated approach are; *i*) provides access control in a generalized and uniform way, *ii*) not customized to any underlying system, *iii*) it can provide role-based authorizations to diverse domains and not domain-specific, and *iv*) cost and time effective.

SmartAccess Role-Based Authorizations: As shown in Figure 4 *SmartAccess* follows the mediated approach and handles all the access requests that are generated in the underlying system. Whenever the underlying system receives an access request it triggers an event which in turn invokes a rule. Once the authorization rule is triggered, it is executed, which in turn evaluates the necessary conditions and triggers an action which can be allow, deny or provide feedback. Whenever there is an object access request, then the rule has to perform role checking as explained in Section 5.3.1. On the other hand, when the request is administrative requests, such as adding roles, users, etc. then the rule should also update lists that are maintained in the RBAC server.

6 Implementation

In the *SmartAccess* system, the RBAC server is a typical server that maintains the lists as explained in Section 5.2. Role checking is performed once the authorization rule is triggered. Authorization rule server is an active object oriented system that has an event based rule capability named the Event-Condition-Action model which supports various needs of applications using a uniform framework. It is implemented in both C++ and Java, and the key

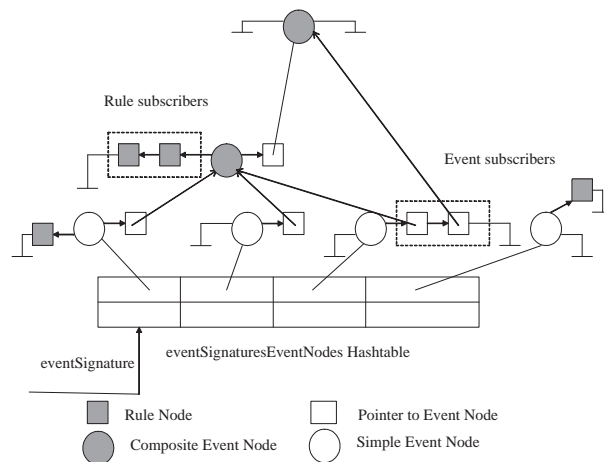


Figure 9: Event Detector

classes that are used in the implementation are shown in Figure8. Notifiable and Executable are Interfaces, which indicates the behavior of classes that implement the interface. Event and complex event are abstract classes as they need not be instantiated. All the other classes are instantiable. There is one class for each complex event operator and a class for simple event. Simple events include system events (e.g., temporal), application generated events, and external events (e.g., from sensors). As shown Notifiable interface has a notifyEvent method that is implemented by all the event operator classes. This method implements the detection logic in an operator class according to the semantics of the operator.

In the server, a reactive object is an object that has traditional object definition plus an event interface and a notifiable object is capable of being informed of the occurrence of some event. The event interface lets the object designate some or all of reactive object methods as primitive event generators. Together, both the kind of objects enable asynchronous communication with the rest of the system. Its external monitoring module supports external events such as those from sensors, thus, supporting location/context-aware events. Figure 9 illustrates an event detector that is responsible for processing all the notifications from different objects and eventually signaling to the rules that some event has occurred triggering them. As shown, an event detector consists of simple event, complex event and rule nodes, and various pointers. Event nodes contain event subscribers and rule subscribers. Thus, whenever an event is detected its corresponding event subscribers and rule subscribers are notified. “eventSignatureEventNodes” hashtable shown contains the signature of all events and propagates the simple event occurrences to its nodes. Event detector follows a bottom-up data flow architecture. Thus, when any event occurs the corresponding rules are triggered. Once the rules are triggered they check the associated

conditions and trigger the necessary actions.

