Role-Based Access Control on the Web

JOON S. PARK and RAVI SANDHU
George Mason University
and
GAIL-JOON AHN
University of North Carolina at Charlotte

Current approaches to access control on Web servers do not scale to enterprise-wide systems because they are mostly based on individual user identities. Hence we were motivated by the need to manage and enforce the strong and efficient RBAC access control technology in large-scale Web environments. To satisfy this requirement, we identify two different architectures for RBAC on the Web, called *user-pull* and *server-pull*. To demonstrate feasibility, we implement each architecture by integrating and extending well-known technologies such as cookies, X.509, SSL, and LDAP, providing compatibility with current Web technologies. We describe the technologies we use to implement RBAC on the Web in different architectures. Based on our experience, we also compare the tradeoffs of the different approaches .

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Design, Experimentation, Security

Additional Key Words and Phrases: Cookies, digital certificates, role-based access control, WWW security

1. INTRODUCTION

The World Wide Web (WWW) is a critical enabling technology for electronic commerce on the Internet. Its underlying protocol, HTTP (HyperText Transfer Protocol [Fielding et al. 1999]), has been widely used to synthesize diverse technologies and components, to great effect in Web environments.

Authors' addresses: J. S. Park and R. Sandhu, Laboratory for Information Security Technology (LIST), Information and Software Engineering Department, George Mason University, Mail Stop 4A4, Fairfax, VA 22030; email: jpark@itd.nrl.navy.mil; http://www.list.gmu.edu; sandhu@gmu.edu; http://www.list.gmu.edu; G.-J. Ahn, College of Information Technology, University of North Carolina at Charlotte, 9201 University City Blvd., Charlotte, NC 28223-0001; email: gahn@uncc.edu; http://www.coit.uncc.edu.

Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. However, current approaches to access control on Web servers are mostly based on individual user identity; hence they do not scale to enterprise-wide systems.

If the roles of individual users are provided securely, Web servers can trust and use the roles for role-based access control (RBAC [Sandhu et al. 1996; Sandhu 1998]). So a successful marriage of the Web and RBAC has the potential for making a considerable impact on deployment of effective enterprise-wide security in large-scale systems.

In this article we present a comprehensive approach to RBAC on the Web. We identify the *user-pull* and *server-pull* architectures and analyze their advantages and disadvantages. To support these architectures on the Web, we take relatively mature technologies and extend them for secure RBAC on the Web. In order to do so, we make use of standard technologies in use on the Web: cookies, X.509, SSL, and LDAP.

First, we investigate how to secure and use the very popular cookies technology [Kristol and Montulli 1999; Moore and Freed 1999] for RBAC on the Web. Cookies were invented to maintain continuity and state on the Web. Cookies contain strings of text characters encoding relevant information about the user, and are sent to the user's machine via the browser while the user is visiting a cookie-using Web site. The Web server gets those cookies back and retrieves the user's information from the cookies when the user later returns to the same Web site. The purpose of a cookie is to acquire information and use it in subsequent communications between the Web server and the browser, without asking for the same information again. Cookies can also be used at a different Web server from the one that issued the cookie. However, it is not safe to store and transmit sensitive information in cookies because cookies are insecure. Cookies are stored and transmitted in clear text, which is readable and can be forged easily. One contribution of this article is to identify and discuss techniques to make cookies secure, so that they can carry and store sensitive data. We call these cookies *secure cookies*. These techniques have varying degrees of security and convenience for users and system administrators. To demonstrate the feasibility of these ideas, we implement RBAC on the Web in the user-pull architecture using secure cookies.

Second, we also use X.509v3 certificates [ITU-T Recommendation X.509 1993; 1997; Housley et al. 1998], an ISO standard, since public-key infrastructure (PKI) is recognized as a crucial enabling technology for security in large-scale networks. The basic purpose of X.509 certificates is simply the binding of users to keys. Even though X.509 can be extended, the application of the extensions of X.509 for RBAC is not yet precisely defined. We describe how to extend and use existing X.509 certificates for RBAC on the Web. We call these extended X.509 certificates *smart certificates*. Smart certificates have several sophisticated features: they support short-lived lifetime and multiple certificate authorities, contain attributes, provide postdated and renewable certificates, and provide confidentiality. Selection

of these new features depends on applications. To prove the feasibility of these ideas, we implement RBAC on the Web in the user-pull architecture using smart certificates.

Third, we implement RBAC on the Web in the server-pull architecture using LDAP (Lightweight Directory Access Protocol [Howes et al. 1999]) and SSL (Secure Socket Layer [Wagner and Schneier 1996; Dierks and Allen 1999]). LDAP is a protocol that enables X.500–based directories to be read through Internet clients. When an LDAP client needs a specific entry in an LDAP server, the LDAP client generates an LDAP message containing a request and sends this message to the LDAP server. The server retrieves the entry from its database and sends it to the client in an LDAP message. With this directory services feature, we use LDAP and SSL between Web servers and the role server to implement RBAC on the Web in the server-pull architecture.

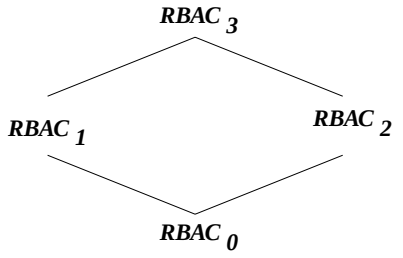
Secure cookies inherently support the user-pull architecture only—since cookies are stored in users' machines they cannot operate in the server-pull architecture. In contrast, smart certificates and LDAP support both user-pull and server-pull architectures.

This article is organized as follows. Section 2 introduces an overview of role-based access control (RBAC). In Section 3, we identify operational architectures for RBAC services on the Web. Section 4 describes how we render secure cookies using cryptographic technologies and implement RBAC on the Web in the user-pull architecture using secure cookies. Section 5 describes how we extend X.509 certificates with new features and implement RBAC on the Web in the user-pull architecture using smart certificates. Section 6 describes how we implement RBAC on the Web in the server-pull architecture using LDAP and SSL between Web servers and the role server. In Section 7, we discuss the tradeoffs among the different technologies we have developed and implemented on the Web. In Section 8, we compare our approaches with existing RBAC products. Finally, Section 9 gives our conclusions.

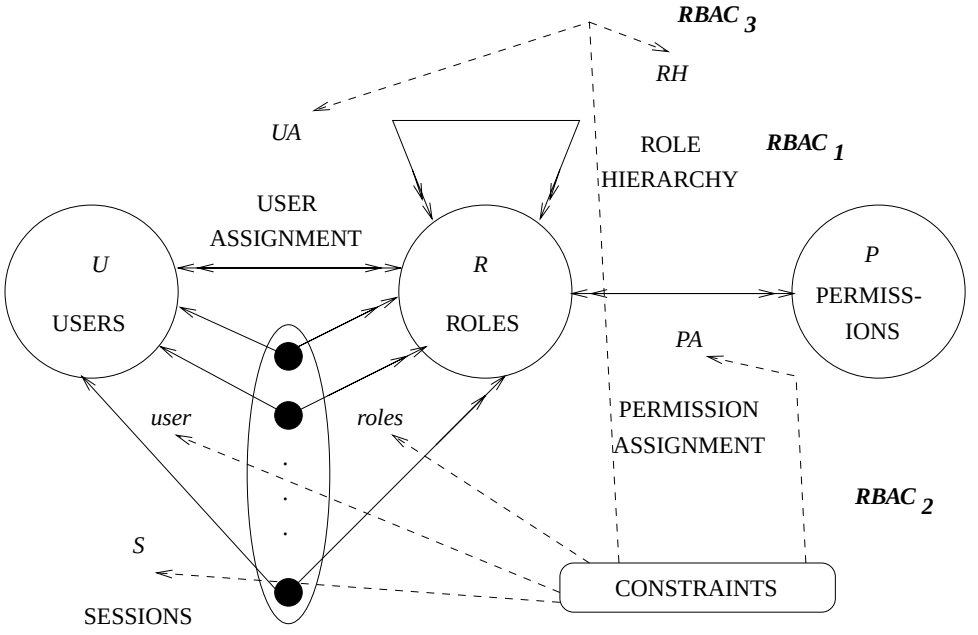
2. ROLE-BASED ACCESS CONTROL (RBAC) OVERVIEW

Role-based access control (RBAC) emerged rapidly in the 1990s as a proven technology for managing and enforcing security in large-scale enterprise-wide systems. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management. A significant body of research on RBAC models and experimental implementations has developed [Ferraiolo and Kuhn 1992; Ferraiolo et al. 1995; Guiri 1995; Guiri and Iglio 1996; Mohammed and Dilts 1994; Hu et al. 1995; Nyanchama and Osborn 1995; Sandhu et al. 1996; von Solms and van der Merwe 1994; Youman et al. 1997; Sandhu 1998; Ahn and Sandhu 2000; Osborn et al. 2000].

We were motivated by the need to manage and enforce the strong and efficient access control technology of RBAC in a large-scale Web environment,



(a) Relationship among RBAC models



(b) RBAC models

Fig. 1. A family of RBAC models.

since RBAC is a successful technology that will be a central component of emerging enterprise security infrastructures.

RBAC is a proven alternative to traditional discretionary and mandatory access controls; it ensures that only authorized users are given access to certain data or resources. It also supports three well-known security principles: information hiding, least-privilege, and separation of duties.

A role is a semantic construct forming the basis of access control policy. With RBAC, system administrators can create roles, grant permissions to those roles, and then assign users to the roles on the basis of their specific

job responsibilities and policy. In particular, role-permission relationships can be predefined, making it simple to assign users to the predefined roles. Without RBAC, it is difficult to determine what permissions have been authorized for which users.

Access control policy is embodied in RBAC components such as user-role, role-permission, and role-role relationships. These RBAC components determine if a particular user is allowed access to a specific piece of system data.

Users create sessions during which they may activate a subset of roles to which they belong. Each session can be assigned to many roles, but it maps only one user. The concept of a session corresponds to the traditional notion of subject in the access control literature.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility. By convention, junior roles appear at the bottom of the hierarchic role diagrams and senior roles at the top. The hierarchic diagrams are partial orders, so they are reflexive, transitive, and antisymmetric. Inheritance is reflexive because a role inherits its own permissions, transitive because of a natural requirement in this context, and antisymmetry rules out roles that inherit from one another, and would therefore be redundant.

