elpub2002 Proceedings — J. A. Carvalho et. al. (Eds.) — VWF Berlin, 2002

Implementing RBAC Policies in a Web Server

Daniel Sanz, Paloma Díaz, and Ignacio Aedo

Laboratorio DEI. Departamento de Informática.
Universidad Carlos III de Madrid
Avda. de la Universidad, 30, 28911 Leganés. Spain.
{dsanz@dei.inf, pdp@inf, aedo@ia}.uc3m.es

Abstract. Security is a key requirement in multi-user hypermedia systems, where the ability of different users to access and manipulate the information depend on their needs and responsibilities. If security policies are specified in terms of individual users and low-level abstractions not related to the hypermedia domain, security administration becomes complex and prone to error. This paper describes how an RBAC (Role Based Access Control) module is integrated into a web server that is treated as a hypermedia system instead of as a set of files, programs and network protocols. This implies the definition of a set of hypermedia related operations that authorised roles can execute on the system objects.

1 Introduction

Hyperdocuments, widely represented by web-based applications, have experienced a considerable growth in the last years, offering more and more services and contents. Indeed, web sites are used on private networks to provide more specific services since the portability of web browsers makes access to electronic information consistent and concurrent by all users [1]. Hypermedia applications should offer different views of the same information and different manipulation abilities in order to fit more specific users needs and responsibilities in a particular context [3]. In order to be efficient, security policies have to be defined in terms of nodes, contents, links and, in summary, hypermedia components. Such security policies can follow different approaches, including DAC (Discretionary Access Control), MAC (Mandatory Access Control) and RBAC (Role Based Access Control) models, being the latter a quite convenient solution to make security management a more efficient and less prone to error task [6].

This paper addresses how to integrate an RBAC module into the Apache httpd web server. This server has been chosen to test our ideas due to its powerful and modular design, widespread use, support for most common platforms, high configurability, full source code availability and extensibility.

Two major related questions need to be discussed: the hypermedia model and the security model assumed in the implementation of the module. A third question is also considered: performance should not be damaged, so the level of efficiency (mainly considered as the response time) should be kept as similar as possible to the one offered by the server without the RBAC module.

2 Related Works

One of the most common problems in network systems management is the complexity of the security administration, due to the difficulty of managing authorizations in an efficient way. Traditional access policies (such as DAC or MAC) seem insufficient for meeting today's security needs in most hypermedia and web systems since they are chiefly focused on group-based policies. Thus, authorizations are assigned in terms of users, so as the number of users becomes larger, the associations user/authorizations tends to be unmanageable and security management becomes a laborious and prone to error tasks. Role based access control (RBAC) is a term used to refer to a class of security mechanisms that regulate the access to resources through organizational identities called roles [7]. A role gathers a set of allowed activities for the users holding such a role. Instead of granting privileges on an individual basis, privileges are assigned to roles, so users can be added or removed from role membership according to their job functions or position within an organization without modifying the access structure. Although user's responsibilities are expected to change over time, the authority assigned to roles is normally much more stable. Moreover, as new applications and operations are incorporated to the system, roles can be assigned or revoked permissions as needed.

This paper describes how an RBAC model described in [2] is integrated into a web server. Thus, classical RBAC concepts and models [9, 6, 5, 8, 7] have provided the basis for some key concepts of the security model assumed by the implemented module as it will be discussed in the next section.

The integration of RBAC policies into web servers has been already discussed in [4, 1]. Despite of performance considerations, the proposed design maps the operations directly into HTTP methods, takes URLs as the system objects and translates the URL to filenames. Operations are tied to the transmission protocol, so abstract manipulation abilities over the hyperdocument are not considered. The object concept does not take into account relationships among objects, nor consider other hypermedia elements, such as links or contents, so that multilevel access policies can not be supported. What we propose in this paper is to treat the web

site as a hypermedia system instead of as a set of files, programs and network protocols and, consequently, to specify security rules in terms of hypermedia elements and operations with a view to integrating security into the whole hypermedia or web engineering process.