7 Future Directions

Even though, in this paper we show only the algorithm for handling object access requests for core RBAC, *SmartAccess* can support other components by adding these functionalities in the RBAC server and modifying the corresponding algorithms. Similarly, other constraints such as time of the day, IP address, quotas based on bandwidth or time, and so on can be supported. All the above additional functions can be supported seamlessly in *SmartAccess* as it takes a generalized approach for providing RB authorizations. In general, any type of constraint can be supported when the required attribute values are provided. For example, assume that user *Jack* is allowed to perform any action only between 9.00 a.m. to 5.00 p.m. on weekdays. When *Jack* requests at 6.00 p.m. using the following values (*Jack*, $\{ARS|DRS\}$, *OBJ*), then *SmartAccess* checks the enterprise policy and denies access. There may be a slight increase in user response time when compared to the current systems as *SmartAccess* checks the roles and provides feedback (if required) for every request that comes from the user. Performance optimizations are possible by caching the user request history while checking for roles and permissions. While maintaining the user request history, if there is any change in the role policy then it should be propagated to the role checking module so that access is not granted for an unauthorized user.

8 Related Work

None of the existing systems that enforce RBAC and its extensions, to the best of our knowledge, provide a feedback based RB authorizations. Thus, we explain some of the current systems along with the supported features.

OASIS [8, 22] supports dynamic role deactivations by use of rules, and it does not support role hierarchies explicitly, and cardinality constraints. It also supports minimal temporal constraints, and context dependent constraints in the form of environmental predicates. Adage [9, 23], a rule-based authorization system for distributed applications, supports separation of duty by using history based constraints. It does not support important RBAC features such as role hierarchies and cardinality constraints and it requires the administrators to specify the authorization rules manually. Attribute-Based RBAC [11, 24] is a rule-based model that was developed to assign users to roles automatically, based on the authorization rules defined by enterprise administrators. X-GTRBAC [25] is an XML based policy specification framework and architecture for enterprise wide access control. This framework supports the generalized temporal RBAC specifications.

Crampton [13] propose a model for a scalable role-based reference monitor, based on dynamic access control structures, that is used to enforce constraints.

The enforcement model can only enforce constraints in which the constraint set has no more than two elements and this is because of the limitation of the blacklists used. Oppiliger et. al. [10] show how attribute certificates are used for implementing role-based authorization and access controls. It associates each entity that can take part in a role with an attribute certificate so that authorizations can be made using the certificates. It does not provide a way of enforcing the complete RBAC reference model and its extensions. Bertino et. al. [26] and Koch et. al. [27] show how multipolicy access control is supported but they do not explicitly consider the extended RBAC with various constraints and active security. Neumann et. al. [12] show how context constraints are enforced in an RBAC environment.

There has been some work [15, 16, 17, 18] done in the context of access control mechanisms that provide more than just binary replies. In general these systems deal with the disclosure/release of policies, automatic trust negotiations, interactive access control to autonomic communications and web services, and so forth. Furthermore, these systems does not deal with complete NIST RBAC [3] that includes role hierarchies and separation of duty relations, and its extensions [5, 6, 7] such as temporal-constraints, context-aware constraints, and so forth.

9 Conclusions

In this paper we have provided *SmartAccess*, a smart push-pull approach for supporting RB authorizations. *SmartAccess* is proactive in a way that it provides the necessary feedback to the user acting in anticipation of future problems that the user may face when he is requesting for access. *SmartAccess* uses smart pull for getting the required information from RBAC server and sends feedback to the users. By providing feedback without information leak it allows the users concentrate on what data needs to be accessed rather than the roles that are required for access and preserves the principle of least privilege. In addition, by using event based authorization rules and a mediated approach it provides RB authorizations to diverse underlying systems in a uniform and transparent way.

References

- [1] D. F. Ferraiolo, J. A. Cugini, and D. R. Kuhn, "Role-Based Access Control: Features and Motivations," in *Proc. of Computer Security Applications Conference*, 1995.
- [2] R. S. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
- [3] *RBAC Standard, ANSI INCITS 359-2004*, ANSI INCITS 359-2004, International Committee for Information Technology Standards, 2004.