Constraints are an effective mechanism to establish higher-level organizational policy. They can apply to any relation and function in an RBAC model. When applied, constraints are predicates that return a value of *acceptable* or *not acceptable*. A general family of RBAC models was defined by Sandhu et al. [1996]. Figure 1 shows the most general model in this family.

RBAC0 is the base model that specifies the minimum requirement for any system that fully supports RBAC. RBAC1 and RBAC2 both include RBAC0, but they also have independent features. RBAC1 adds the concept of role hierarchies, which imply situations in which roles can inherit permissions from other roles. RBAC2 adds constraints that impose restrictions on components of RBAC. RBAC1 is incomparable with RBAC2, and vice versa. RBAC3 is the consolidated model that includes RBAC1 and RBAC2 and, by transitivity, RBAC0. The relationship among the four RBAC models and the consolidated RBAC3 model is shown in Figure 1.

Details for motivation and discussion on the RBAC family of models (RBAC0, RBAC1, RBAC2, RBAC3, ARBAC0, ARBAC1, ARBAC2, and ARBAC3) are described in Sandhu et al. [1996]; Sandhu [1997]; Sandhu et al. [1999].

3. OPERATIONAL ARCHITECTURES

Park and Sandhu identified two different approaches for obtaining a user's attributes on the Web, especially with respect to user-pull and server-pull architectures, in which each architecture has user-based and host-based modes [Park and Sandhu 1999b]. An attribute is a particular property of an entity, such as a role, access identity, group, or clearance. In this section,

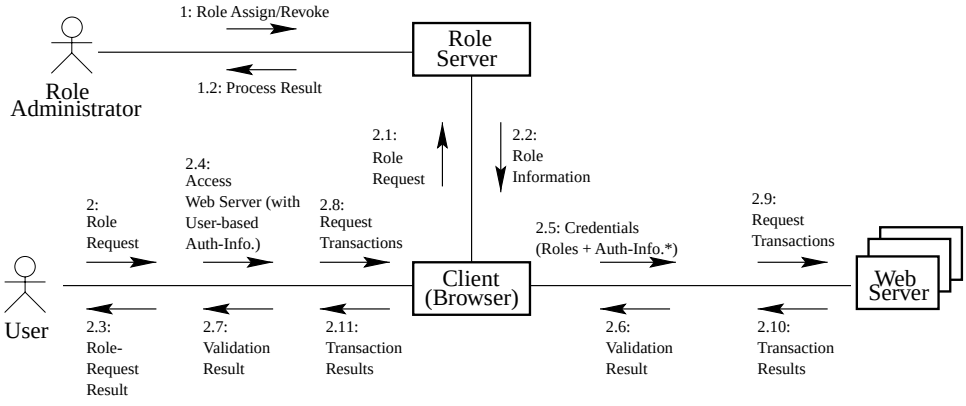


Fig. 2. Collaborational diagram for the user-pull architecture.

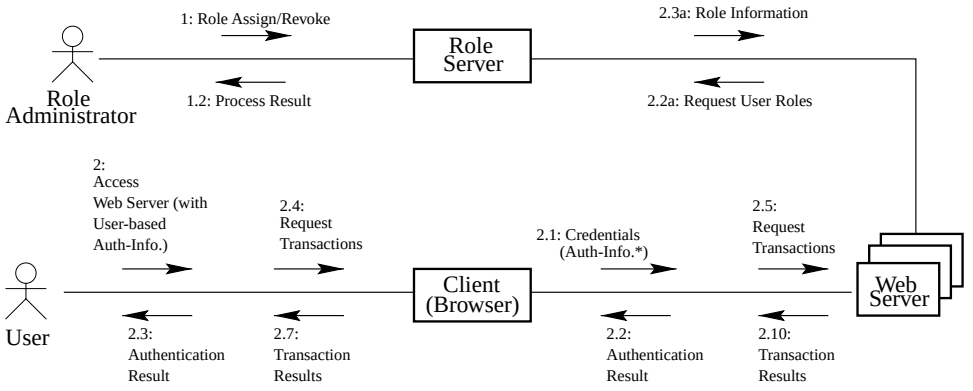
we embody those general approaches for RBAC on the Web with specific components and relationships. Each approach is implemented and described in this article, and we provide an analysis of their relative advantages and disadvantages. Basically, there are three components in both architectures: client, Web server, and role server. These components are already being used on the Web. Clients connect to Web servers via HTTP using browsers. The role server is maintained by an administrator and assigns users to the roles in the domain [Sandhu and Park 1998]. Detailed technologies (such as authentication, role transfer and protection, and verification) to support these architectures depend on the applications that are used.

3.1 User-Pull Architecture

In user-pull architecture, a user, say Alice, pulls her roles from the role server and then presents them to the Web servers, as depicted in the UML (Unified Modeling Language [Booch et al. 1998]) collaborational diagram in Figure 2. We call this a user-pull architecture, since the user pulls her roles from the role server where roles are assigned to the users in the domain. HTTP is employed for user-server interaction with standard Web browsers and Web servers.

In user-pull-host-based mode, the user needs to download her roles from the role server and store them in her machine (which has her host-based authentication information, such as IP numbers).[1] Later, when Alice wants

[1]Address-based authentication is a convenient authentication mechanism because the authentication process is transparent to users, but such a method is not always desirable. For example, if the user's IP address is dynamically assigned to her computer whenever she connects to the Internet, or the user's domain uses a proxy server, which provides the same IP numbers to the users in the domain, this is not a proper authentication technique. In addition, we cannot avoid IP spoofing, which is a technique for gaining unauthorized access by sending messages to a computer with a trusted IP address.



*Authentication Information can be either user-based or host-based.

Fig. 3. Collaborational diagram for server-pull architecture.

to access the Web server, which requires proper authentication information and roles, her machine presents that information to the Web server. After client authentication and role verification, the Web server uses the roles for RBAC. However, since this mode is host-based, it cannot support high user mobility, while it may support a more convenient service than the user-based mode, which requires the user's cooperation (e.g., typing in passwords).

On the other hand, the user-pull-user-based mode supports high user mobility. The user can download her roles to her current machine from the role server. Then, she presents those roles to the Web server along with her user-based authentication information, such as her passwords. After user authentication and role verification, the Web server uses the roles for RBAC. In this mode, the user can use any machine that supports HTTP, as long as she has the right user-based authentication information (e.g., passwords).

In this user-pull architecture, we must support the binding of roles and identification for each user. For instance, if Alice presents Bob's roles with her authentication information to the Web server, she must be rejected. In Section 5.2 we describe how to solve this problem efficiently by means of smart certificates between existing Web servers and browsers. General approaches for binding user attributes and their identities are discussed by Park and Sandhu [2000a].

3.2 Server-Pull Architecture

In server-pull architecture, each Web server pulls user's roles from the role server as needed and uses them for RBAC, as depicted in the UML collaborational diagram in Figure 3. We call this a server-pull architecture, since the server pulls the user's roles from the role server. HTTP is used for user-server interaction with standard Web browsers and servers. If the role server provides the user's roles securely, the Web server can trust those roles and uses them for RBAC.

In this architecture the user does not need access to her roles; she needs only her authentication information. In server-pull-host-based mode, she presents host-based authentication information (e.g., IP numbers) to the Web server. The role-obtaining mechanism is transparent to the user, while limiting user portability. However, in server-pull-user-based mode, Alice presents user-based authentication information (e.g., passwords) to the Web server. This supports high user portability, while it requires the user's cooperation (e.g., typing in passwords). After user authentication, the Web server downloads the user's roles from the role server and uses them for RBAC.

4. RBAC ON THE WEB IN USER-PULL ARCHITECTURE USING SECURE COOKIES

Cookies were invented to maintain continuity and state on the Web [Kristol and Montulli 1999; Moore and Freed 1999]. The purpose of a cookie is to acquire information and use it in subsequent communications between the Web server and the browser without asking for the same information again. Technically, it is not difficult to make a cookie carry relevant information. However, because they are insecure it is not safe to store and transmit sensitive information in cookies. Cookies are stored and transmitted in clear text, which is readable and easily forged. Hence, we should render cookies secure in order to carry and store sensitive data in them.

We provide *secure cookies* with three types of security services: authentication, integrity, and confidentiality. Authentication services verify the owner of the cookies. Integrity services protect cookies against the threat that their contents might be changed by unauthorized modification. Finally, confidentiality services protect cookies against having their values revealed to an unauthorized entity. Details for these techniques have varying degrees of security and convenience for users and system administrators. Our motivation for using the cookie mechanism is that it is already widely deployed in existing Web browsers and servers for maintaining state on the Web. There are other techniques to make Web transactions secure without using secure cookies. For example, the secure HTTP protocol (S-HTTP) and HTML security extensions [Rescorla and Schiffman 1998; Schiffman and Rescorla 1998] can be used for this purpose. Other protocols and extensions could be devised to operate in conjunction with the SSL protocol. However, these technologies cannot solve the stateless problem of HTTP. Furthermore, none of them can prevent end-system threats (described in Section 4.2) to cookies.

In this section we describe how we developed secure cookies and implemented RBAC with role hierarchies [Ferraiolo et al. 1995 ; Sandhu et al. 1996] on the Web in user-pull architecture using secure cookies.

4.1 Related Technologies

4.1.1 *Cookies*. Cookies serve many purposes on the Web, such as selecting display mode (for example, frames or text only), maintaining shopping-cart selections, and storing user identification data.

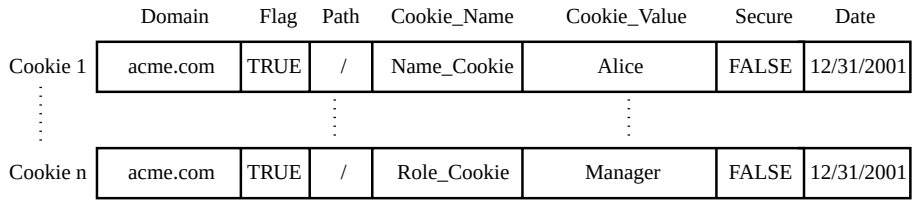


Fig. 4. An example of cookies on the Web.

All cookies are fundamentally similar. A typical cookie, shown in Figure 4, has several fields. Cookie_Name and Cookie_Value contain information a Web site would want to keep. For example, in the figure, the values of Name_Cookie and Role_Cookie are "Alice" and "Manager," respectively. Date is the cookie's valid lifetime. Domain is a host or domain name where the cookie is valid. Flag specifies whether or not all machines within a given domain can access the cookie's information. Path restricts cookie usage within a site (only pages in the path can read the cookie). If the Secure flag is on, the cookie will be transmitted over secure communications channels only, such as SSL. Detailed cookie specifications are available in Kristol and Montulli [1999] and Moore and Freed [1999].