3 MARAH: An RBAC Model for Hypermedia

The security model assumes some well-known security principles:

1. **Well-formed transactions**: users manipulate information only through a number of controlled programs, which are the operations of the model (see below).
2. **Authenticated users**: only authorised users can perform operations.
3. **Least Privilege**: operations have only those privileges they require to accomplish their objectives, and users are granted only the abilities they need to do their work.
4. **Delegation of authority**: security management is not centralised; some actions which are not too critical can be delegated to the users responsibility.
5. **Positive and negative authorizations**: the model supports both positive authorizations (that grant access) as well as negative ones (that deny access). When the number of users is unmanageable, as happens with most web-based applications, both mechanisms are desirable to lighten the security management tasks.
6. **Data abstraction**: security categories are defined in terms of manipulations abilities pertaining to the application domain.
7. **Separation of duties**: sensitive tasks must require the action of mutually exclusive subjects, in order to avoid collaboration between various job related capabilities.

The security model is summarised in Tab. 1. *Subjects* are active entities that perform actions on *objects* (passive entities). The *operations* are the actions which can be done on the hyperdocument (e.g. managing nodes or links). The manipulation abilities supported in the application are the *security categories*, that define a partial order relationship, where each category adds permissions for manipulating the hyperdocument to the previous one. Three access categories are defined: *browsing* to retrieve objects; *personalising* to browse and create a personalised version of an object, for private use of a role or team; and *editing* to browse, personalise and modify objects. The least privilege principle leads to define the *classification of operations* and the *classification of objects*. While the former specifies the minimum category required to perform an operation, the latter states the most permissive operation an object allows. The security policy is defined in terms of the *confidentiality* and *clearance* functions. The confidentiality relationship establishes a negative ACL for

each object. The clearance relationships allow to define context-dependent subject authorizations, that is, to specify manipulation abilities for some specific subjects with respect to some specific objects. The *separation of duty* is a symmetric relation that defines mutually exclusive subjects (that is, those roles that cannot be assumed concurrently by the same user). Finally, the *transition function* determines if an operation initiated by a subject is or not safe. Next subsections describe some relevant features of this model.

Table 1. Formal specification of the security model

Model Element	Specification
Subjects	$S = \{s_i \mid i = 1, \ldots, n (n \in \mathbf{N})\}$
Objects	$O = \{o_i \mid i = 1, \ldots, m \ \ (m \in \mathbf{N})\}$
Operations	$Op = \{op_i \mid i = 1, \ldots, p \ \ (p \in \mathbf{N})\}$
Security Categories	$Sc = \{sc_i \mid i = 1, \ldots, q \ \ sc_{i-1} \subset sc_i, \forall sc_i \in Sc, (q \in \mathbf{N})\}$
Separation of Duty	$SD = \{\langle s_i, s_j \rangle \mid i, j = 1, \ldots, n \ \ i \neq j, (n \in \mathbf{N})\}$
Classification of Operations	$\omega : Op \rightarrow Sc$
Classification of Objects	$\delta : O \rightarrow Sc$
Confidentiality	$\psi : O \rightarrow S^n$
Clearance	$\phi : O \times S \rightarrow Sc$
Transition	$\theta : Op \times O^n \times S \rightarrow O^m, \ \ (n, m \in \mathbf{N})$

Using roles and teams to model the concept of subject. Subject modelling considers roles, teams and users. Roles are job functions or organizational positions, while teams gather a set of roles, whether to represent groups of users or just to simplify the security management tasks. Individual users must be assigned to one or more roles and they cannot belong to teams directly but take part in them through the roles they assume.

The roles set is a partial order, defined with two irreflexive, transitive and antisymmetric relations, and it is represented as a DAG (Directed Acyclic Graph). The two role composition mechanisms used to define hierarchies are: *generalization* ("is-a" relationship) and *aggregation* ("whole-part" relationship). The former allows to establish the relationship among a general role and its specializations, while the latter is used to define the composition of teams. Generalization can be *exclusive*, what implies that a user will be only allowed to be assume one of the child roles. Users can be assigned to general roles, since generalization hierarchies are partial, whereas aggregation are total.