- [4] *The Economic Impact of Role-Based Access Control*, RTI Project Number: 07007.012, National Institute of Standards and Technology (NIST), 2002. [Online]. Available: <http://www.nist.gov/director/prog-ofc/report02-1.pdf>
- [5] M. J. Moyer and M. Ahamad, "Generalized Role-Based Access Control," in *Proc. of ICDCS*, 2001.
- [6] J. B. D. Joshi, E. Bertino, U. Latif, and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model," *TKDE*, vol. 17, no. 1, 2005.
- [7] Q. He., "Privacy Enforcement with an Extended Role-Based Access Control Model," Department of Computer Science, NCSU, Tech. Rep. TR-2003-09, 2003.
- [8] J. Bacon, K. Moody, and W. Yao, "A Model of OASIS Role-Based Access Control and its Support for Active Security," *TISSEC*, vol. 5, no. 4, 2002.
- [9] R. T. Simon and M. E. Zurko, "Separation of Duty in Role-Based Environments," in *Proc. of IEEE CSF Workshop*, 1997.
- [10] R. Oppliger, G. Pernul, and C. Strauss, "Using Attribute Certificates to Implement Role-Based Authorization and Access Controls," in *Proc. of Fachtagung Sicherheit in Informationssystemen (SIS 2000)*, 2000.
- [11] M. A. Al-Kahtani and R. Sandhu, "A Model for Attribute-Based User-Role Assignment," in *Proc. of the Annual Computer Security Applications Conference*, 2002.
- [12] G. Neumann and M. Strembeck, "An Approach to Engineer and Enforce Context Constraints in an RBAC Environment," in *Proc. of the ACM Symposium on Access Control Models and Technologies*, 2003.
- [13] J. Crampton, "Specifying and Enforcing Constraints in Role-Based Access Control," in *Proc. of the ACM Symposium on Access Control Models and Technologies*, 2003.
- [14] M. Strembeck, "Conflict Checking of Separation of Duty Constraints in RBAC - Implementation Experiences," in *Proc. of the Conference on Software Engineering*, 2004.
- [15] P. A. Bonatti and P. Samarati, "A uniform framework for regulating service access and information release on the web," *J. Comput. Secur.*, vol. 10, no. 3, pp. 241–271, 2002.
- [16] 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.*, vol. 6, no. 1, pp. 1–42, 2003.

- [17] H. Koshutanski and F. E. Massacci, “Deduction, abduction and induction, the reasoning services for access control in autonomic communication,” in *Proceedings, IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC)*, 2004.
- [18] —, “Interactive access control for web services,” in *Proceedings, IFIP International Information Security Conference*, 2004, pp. 151–166.
- [19] S. Na and S. Cheon, “Role delegation in role-based access control,” in *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*. New York, NY, USA: ACM Press, 2000, pp. 39–44.
- [20] C. Adams and S. Lloyd, *Understanding PKI (2nd ed.)*. Addison-Wesley, 2003.
- [21] J. Widom and S. Ceri, *Active Database Systems: Triggers and Rules*. Morgan Kaufmann Publishers, Inc., 1996.
- [22] J. Bacon, M. Lloyd, and K. Moody, “Translating Role-Based Access Control Policy Within Context.” in *Proc. of POLICY*, 2001.
- [23] J. Hoagland, “Adage,” 1999. [Online]. Available: <http://seclab.cs.ucdavis.edu/secsem2/01-20-99.pdf>
- [24] M. A. Al-Kahtani and R. Sandhu, “Induced Role Hierarchies with Attribute-Based RBAC,” in *Proc. of the ACM Symposium on Access Control Models and Technologies*, 2003.
- [25] R. Bhatti, “X-GTRBAC: An XML-based Policy Specification Framework and Architecture for Enterprise-Wide Access Control,” Master’s thesis, Dept. of Electrical and Computer Science, Purdue University, 2003.
- [26] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, “A System to Specify and Manage Multipolicy Access Control Models,” in *Proc. of POLICY*, 2002.
- [27] M. Koch, L. V. Mancini, and F. Parisi-Presicce, “On the specification and evolution of access control policies,” in *Proc. of the ACM Symposium on Access Control Models and Technologies*, 2001.