According to the current HTTP state management mechanism, whenever a browser requests a URL to a Web server, it sends only the relevant Cookie_Name and Cookie_Value fields (selected by the Domain and Flag fields) to the server. Cookies received by the server are used during this browser-server communication. If the server does not receive any cookies, however, it either works without using cookies or it creates new ones for subsequent browser-server communication.

A Web server can update the cookies' contents whenever the user visits the server. The cookie issuer is not important for validation; any Web server can issue cookies for other Web servers.

4.1.2 *Pretty Good Privacy (PGP).* PGP (Pretty Good Privacy [Zimmermann 1995; Garfinkel 1995]), a popular software package originally developed by P. Zimmermann, is widely used by the Internet community to provide cryptographic routines for email, file transfer, and file storage applications. A proposed Internet standard has been developed [Callas et al. 1998], specifying use of PGP. It employs existing cryptographic algorithms and protocols and runs on multiple platforms. It provides data encryption and digital signature functions for basic message protection services.

PGP is based on public-key cryptography, and defines its own public-key pair-management system and public-key certificates. The PGP key-management system is based on the relationship between key owners, rather than on a single infrastructure such as X.509. Basically, it uses RSA [Rivest et al. 1978] for the convenience of the public-key cryptosystem, message digests (MD5 [Rivest 1992]), and IDEA [Lai and Massey 1991] for process speed, and Diffie-Hellman [Diffie and Hellman 1997] for key

exchange. The updated version supports additional cryptographic algorithms.

Although the original purpose of PGP was to protect casual email among Internet users, we decided in our implementation to use the PGP package for secure cookies.

4.2 Security Threats to Typical Cookies

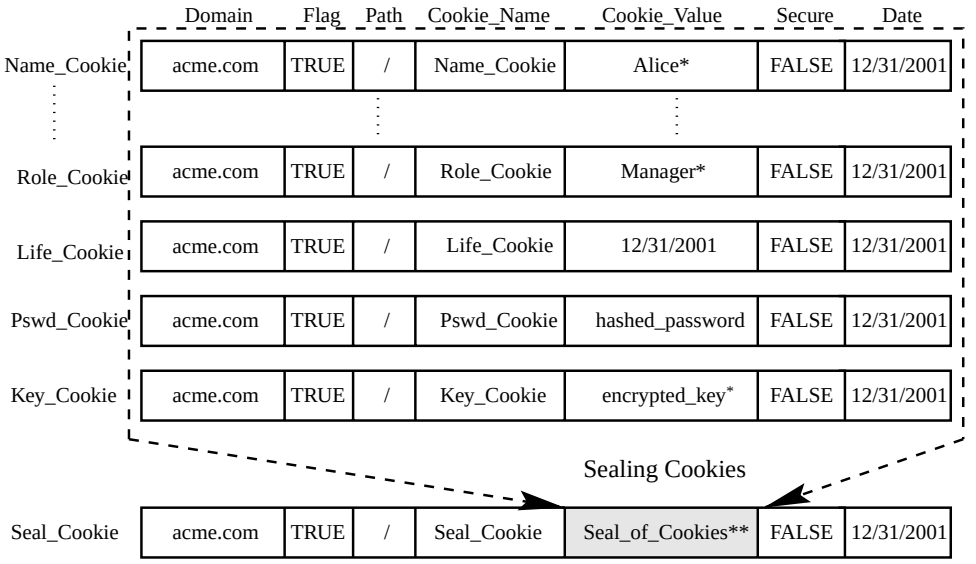
We distinguish three types of threats to cookies: *network security threats*, *end-system threats* and *cookie-harvesting threats*. Cookies transmitted in clear text on the network are susceptible to snooping (for subsequent replay) and to modification by network threats. Network threats can be foiled by using the Secure Sockets Layer (SSL) protocol, which is widely deployed in servers and browsers. However, SSL can only secure cookies while they are on the network. Once the cookie is in the browser's end system, it resides on the hard disk or memory in clear text. It is trivial to alter such cookies, and they are easily copied from one computer to another, with or without the connivance of the user on whose machine the cookies were originally stored. We call this the end-system threat. The ability to alter cookies allows users to forge authorization information in cookies and to impersonate other users. The ability to copy cookies makes such forgery and impersonation all the easier. Additionally, if an attacker collects cookies by impersonating a site that accepts cookies from the users (who believe that they are communicating with a legitimate Web server), the attacker can later use those harvested cookies for all other sites that accept them. We call this the cookie-harvesting threat. These attacks are all relatively easy to carry out, and certainly do not require great hacker expertise.

4.3 Designing Secure Cookies

In this section we describe how to transform regular cookies—which have zero security—into secure cookies, which provide the classic security services against the three types of threats to cookies (described in Section 4.2). Details for secure cookies and their applications are described in Park and Sandhu [2000b].

Basically, secure cookies provide three types of security services: *authentication*, *integrity*, and *confidentiality services*. Selection of the kinds and contents of secure cookies depends on applications and the given situation.

Figure 5 shows a set of secure cookies that we will create and use for RBAC on the Web. The Name_Cookie contains the user's name (e.g., Alice) and the Role_Cookie holds the user's role information (e.g., Manager). The Life_Cookie is used to hold the lifetime of the secure-cookie set in its Cookie_Value field and enables the Web server to check the integrity of the lifetime of the secure-cookie set. To protect these cookies from possible attacks, we use IP_Cookie, Pswd_Cookie, and Seal_Cookie. Authentication cookies (i.e., IP_Cookie and Pswd_Cookie) verify the owner of the cookies by comparing the authentication information in the cookies to those coming



* Sensitive fields are encrypted in the cookies.
** Seal_of_Cookies can be either MAC or a signed message digest of cookies.
Note: Pswd_Cookie can be replaced with one of the other authentication cookies in Figure 2.

Fig. 5. A set of secure cookies for RBAC on the Web.

from the users. The IP_Cookie holds the IP number of the user's machine, and the Pswd_Cookie holds the user's encrypted passwords. This confidentiality service protects the values of the cookies from being revealed to any unauthorized entity. In our implementation, we use the IP_Cookie and Pswd_Cookie together to show feasibility, but only one of these authentication cookies can be used to provide the authentication service. The choice of an authentication cookie depends on the situation.[2] Finally, the Seal_Cookie—which has the digital signature or MAC (Message Authentication Code [Bellare et al. 1996]) of the cookie-issuing server on the secure cookie set—supports integrity service, protecting cookies against the threat that their contents might be changed by unauthorized modification.

There are basically two cryptographic technologies applicable for secure cookies: public-key-based and secret-key-based solutions. In our implementation, we use the public-key-based solution for security services provided by a PGP package via CGI (Common Gateway Interface) scripts. In the next section we will describe, in turn, secure cookie creation, verification, and use of role information in the Role_Cookie for RBAC with role hierarchies. A detailed description for this implementation is available in Park et al. [1999].

[2]It is also possible for authentication to be based on protocols such as RADIUS [Rigney et al. 1997], Kerberos [Steiner et al. 1988; Neuman 1994], and other, similar protocols. Our focus in this work is on techniques that make secure cookies self-sufficient, rather than partly reliant on other security protocols, which is always possible.

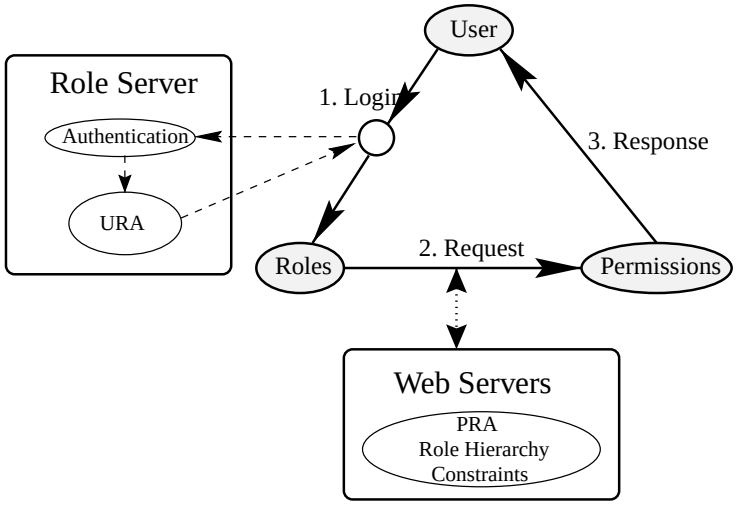


Fig. 6. A schematic of RBAC on the Web.

4.4 RBAC Implementation by Secure Cookies

Figure 6 shows a schematic of RBAC on the Web. The role server has URA (user-role assignment) information for the domain. After a successful user authentication, the user receives his or her assigned roles in the domain from the role server. Later, when the user requests access to a Web server with the assigned roles in the domain, the Web server allows the user to execute transactions based on the user's roles instead of his or her identity. The Web servers may have role hierarchies, PRA (permission-role assignment) information, or constraints based on their policies.

But how can the Web servers trust the role information presented by users? For instance, a malicious user may have unauthorized access to the Web servers by using forged role information. So we must protect the role information from being forged by any possible attacks on the Web as well as in the end-systems.