Controlling the users allocation to roles and teams. There are two issues to take into account when assigning specific users to a role: (1) if the role has constraints in the number of allocations and (2) if there are roles who cannot be held by the same user at the same time.

Concerning the first issue, teams and roles support the definition of cardinalities, both minimum and maximum. Membership bound remains unconstrained if the value is unspecified. Let $\eta_{r\downarrow}$ be the minimum cardinality of role r, $\eta_{r\uparrow}$ the maximum and $\#_r$ the number of directly assigned users of r. Then, the number of users of r are calculated as follows:

$$\eta_r = \#_r + \sum_{r' \succ r} \eta_{r'} \tag{1}$$

where $\succ$ refers to every child role of r, whether aggregated or generalised. Note that $\eta_r = \#_r$ for leaf roles, while $\#_r = 0$ for teams. Some consistency constraints are applied in order to ensure a proper cardinality definition, taking into account if the hierarchy is partial or total. For example, the value of $\eta_{r\uparrow}$ of a member of a team cannot be higher than the value of $\eta_{r\uparrow}$ for the team itself.

To deal with the second constraint, we use the separation of duties as a symmetric relationship that involves pairs of mutually exclusive roles. The *static separation of duty* (SSD) assures that no user is assigned to two exclusive roles, while the *dynamic separation of duty* (DSD) allows this assignation, but avoiding the simultaneous use of the exclusive roles by the same user. The separation of duties is propagated to every child role.

Propagating authorizations in the subjects hierarchies. Subjects are then defined by means of a hierarchy of roles and teams, an structure that can be used to propagate authorizations and reduce the security management efforts. The following inheritance rules are defined [2]:

1. Direct propagation of authorization: Each role inherits the permissions given by the confidentiality and clearance relationships that apply to its parents.
2. Authorization propagation in nested relationships: If there are nested generalizations, the child assumes the security rules that apply to the most specialised role, that is, to its immediate ancestor.
3. Authorization propagation in parallel relationships: If a child role is generalised by several alternative ones, it assumes the most permissive authorization.
4. Direct assignment of authorization: If a role takes part in a team and has no authorization, neither directly assigned nor inherited, it assumes the team authorization.
5. Authorization overriding: Direct propagation is inhibited if the child role is explicitly assigned a permission for the same object.

Objects composition mechanisms. Object modelling is also based on generalizations and aggregations, with the purpose of gathering the hyperdocument's structure and semantics [3]. The object set has the same properties than the role set. The concept of *domain* allows hierarchical structures to be used as objects themselves: a domain of an object o contains o, the domain of the objects o' aggregated by o and the domain of the objects o'' generalised by o. Each hyperdocument has a root domain representing the hyperdocument itself. In practise, domains allows objects to be applied the above inheritance rules with respect to the classification of objects (see Tab. 1).

4 Implementation of the RBAC Module

This section describes the implementation of an RBAC module that has been embedded into the Apache httpd web server. First, we will discuss how the security model is instantiated to the web domain, analysing some key decisions such as what is an object, how to name entities or how to ensure a proper object manipulation. Then, the system architecture is described.

Objects. Web servers do not explicitly distinguish between nodes and contents, a necessary separation to support multilevel policies. In the module, we have considered that when an HTML tag is used to refer to or to embed elements (e.g. links and images respectively), such elements are also considered as objects that inherit the properties of the referred element, and are detected automatically. Non HTML contents (e.g. images) are treated as a whole. This facilitates the management, because objects are closely related with files, so the granularity level seems reasonable. Indeed, objects are identified with the filename of the resource, what hides objects names to users, who will refer to them by the URL, and can reduce the number of objects when aliases are defined. The security administrator needs to define the object name, the hierarchical relationships it has with other objects, its security category (δ), and the set of roles with denied access (ψ). δ function is propagated through the object DAG according to rules defined above. Thus, every object must have defined δ. The ψ function works slightly different:

$$\psi(o) = \emptyset \rightarrow \psi(o) = \bigcup_{o' \underset{g \otimes a}{\succ} o} \psi(o'), \tag{2}$$

where $o' \underset{g \otimes a}{\succ} o$ refers to $o' \underset{g}{\succ} o$ if $\exists\ o'$. If not, refers to $o' \underset{a}{\succ} o$.