There can be many possible ways to support the above requirement. In this section, as one possible solution, we describe how to protect the role information from possible threats using secure cookies, and how we implemented RBAC with role hierarchy on the Web. Although we use PGP-based public-key cryptography, it is always possible to use other cryptographic technologies with secure cookies. Figure 7 shows how secure cookies (including a Role_Cookie) for RBAC are created and used on the Web. If a user, let's say Alice, wants to execute transactions in the Web servers in a RBAC-compliant domain, she first connects to the role server in the beginning of the session. After the role server authenticates Alice, it finds Alice's explicitly assigned roles in the URA database and creates a set of secure cookies. Those secure cookies are then sent to and stored securely in Alice's hard drive, so that Alice does not need to go back to the role server to get her assigned roles until the cookies expire. She can use the roles in

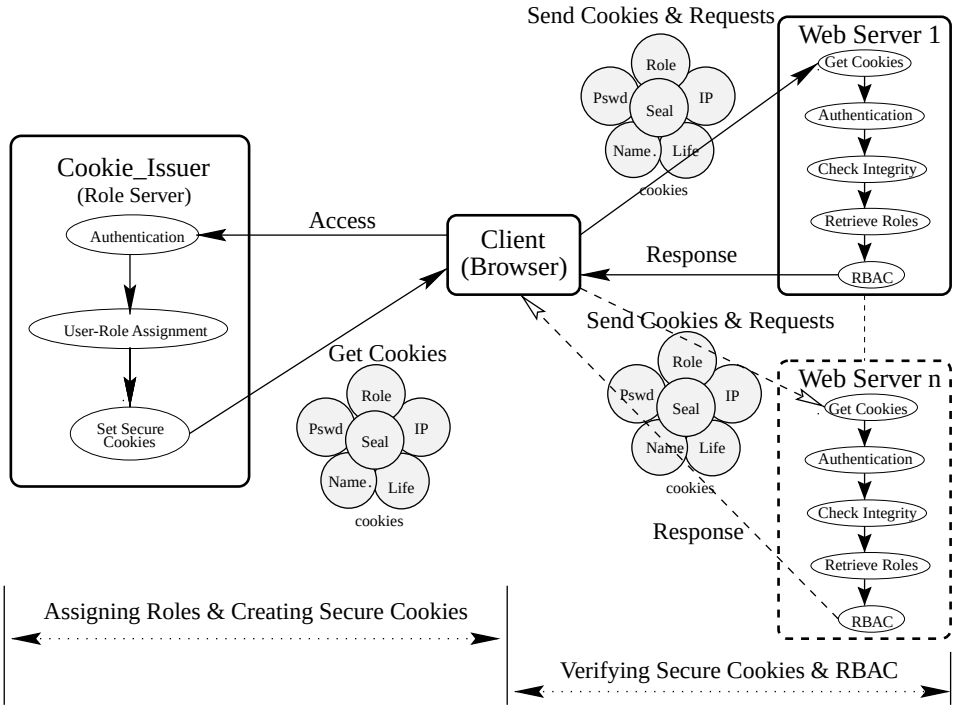


Fig. 7. RBAC on the Web by means of secure cookies.

her Role_Cookie securely in the RBAC-compliant domain as long as the cookies are valid.

When Alice requests access to a Web server—which has PRA information—by typing the server URL in her browser, the browser sends the corresponding set of secure cookies to the Web server. The Web server authenticates the owner of the cookies by using the authentication cookies, such as IP_Cookie and Pswd_Cookie, comparing the values in the cookies with the values from the user. Finally, the Web server checks the integrity of the cookies by verifying the role server's digital signature in the Seal_Cookie using the role server's public key. If all the cookies are valid and successfully verified, the Web server trusts the role information in the Role_Cookie and uses it for RBAC with role hierarchy and permission-role assignment information in the Web server.

4.4.1 *Creating Secure Cookies* . Figure 8 is a UML (Unified Modeling Language) collaborational diagram for secure cookie creation. This diagram shows how we created a set of secure cookies for our implementation (see left side of Figure 7).

When a user, Alice, connects to the role server (which supports HTTP) of the domain with her Web browser, she is prompted by the HTML form to type in her user ID and passwords for the domain. The "set-cookie.cgi" program first retrieves the user ID and passwords and the IP number of

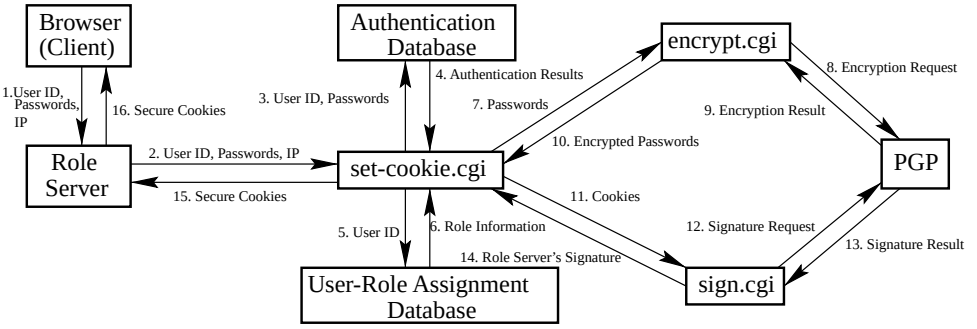


Fig. 8. Creating secure cookies.

the client machine. The program authenticates the user by comparing the user ID and passwords with the ones in the authentication database.[3] It then assigns the user to roles by matching the user ID and the corresponding roles from the URA (user-role assignment) database.

Subsequently, a subroutine for encryption is called to another CGI program (encrypt.cgi), which uses PGP to encrypt the passwords by the cookie-verifying Web server's public key. These encrypted passwords will be stored in the Pswd_Cookie by the "set-cookie.cgi" program. The "set-cookie.cgi" program then creates IP_Cookie, Pswd_Cookie, Name_Cookie, Life_Cookie, and Role_Cookie, giving each cookie the corresponding value: IP numbers of the client machine, encrypted passwords, user's name, lifetime of the cookie set, and assigned roles.

To support the cookies' integrity service, the "set-cookie.cgi" program calls another CGI program (sign.cgi), which uses PGP to sign the cookies with the role server's private key. The "set-cookie.cgi" then creates the Seal_Cookie, which includes the digital signature of the role server on the contents of the cookies.

Finally, the role server sends the HTTP response header, along with the set of secure cookies, back to the user's browser, where the cookies are then stored until they expire. These secure cookies will be verified and used in the Web servers as described in Section 4.4.2. Figure 9 is an actual snapshot of a set of secure cookies from our implementation stored in the user's machine after they have been generated by the role server. The contents of the cookies reflect the ones in Figure 5 exactly.

4.4.2 *Secure Cookie Verification*. Figure 10 is a UML collaborational diagram for secure cookie verification. This diagram shows how we verified (corresponding to the right side of Figure 7) the set of secure cookies that we generated in Section 4.4.1 for our implementation. When Alice connects to a Web server (which accepts the secure cookies) in an RBAC-compliant domain, the connection is redirected to the "index.cgi" program. The related secure cookies are sent to the Web server and Alice is prompted by the

[3]If the user already has an authentication cookie in a set of secure cookies, Web servers can use the authentication cookie for user authentication instead of authentication databases.

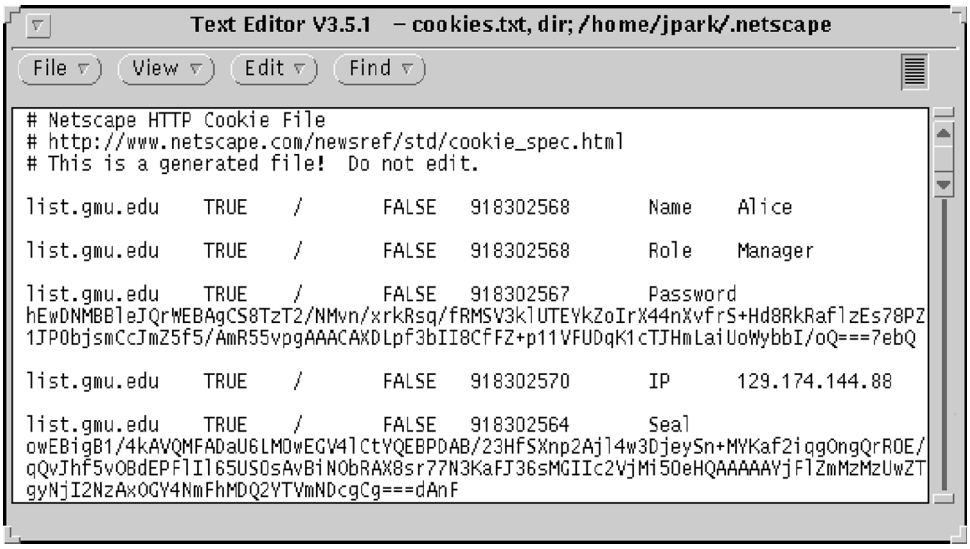


Fig. 9. An example of secure cookies stored in a user's machine.

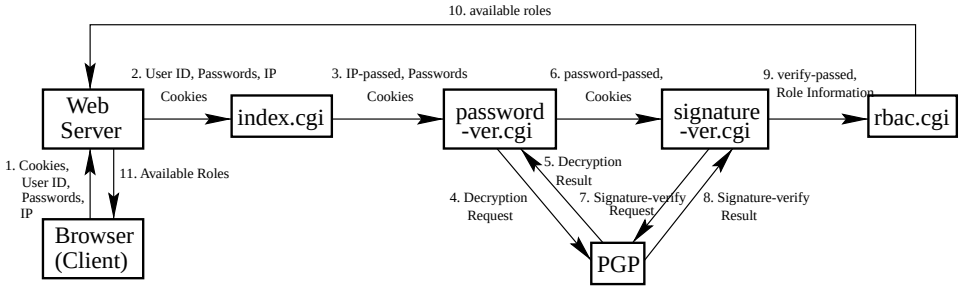


Fig. 10. Verifying secure cookies.

HTML form to type in her user ID and passwords. The "index.cgi" program checks the validity of all the cookies. The two IP addresses, one from the IP cookie and the other from the environment variable, REMOTE_ADDR, are compared. If they are identical, then the host-based authentication is passed and a hidden field[4] "status" with the "IP-passed" value is created to indicate that this stage was passed. However, if the IP numbers are different, the user is rejected by the server.

When the user submits her user ID and passwords to the server, the Web server translates the request headers into environment variables, and another CGI program, "password-ver.cgi," is executed. The first thing the "password-ver.cgi" does is to check the hidden field "status" to see if the

[4]We used a hidden field to check the completion of the previous stage, which is passed on to the next program. The hidden field protects the pages from being accessed directly (skipping required verification steps) by a malicious user. For example, without this hidden field, a malicious user could access the pages directly with forged cookies.

previous stage was successfully completed. If this is "IP-passed," the program decrypts the value of the Pswd_Cookie (encrypted user password) using the PGP with the Web server's private key, since it was encrypted with the Web server's public key by the role server. The program (password-ver.cgi) then compares the two passwords: one from the user and the other decrypted from the Pswd_Cookie. If they are identical, then the user-based authentication is passed, and a hidden field "status" with the value of "password-passed" is created to indicate that this stage was passed. However, if the two passwords are different, the user has to start again by either retyping the passwords or receiving new cookies from the role server.