Users. The users set is directly taken from the Apache database since authentication will be delegated to the server instead of being part of the RBAC module. Once

the server and the browser have established how to authenticate users, the module asks Apache what user is involved when a request arrives. The security administrator should consider only one authentication realm for all the RBAC controlled web space. The specification of the RBAC user set must be in conformance with the users defined in Apache, because the module needs a defined user, while Apache needs to authenticate it. Groups and ACLs are overridden by the RBAC module. The security administrator only needs to define the user name and the list of her possible roles.

Roles. The security administrator defines the role set specifying the role name, cardinalities, hierarchical relationships (including exclusive generalizations), the SSD related roles and the clearances for objects (ϕ). The SSD relationship only needs to be defined in one role, due to its symmetry. The exclusive generalizations are implemented by adding an SSD relation between every pair of child roles. In order to simplify the access to permissions, the nACLs are transformed into negative clearances, that is, $\forall o, \forall r \in \psi(o) \Rightarrow \phi(r, o) =$"denied". Negative clearances are propagated with maximum priority, so the nACL is inherited too through the roles hierarchy. After this process, every role has defined a clearance for each object.

Operations. In order to provide an extensible and abstract operation set, the use of HTTP methods is avoided. The operation is derived from the URL as follows: if the URL contains a query, the identifier "rbac_op" indicates the name of the operation. In other case, the "browse" operation is assumed. This naming scheme does not interfere with other parts of the URL, provides independence with respect to HTTP methods, and allows to easily add new operations in the future. The supported operation set is defined by the module, who is responsible for defining a proper classification of operations. Each operation defines a handler that provides all operation functionality.

RBAC system architecture. The system architecture is depicted in Fig. 1, that describes the Apache request cycle, focusing on the RBAC phase. The modular design of the request loop allows to modify the behaviour of the server at any stage of the cycle, leaving the rest untouched. Like [4, 1], all processing is done in the server, so any web browser can be used.

Note that tasks such as mapping URLs to filenames, checking access and determining the MIME type of the document are performed by Apache before the RBAC module is initiated. When RBAC starts, Apache has determined all relevant information about the request, so the only responsibility for RBAC module is preparing a response for the user according to the security rules. Again, the task of sending the response to the browser is better performed by the Apache handlers. In the

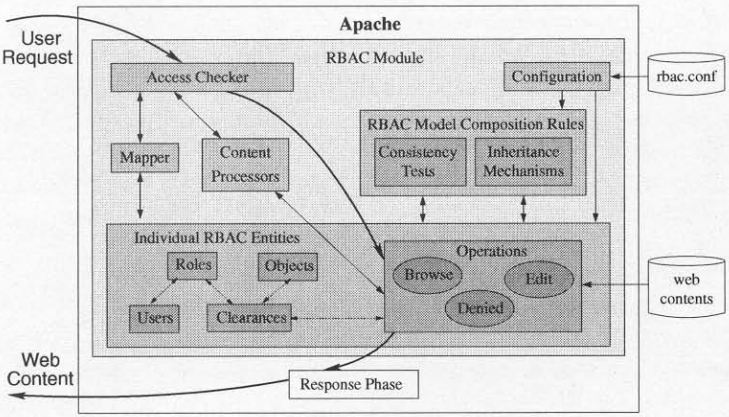

Fig. 1. *System architecture*

figure, dotted arrows represent the main request flow. The software components interactions are shown with thin arrows, and are described in Tab. 2.