After password verification is completed, another CGI program, "signature-ver.cgi," is activated to check the integrity of the cookies. Like the other programs, it first checks the value of "status" passed on from the previous program, and proceeds only if it is shown that the user has been through the password verification stage. If the value is "password-passed," then the program verifies the signature in the Seal_Cookie with the role server's public key using PGP. If the integrity is verified, it means that the cookies have not been altered, and a hidden field "status" with the value "verify-passed" is created to indicate that this stage was passed and forwarded to the final program, "rbac.cgi." This program uses the role information in the Role_Cookie for RBAC in the Web server as described in Section 4.4.3. However, if the signature verification fails, the user has to start again by receiving new cookies from the role server.

4.4.3 *RBAC in the Web Server*. After verifying all the secure cookies, the Web server allows the user, Alice, to execute transactions based on her roles, contained in the Role_Cookie, instead of her identity. In other words, for authorization purposes, the Web server does not care about the user's identity. This resolves the scalability problem of identity-based access control, which is widely used in existing Web servers. Furthermore, the Web server can also use a role hierarchy, which supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy different from that in other servers. In our implementation, we used a role hierarchy in the Web server, depicted in Figure 11.

If the "rbac.cgi" program in Figure 10 receives the value "verify-passed" from the previous verification step, it means that the cookies have successfully passed all the verification stages, such as IP, passwords, and signature verification. Hence the Web server can trust the role information in the Role_Cookie and use it for RBAC in the server.

How then can the Web server protect the pages from being accessed by unauthorized users? Suppose a malicious user, Bob, has the role PE1 but wishes to access pages that require the PL1 role (see Figure 11). He could change the value of his Role_Cookie so that it has PL1, or roles senior to PL1. He would go through the password verification stages, since he would be able to log in as Bob by using his own passwords. However, a problem

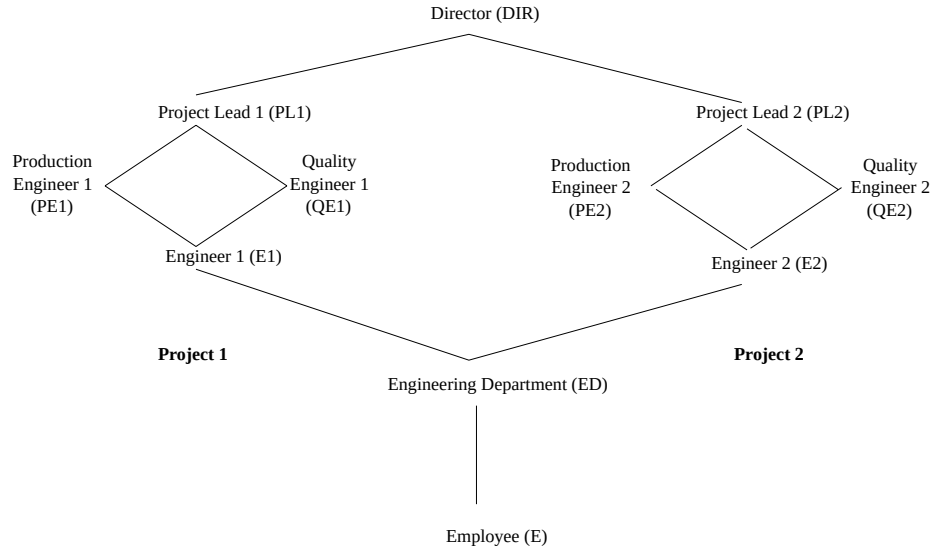


Fig. 11. An example role hierarchy.

will arise while his Seal_Cookie is being verified, as the signature verification will fail. He will not be allowed to move beyond this stage. On the other hand, he could try accessing the pages directly by typing the URLs. This is not allowed, since each page checks to see if he has activated the required role, PL1, or roles senior to PL1. In other words, Bob is not allowed to access the pages that require roles senior to his because he cannot activate the senior roles that are out of his available role range.

As a result, the Web server allows only users who have gone through all the verification steps with the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, Seal_Cookie) to access the pages. Also, this access is possible only if the users have the required roles and activate them among their available roles based on the role hierarchy.

4.5 Summary

In this section we describe how we implemented RBAC with role hierarchies on the Web using *secure cookies* in the user-pull architecture. To protect role information in the cookies, we provided security services such as authentication, confidentiality, and integrity to the cookies using PGP and CGI scripts in the Web servers. Although we used PGP-based public-key cryptography to protect cookies on the Web in this implementation, it is always possible to use other cryptographic technologies with secure cookies. The use of secure cookies is a transparent process to users, and is applicable to existing Web servers and browsers.

5. RBAC ON THE WEB IN USER-PULL ARCHITECTURE USING SMART CERTIFICATES

Public-key infrastructure (PKI) is recognized as a crucial enabling technology for security in large-scale networks. To support PKI, X.509 certificates have been widely used [Housley et al. 1998 ; ITU-T Recommendation X.509 1993; 1997]. The basic purpose of X.509 certificates is simply the binding of users to keys. Hence we developed *smart certificates* by extending X.509 with several sophisticated features for secure attribute services and introduced their possible applications on the Web [Park and Sandhu 1999b].

In this section we give an overview of smart certificates and describe an implementation of RBAC with role hierarchies on the Web in user-pull architecture using smart certificates. In this implementation, we used a Netscape Certificate server to issue smart certificates and a Microsoft IIS 4.0 on a Windows NT platform to support RBAC on the Web. However, this approach is also possible using other certificate servers or Web servers on different platforms by proper configuration. To maintain compatibility with existing technologies, such as SSL, we used a bundled (subject's identity and role information) smart certificate [Park and Sandhu 2000a]). A detailed description of an RBAC implementation using smart certificates in user-pull architecture is available in Park and Sandhu [1999a]; Ahn et al. [2000].

5.1 Related Technologies

5.1.1 *Public-Key Certificate (X.509).* A public-key certificate [Housley et al. 1998; ITU-T Recommendation X.509 1993; 1997] is digitally signed by a certificate authority (CA) to confirm that the identity or other information in the certificate belongs to the holder (subject) of the corresponding private key. If a message sender wishes to use public-key technology to encrypt a message to a recipient, the sender needs a copy of the recipient's public key. In contrast, when a party wishes to verify a digital signature generated by another party, the verifying party needs a copy of the public key of the signing party. Both the encrypting message sender and the digital signature verifier use the public keys of other parties. Confidentiality, which keeps the value of a public key secret, is not important to the service. However, integrity is critical, as it assures public-key users that the public key used is the correct one for the other party. For instance, if an attacker is able to substitute his or her public key for the valid one, encrypted messages can be disclosed to the attacker and a digital signature can be forged by the attacker.

ITU (International Telecommunication Union) and ISO (International Organization for Standardization) published the X.509 standard in 1988, which has been adopted by IETF (International Engineering Task Force). X.509 is the most widely used data format for public-key certificates today, and is based on the use of designated certificate authorities (CAs) that verify that the entity is the holder of a certain public key by signing public-key certificates. An X.509 certificate has been used to bind a public

Certificate Content:

```
Certificate:
    Data:
        Version: v3 (0x2)
        Serial Number: 5 (0x5)
        Signature Algorithm: PKCS #1 MD5 With RSA Encryption
        Issuer: CN=data.list.gmu.edu, OU=LIST, O=GMU, C=US
        Validity:
            Not Before: Tue Feb 09 03:10:38 1999
            Not  After: Wed Feb 09 03:10:38 2000
        Subject: CN=admin.list.gmu.edu, OU=LIST, O=GMU, C=US
        Subject Public Key Info:
            Algorithm: PKCS #1 RSA Encryption
            Public Key:
                Modulus:
                    00:bc:d7:fc:4f:29:a4:29:a5:21:be:69:47:4d:55:db:37:50:
                    18:2b:6e:3e:b0:85:3e:0f:86:0f:be:58:2b:c9:d3:dc:bc:03:
                    bc:86:44:c4:f4:18:94:51:96:c6:f9:c5:db:b8:9d:88:5b:53:
                    b7:08:2f:86:64:cb:c2:7b:60:36:87
                Public Exponent: 65537 (0x10001)
        Extensions:
            Identifier: Certificate Type
                Critical: no
                Certified Usage:
                    SSL Client
            Identifier: Authority Key Identifier
                Critical: no
                Key Identifier:
                    a5:d7:08:bc:ff:07:bd:5a:d4:8d:d4:68:53:87:4b:af:81:90:
                    f0:4d
    Signature:
        Algorithm: PKCS #1 MD5 With RSA Encryption
        Signature:
            11:ca:b1:94:14:fb:67:a2:ad:90:f1:ee:88:24:a8:d3:fd:5c:75:34:fc:
            c1:68:23:e6:12:19:3a:5c:45:62:af:51:a0:2f:44:96:f8:2e:1f:75:9a:
            4b:9c:ed:2a:45:2e:db:c8:9c:56:1a:e1:75:0a:8e:bf:f8:44:b8:84:31:
            d8
```

Fig. 12. An example of X.509.

key to a particular individual or entity, and it is digitally signed by the issuer of the certificate (certificate authority) that has confirmed the binding between the public key and the holder (subject) of the certificate. An X.509 certificate consists of the following (see Figure 12):

—Version of a certificate format:

—certificate serial number;

—subject's X.500 name (assigned by a naming authority);

—subject's public key and algorithm information;

—validity period (beginning and end dates);

—issuer's X.500 name (certificate authority);

—optional fields to provide unique identifiers for subject and issuer (Version 2);

—extensions (Version 3);

—digital signature of the certificate authority.

In the event the same name over time has been reassigned to different entities, the optional fields are available from Version 2 to make the subject

name or the issuing certificate authority name unambiguous. Version 3 provides the extension field for incorporating any number of additional fields into the certificate. These extensions make X.509v3 a truly open-ended standard, with room to support diverse needs. It is possible for interested certificate issuers to define their own extension types and use them to satisfy their own particular needs.

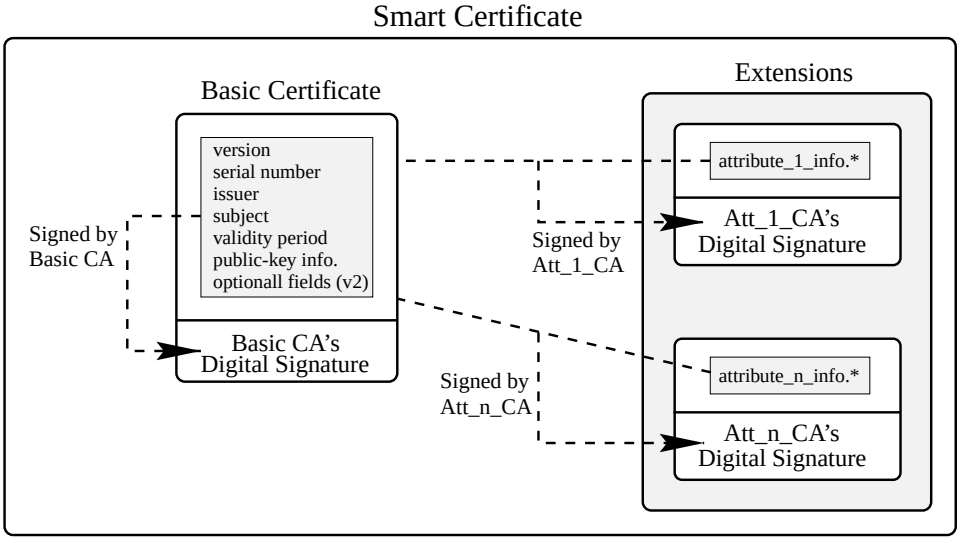
5.1.2 *Attribute Certificate*. The US financial industry through the ANSI X9 Committee developed attribute certificates [Farrell 1998a; 1998b], which have now been incorporated into both the ANSI X9.57 standard and X.509. An attribute certificate binds attribute information to the certificate's subject. Anyone can define and register attribute types and use them for his or her purposes. The certificate is digitally signed and issued by an attribute authority. Furthermore, an attribute certificate is managed in the same way as an X.509 certificate. However, an attribute certificate does not contain a public key. Therefore, an attribute certificate needs to be used in conjunction with authentication services, such as another certificate (X.509) and SSL, to verify the subject of the attribute. However, as we discussed in Section 5.2, our smart certificates have the ability to build attribute information into X.509v3 extensions, without losing effective maintenance, rather than putting this information into separate certificates.

5.1.3 *SPKI (Simple Public Key Infrastructure)*. The SPKI Working Group in IETF [Ellison et al. 1999] developed a standard form for digital certificates, focusing on authorization rather than authentication. A SPKI certificate grants specific authorization to a public key, without necessarily requiring identity of the holder of the corresponding private key. The public key can be used as a unique identifier for the key holder. Furthermore, a collision-free hash of the public key can also be used as a unique identifier for the key holder. SPKI provides simplicity, using a less rich data-encoding scheme than the ASN.1 notation used in X.509.

5.1.4 *Secure Socket Layer (SSL)*. SSL was introduced with the Netscape Navigator browser in 1994, and rapidly became the predominant security protocol on the Web [Wagner and Schneier 1996; Dierks and Allen 1999]. Since the protocol operates at the transport layer, any program that uses TCP (Transmission Control Protocol) is ready to use SSL connections. The SSL protocol provides a secure means for establishing an encrypted communication between Web servers and browsers. SSL also supports the authentication service between Web servers and browsers.

SSL uses X.509 certificates. Server certificates provide a way for users to authenticate the identity of a Web server. The Web browser uses the server's public key to negotiate a secure TCP connection with the Web server. Optionally, the Web server can authenticate users by verifying the contents of their client certificates.

Even though SSL provides secure communications between Web servers and browsers on the Web, it cannot protect against end-system threats (see



* attribute info.: attributes, attribute issuer, validity period of attributes, etc.

Fig. 13. Attributes signed by multiple CAs in a smart certificate.

Section 4.2). For instance, if a user receives attributes from the server over a secure channel, it does not mean that we can trust the user. In other words, once the user, Alice, receives some attributes from the server over the secure channel, she is able to change the attributes or give them to other people, since SSL does not support the integrity service in the user's end system. Then, Alice (or the person impersonating Alice) can access the servers—which accept the attributes—using the forged attributes. However, as we will see later in this section, SSL can be used as part of our solution to protect information on the Web.

5.2 Smart Certificates Overview

To satisfy their own particular needs, it is technically possible for certificate authorities to simply issue a certificate that includes the subject's identity (e.g., public-key information) and attributes (e.g., roles) in the same certificate.

However, the lifetime of the identity (public key) in the certificate may be different from that of other attributes (roles) in it. Furthermore, an organizational policy usually requires different authorities for maintaining attributes and public keys. Since the current X.509 certificate cannot satisfy all the above requirements, we were motivated to design *smart certificates*, which support secure attribute services on the Web with several sophisticated features, without losing compatibility with X.509.

If we use the extension fields in an X.509 certificate effectively, as depicted in Figure 13, we can separate the authority for attribute-issuing from public-key-issuing. In other words, after a public-key authority (basic CA) issues an X.509 basic certificate for a user, Alice, an attribute authority (for

instance, Att_1_CA) adds attributes for Alice to an extension field of the basic certificate (which contains public-key information). Consequently, the attribute authority (Att_1_CA) signs on the basic certificate and the attributes the authority added and puts the signature to another extension field in the basic certificate. This can happen multiple times on a basic certificate by different attribute authorities (denoted Att_n_CA in the figure). Later, identity verification should precede attribute verification. For instance, another party, say Bob, verifies Alice's identity first by the basic CA's signature in the smart certificate. If the authentication is successful, Bob verifies Alice's attributes by the corresponding attribute authority's signature in the extension field. If the attributes are valid, then Bob uses those attributes for his purposes. The contents of the attribute information in a smart certificate depend on applications.

The public key and the attributes can be maintained independently. For instance, even though Alice's attributes issued by her school-attribute authority expired (were revoked) in the certificate, the rest of the attributes, such as those issued by her company-attribute authority and public-key information in her basic certificate, are still valid. Each attribute authority has independent control over the attributes it issued. In other words, the school-attribute authority for Alice can change, revoke, or reissue the school attributes in Alice's certificate. Intuitively, if her basic certificate expired (was revoked), then all the attributes become meaningless. Although a smart certificate can support independent management for the public-key information and attributes, system management becomes simpler if there is only one authority controlling both sets of information.

Additionally, smart certificates are able to provide short-lived, lifetime, postdated, renewable, and confidentiality services in PKI. According to application requirements, some of these new features can be used selectively in conjunction with currently existing technologies. Note that a smart certificate is compatible with an X.509, since it keeps the same data format as an X.509. Details for motivation and techniques about smart certificates are described in Park and Sandhu [1999b].

Smart certificates support both user-pull and server-pull architectures (see Section 3). A bundled (identity and attributes) smart certificate is useful for the user-pull architecture, since Web servers require both identity and attribute information from each user in the architecture. In contrast, the bundled certificate is not a good solution for the server-pull architecture, since identity and attribute come from different places in the architecture. In the server-pull architecture, an additional channel is required for attribute transfer between the attribute server (e.g., role server) and Web servers. A detailed dscription of this implementation is available in Park et al. [20001]

5.3 RBAC Implementation by Smart Certificates

In this section we describe how we implemented RBAC with role hierarchy on the Web in the user-pull architecture using smart certificates. In this

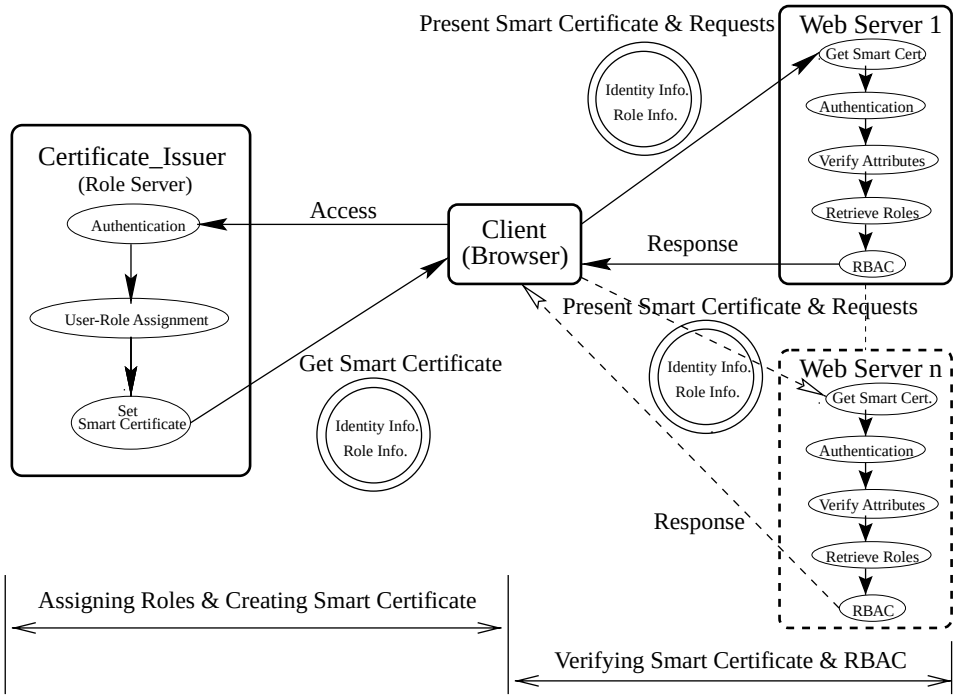


Fig. 14. RBAC on the Web by means of smart certificate.

implementation, we used a bundled (subject's identity and roles) smart certificate, maintaining compatibility with existing technologies such as SSL, without requiring an additional channel for role information transfer on the Web. We use a Netscape Certificate server and a Microsoft IIS 4.0 in the Windows NT platform for our implementation. However, this approach is also possible using other certificate servers or Web servers in different platforms.

5.3.1 *Obtaining and Presenting Assigned Roles on the Web.* Figure 14 shows how a bundled smart certificate is issued and used for RBAC on the Web. If a user, Alice, wants to execute transactions in the Web servers in an RBAC-compliant domain, she first connects to the role server at the beginning of the session. After the role server authenticates Alice, it finds her explicitly assigned roles in the URA (user-role assignment) database and creates a smart certificate (which holds her explicitly assigned roles). Then, the smart certificate is sent to and stored in Alice's machine—which has Alice's private key corresponding to the smart certificate—so that Alice does not need to go back to the role server to obtain her assigned roles until the certificate expires. Consequently, she can use the roles in her smart certificate in the RBAC-compliant domain as long as the certificate is valid. In this particular implementation, we used the OU (organization unit) field in X.509 certificates to store each subject's role information, and both identity and roles are signed by a single certificate authority. However, if a

smart certificate has different kinds of attributes (which need to be signed by different CAs), or obtains detailed attribute information, such as validity for each attribute or attribute issuer, we can use the extension fields of X.509, as we described in Section 5.2. Alternatively, separate certificates for identity and roles are also possible, especially in the server-pull architecture.