The key concept in the architecture is the ability of the access checker for executing one operation for each user request. Even if the request is not allowed, the "denied" operation prepares a message with the reasons of the denial. Each operation can register its own functions for filtering the contents, so that the operation handler may use them. Although the operation is determined by the URL query, sometimes the HTTP method helps to know the phase of execution. When editing, the GET method means the user has requested to edit o, so the edition form is prepared with the fragments of o that can be edited. When the user finishes, the form is submitted, and the server executes again the "edit" operation on o, but now the method is POST, so the edition handler now reads the request body and uses other content processor in order to ensure that the incoming contents are allowed. This avoids, for example, to put a link to an object the user can't edit.

In order to enforce the security rules, the "browse" handler takes into account the dynamic nature of web. The operation provides generic support for dynamic web generation technologies (e.g. CGI, SSI, index generation), using a three phase filtering scheme that avoids the dynamic inclusion of forbidden elements not present in the original file.

5 Conclusions

The experience gained during the implementation of the module demonstrates that RBAC models can be incorporated into the normal behaviour of the web server without a considerable degradation of the level of service. The complexity of the model does not affect the response time, assuming that all properties of the different

Table 2. Main elements of RBAC system

Element	Description
Access checker	It implements θ, determining if a request is or not allowed. If request comes from the user, the proper operation is executed. The most permissive role of the user is taken as argument of θ.
Mapper	This component provides independence to the access checker by means of a mapping between RBAC identifiers and web-related concepts.
RBAC entities	Implements the concepts of object, role, user, clearance, operation and security category. Each one provides a consistent set of primitives for its manipulation: add, get, modify and get the value of some properties.
Content Processors	Operations use the content processors in order to prepare a response that respect the security rules. Objects are filtered, in order to detect and skip the elements (object or references) whose requested operation cannot be performed by the user. For example, an user must have "edit" clearance for o in order to be allowed by the edition processor to modify any link to o embedded in other objects. Content processors needs to check the access abilities of each object involved in a content, counting on the help of the mapper and access checker.
Model Composition Rules	This component ensures the correct construction of the model. It performs all propagation mechanisms among RBAC entities as well as the consistency tests.
Configuration	It reads the configuration file, creating the RBAC model. Then, the model rules are applied. This component is executed when the server starts up, so once the server is ready for accepting requests, all model properties have been computed what makes the access permission time independent of the model complexity. If the model is not well-formed, errors are stored in the Apache log file, and the server does not start up.

entities have been computed during the server startup process, so new constraints and relationships can be added in the future for improving the model.

Acknowledgements

This work is part of the MARAH project funded by the "Dirección General de Investigación de la Comunidad Autónoma de Madrid y FSE" (07T/0012/2001).

References

1. John Barkley, Anthony Cincotta, David Ferraiolo, Serban Gavrilla, and Richard Kuhn. Role based access control for the world wide web. In *20th National Computer Security Conference*, 1997.
2. Paloma Díaz, Ignacio Aedo, and Fivos Panetsos. Modelling security policies in hypermedia and web-based applications. In San Murugesan and Yoghesh Deshpande, editors, *WebEngineering: Managing diversity and complexity of web application development*, pages 90104. Springer Verlag (LNCS 2016), 2001.
3. Paloma Díaz, Ignacio Aedo, and Fivos Panetsos. Modelling the dynamic behavior of hypermedia applications. *IEEE Transactions on Software Engineering*, 27(6):550572, June 2001.
4. David Ferraiolo, John Barkley, and Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information Systems Security*, 1(2):3464, February 1999.
5. David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (rbac): Features and motivations. In *Proceedings, Annual Computer Security Applications Conference, IEEE Computer Society Press*, 1995.
6. David Ferraiolo and Richard Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554563, 1992.
7. W. A. Jansen. Inheritance properties of role hierarchies. In *21st National Information Systems Security Conference*, October 6-9 1998.
8. W. A. Jansen. A revised model for role-based access control. Technical Report NIST-IR 6192, National Institute of Standards and Technology, Gaithesburg, Maryland, July 1998.
9. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):3847, February 1996.