Alice may have many smart certificates in her machine. When Alice requests access to a Web server—which requires clients' certificates and has PRA (permission-role assignment) information—by typing the server's URL in her browser, the browser and Web server authenticate each other over SSL. After the browser receives and verifies the server's X.509 certificate, Alice needs to select a proper smart certificate—which has her role information—and send it to the Web server. The Web server authenticates Alice by verifying the smart certificate. If the smart certificate is valid and successfully verified, the Web server trusts the role information in the certificate and uses it for RBAC with a role hierarchy and PRA information in the Web server, as described in Section 5.3.2.

5.3.2 *RBAC in the Web Server*. The Internet information server (IIS) depends on Windows NT file system (NTFS) permissions for securing individual files and directories from unauthorized access. NTFS permissions can be precisely defined with regard to the users who can access the contents of the server and which permissions are allowed to the users, while Web server permissions are applied coarsely to the users accessing the Web server.[5] NTFS permissions only apply to a specific user or group of users with a valid Windows NT account.

In a Windows NT environment, we can control user access to the contents in a Web server by properly configuring the Windows NT file system and the security features of the Web server. When the user attempts to access the Web server, the server executes several access control processes to verify the user and determine the allowed level of access based on its policy.

To support RBAC with the role hierarchy depicted in Figure 11, we configure an IIS 4.0 with two creative ideas: *role accounts* and *PAA (permission-account assignment)* in the Web server. These ideas are described below.

Mapping Roles to Role Accounts. Since the Web server uses roles—denoted in the client smart certificates—for its access control mechanism, regular user accounts are not necessary in the server.[6]

Instead, we created the role accounts (e.g., Director, Project_Lead1, Project_Lead2, Project_Engineer1, Quality_Engineer1, and so on) in the Windows NT server, where the Web server (IIS 4.0) is installed. Then, by configuring the Web server's certificate mapping feature, we mapped each

[5]For instance, Web server permissions can control whether users visiting the Web site are allowed to view a particular page, run scripts, or upload information to the site.
[6]The Web server may need administrator accounts for its maintenance.

role in the role hierarchy in Figure 11 to the corresponding role account in the Windows NT server. For example, we mapped the role DIR to the role account Director in the server. After a user (subject), Alice, authenticates to a Web server over SSL by sending her client smart certificate—which has the role "DIR"—to the server, she is mapped to the role account "Director" in the Windows NT server. As a result, even though Alice does not have an account in the server, she acquires the Director's permission in the server, since she is assigned to the role "Director," denoted in her smart certificate. The permission of each role account depends on the policy of the Web server.

Providing Role Hierarchy. How then can the Web server support the role hierarchy? Figure 15 shows how we used a built-in access control mechanism in the Windows NT server to support the role hierarchy depicted in Figure 11. Reflecting the roles in the hierarchy, we created role accounts such as Director, Project_Lead1, Project_Lead2, Project_Engineer1, Quality_Engineer1, and others. We also created directories in the Windows NT file system, where each directory has files to be accessed by a specific role in the role hierarchy. Subsequently, we configured the Windows NT file system to assign each role account specific access rights to the directories based on the role hierarchy. For instance, the role account Project_Lead1 is assigned access rights to the Project_Lead1's directory—which has resources for the role Project_Lead1—and the directories that require the roles junior to the Project_Lead1 role in the role hierarchy. In other words, if Alice is mapped to the role account Project_Lead1, she obtains permissions assigned to the role account Project_Lead1, thereby acquiring access rights to the directories for Project_Lead1, Project_Engineer1, Quality_Engineer1, Engineer1, Engineering Department, and Employee.

As a result, after verifying the smart certificate, the Web server allows the user, Alice, to execute transactions based on her roles—contained in the OU field of the certificate—instead of her identity. In other words, the Web server does not care about the user's identity. This resolves the scalability problem of identity-based access control, which is used primarily in existing Web servers. Furthermore, since the Web server also uses a role hierarchy, it supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy different from that in other servers. The location of RBAC-compliant Web servers is geographically free from that of the role server, since smart certificates (which include the subjects' role information) can be issued by one certificate server for use by other Web servers, regardless of their physical location.

5.4 Summary

In this section we have described how we implemented RBAC with role hierarchies on the Web in user-pull architecture using *smart certificates*. The certificate authority issues a smart certificate, including a subject's

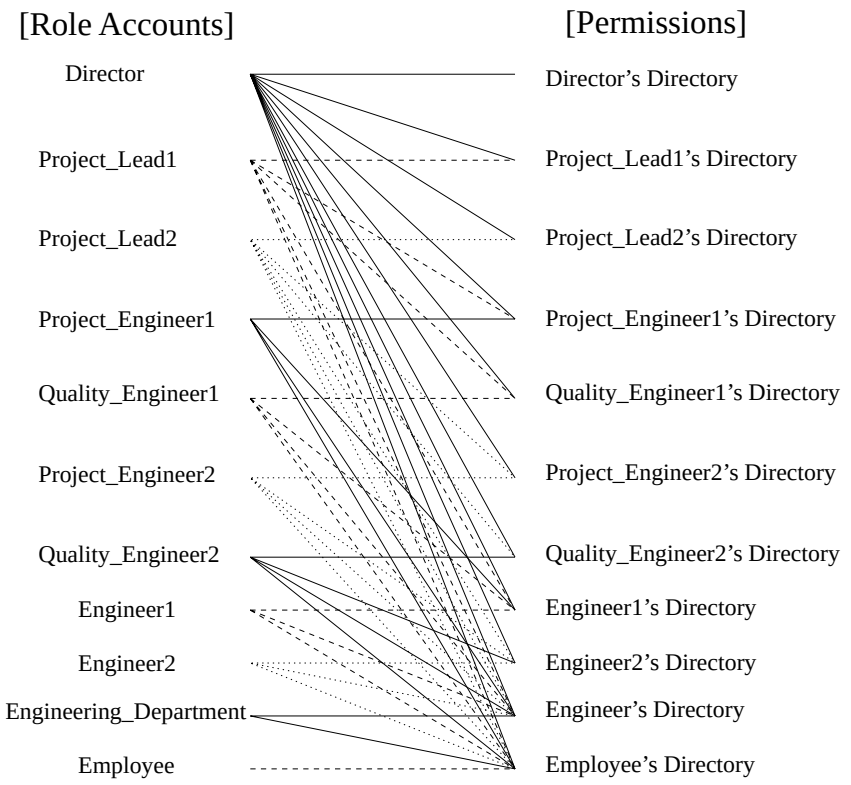


Fig. 15. Permission-account assignment (PAA).

identity and role information, and Web servers use the role information for RBAC with role hierarchies after verification. This access control mechanism solves the scalability problem of existing Web servers. The implementation is transparent to users and applicable to existing Web servers and browsers.

6. RBAC ON THE WEB IN SERVER-PULL ARCHITECTURE USING LDAP

Directory services will be the foundation for e-commerce and extranet applications that put business processes in the network. Directories will allow people to collaborate and share information both internally and externally. With the directory services feature, we use LDAP (Lightweight Directory Access Protocol [Howes et al. 1999]) server and client for our RBAC implementation.

6.1 Lightweight Directory Access Protocol (LDAP)

User information is often fragmented across the enterprise, leading to data that is redundant, inconsistent, and expensive to manage. Directories are viewed as one of the best mechanisms to make enterprise information available to multiple different systems within an organization. Directories

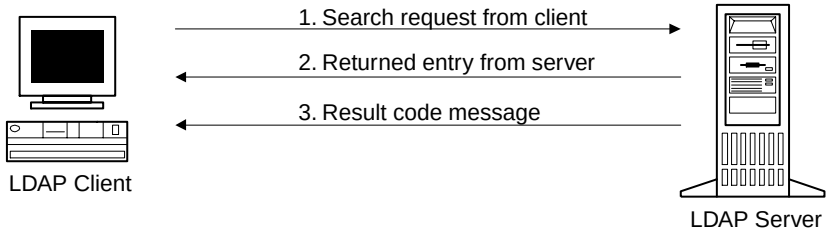


Fig. 16. LDAP operation.

also make it possible for organizations to access information over the Internet. The most common information stored in a directory service is about users on a network; this can include user id, passwords, assigned groups, and a user's network access rights. In order to retrieve the information, a directory access protocol is used to convey the entries from a directory-oriented server. The trend towards directories has been accelerated by the recent growth of LDAP.

LDAP is a protocol that enables X.500–based directories to be read through Internet clients. It was developed by the University of Michigan and the Internet Engineering Task Force (IETF) as a set of network services to provide object management and access over TCP/IP networks. LDAP is a message-oriented protocol. When an LDAP client needs a specific entry in an LDAP server, it generates an LDAP message containing a request and sends this message to the LDAP server. The server retrieves the entry from its database and sends it to the client in an LDAP message. It also returns a result code to the client in a separate LDAP message to terminate the session. Figure 16 shows this interaction between the LDAP server and client.

Although we use LDAP between Web servers and role server for RBAC in server-pull architecture, it is also possible to use LDAP for user-pull architecture, where clients can retrieve their roles from the role server via LDAP and present them to Web servers. A detailed description of this implementation is available in Park et al. [2001].

6.2 Implementing RBAC on the Web in Server-Pull Architecture

We use LDAP to communicate between the directory-oriented role server and the Web servers. In our implementation we use Netscape Directory Server as an LDAP-supporting role server and its group objects as users' role attributes. This directory-oriented role server contains users' role information to be used for access control by Web servers.

The basic scenario of our implementation is that a client presents her authentication information to a Web server and then, after a successful authentication process, the Web server gets the client's role information from the role server via LDAP to use those roles for RBAC services in the Web server.

For this purpose, we set up a computing environment based on the server-pull architecture (see Section 3, Figure 3). Figure 18 shows the

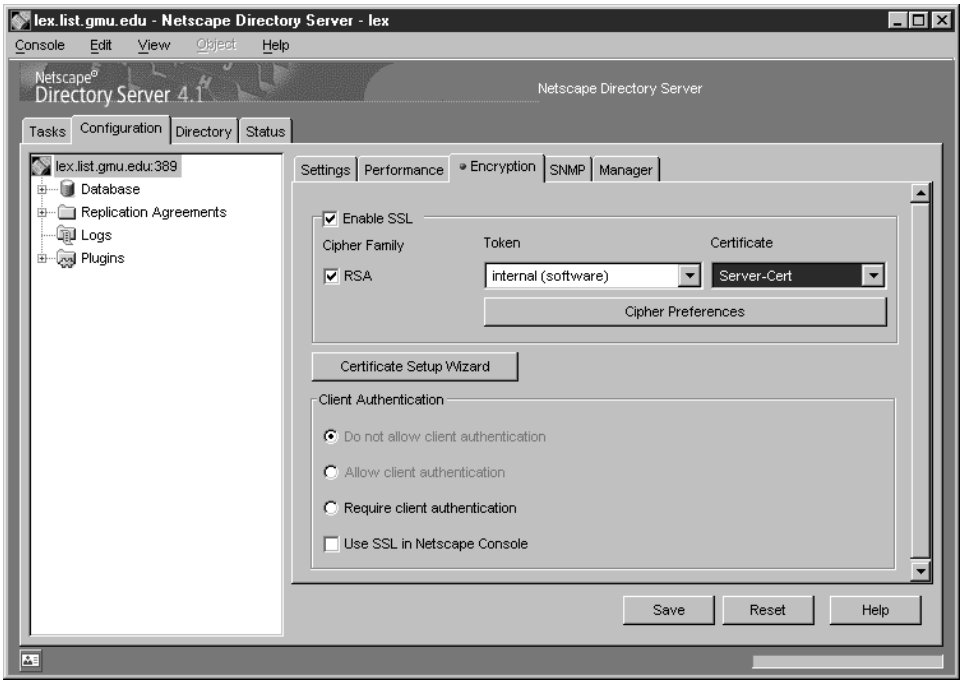
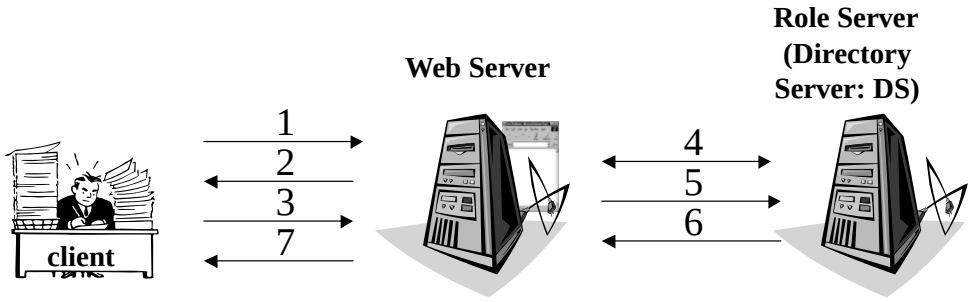


Fig. 17. Directory server: Enabling SSL.

transaction procedures of our experiment. We have three major components: Web server, role server, and client. The role server keeps URA (user-role assignment) information. The Web server contains resources that require particular roles to be accessed. The Web server also contains a PRA (permission-role assignment) table, which specifies the required roles for particular resources in the Web server. This table is referenced to check if the user has proper roles to access particular resources in the Web server. Clients use Web browsers to connect Web servers over HTTP or HTTPS.

The detailed transaction procedures are as follows. A client presents her authentication information to a Web server. We can use username/passwords, IP numbers, client certificates,[7]or other authentication techniques for this purpose. The Web server authenticates the user using a proper authentication mechanism. Once a user is successfully authenticated by the Web server (otherwise the user gets an error message), the Web server triggers the CGI (common gateway interface) scripts that call the LDAP client software. The LDAP software sends a search query to the LDAP server, which retrieves the user's roles from the directory server through SSL. The retrieved roles are sent back to the LDAP client in the Web server during the same SSL session. When the user requests an access to the

[7]To use client certificates, which is an optional operation of SSL, we need to configure the Web server to accept and understand particular client certificates. Note that this SSL channel is optional and independent of the SSL connection between Web servers and the role server.



1 Authenticate client (using password or client certificate)
2 Display the initial page
3 Request resources by clicking a link
4 Establish SSL between DS and Web server
5 LDAP over SSL; Request client's role information to DS
6 LDAP over SSL; Return client's role information to Web server
7 Display appropriate resources after authorization check based on client's roles

Fig. 18. Transaction procedures for RBAC in server-pull architecture.

resources that require particular roles to be accessed, the Web server compares the user's roles (which it pulled from the role server) with the ones in its PRA table by clicking the corresponding link in the initial page. If the user has corresponding roles in the table, the Web server allows the user to access the resources.

Figure 17 is a snapshot that shows the directory server (role server in our case) configuration. This directory server runs a process called *slapd* (LDAP server daemon) to allow requests from LDAP clients. We configure this server to have two network ports: one is for regular port and the other is for secure communications. For secure communications, we need to establish SSL between the LDAP client (installed in the Web server in our implementation) and the directory server, more specifically the LDAP server. Figure 17 also shows the configuration of SSL that installs a server certificate issued by a certificate authority.

Once a user is authenticated by the Web server, the Web server triggers CGI scripts that call LDAP client software. The LDAP client software sends a search query to the LDAP server, which retrieves the user's roles from the directory server. The retrieved roles are sent back to the LDAP client in the Web server. The Web server receives those roles and uses them for RBAC.

6.3 Summary

In this section we described how we implemented RBAC on the Web using LDAP and SSL in the server-pull architecture. The user presents her

Table I. A Comparison of User-Pull and Server-Pull Architectures

	User-Pull Architecture	Server-Pull Architecture
User's convenience	Low	High
Performance	High	Low
Reusability	High	Low
Role freshness	Low	High
Single-point failure	Low	High

authentication information to a Web server and then, after a successful authentication process, the Web server gets the user's role information from the role server through LDAP to use those roles for the access control services in the Web server. We used SSL between the role server and Web servers to protect their communication on the Web.

7. DISCUSSIONS

In this section we discuss the tradeoffs between the user-pull (implemented and described in Sections 4 and 5) and the server-pull (implemented and described in Section 6) architectures using different technologies for RBAC on the Web. In the user-pull architecture, the user pulls her roles from the role server and then presents the role information to the Web servers along with her authentication information. In the server-pull architecture, the user presents her authentication information to the Web servers, which pull the user's role information from the role server for RBAC after successful authentication.

The user-pull architecture requires a user's cooperation to obtain her roles, but it enhances Web server performance. Once the user obtains her roles, she can use them in many different sessions, and even in different Web servers, until the roles expire. This increases reusability. With this feature the user-pull architecture is a good solution for applications, especially where the users' convenience is required for maintaining and using their roles frequently in diverse Web sites. However, the longevity of the roles decreases their freshness. For instance, if the user already pulled her roles, the updated version in the role server would not become effective instantly. Consequently, an additional synchronization process is required. Hence the role server should push the status change in user roles, such as role revocation, to the Web server for updated information.

The server-pull architecture requires the Web server's cooperation for obtaining the user's role information—which decreases Web server performance—from the role server. In this architecture, the Web server retrieves the user's role information from the role server for each session. This increases the freshness of the roles, so the information update (e.g., role revocation) is more efficient than user-pull architecture, since all the roles are stored in the role server and pulled by the Web servers on demand. With this feature, the server-pull architecture is a good solution for applications, especially where dynamic role update is critical. However, it decreases reusability and increases the single-point failure vulnerability

because every session requires an access to the role server. We summarize the comparison of user-pull and server pull architectures in Table I.

Secure cookies inherently support the user-pull architecture only— because cookies are stored in users' machines, they cannot operate in the server-pull architecture. When the user connects to a Web server, the relevant secure cookies are selected and presented to the server by the browser and expired cookies are deleted from the user's machine automatically. In contrast, smart certificates support both user-pull and server-pull architectures. A bundled (identity and roles) smart certificate is useful for the user-pull architecture. To use smart certificates, user cooperation is required. Whenever the user connects to a Web server, which requires a smart certificate from the client, the user needs to select a proper certificate among her available certificates, and present it to the server. Once Web servers install a CA certificate as an acceptable certificate under some policy, a client certificate can be used in many Web servers (even in different domains). For instance, Alice's smart certificate—which has her credit card information—can be used in many Web sites in different domains for electronic commerce on the Web.

The technologies that we introduced in this article are compatible with existing technologies; HTTP can support the secure cookie mechanism as it does for regular cookies; SSL can support smart certificates as it does for X.509 certificates; and LDAP can be easily integrated with existing Web components.

8. RELATED WORK

8.1 *getAccess*

enCommerce has released *getAccess* [enCommerce 2000] to implement a hierarchical RBAC for the organization online. Each role defines a specific access privilege to one or more resources. The roles can be grouped into *macro roles*, and macro roles can also have other macro roles. There are four main software modules in this product: registry server, access server, administration application, and integration tools. The access server is located in a company's Intranet or Extranet, while the registry server is always located in the Intranet. A user always connects to the access server first via browsers. The access server then connects the registry server to obtain the user's identification and roles through a secure connection. Subsequently, the registry server authenticates the user and returns the user's encrypted role information through cookies. These cookies are temporarily stored in RAM on the user's machine while the browser is open. When the user connects to a Web server in the Intranet, the browser sends the cookies to the Web server. The Web server then decrypts and uses the encrypted role information in the cookies for RBAC in the server.

The *getAccess* mechanism uses encrypted cookies, but there is substantial difference between this approach and our *secure cookies*. Their encrypted cookies are not stored in the user's machine after the session. In other

words, if a session is ended by closing the browser, the encrypted cookies disappear. This means that whenever a user, Alice, needs to connect to a Web server with her roles, she must connect to the registry server through the access server first. On the contrary, secure cookies—which obtain the user's role information—can be stored in the user's machine securely after the session, even when the power of the user's machine is off. This is possible because the secure cookies can be provided with integrity and authentication services as well as encryption. Therefore, once Alice obtains her secure cookies, she can use her roles until the cookies expire, without having to connect to the cookie issuer.

8.2 *TrustedWeb*

Siemens Nixdorf has released *TrustedWeb* [Siemens Nixdorf 2000], which supports RBAC for Web contents and applications, as well as security services such as mutual authentication, integrity, and confidentiality for Intranets. The system, combining elements from both Sieman's SESAME [Parker and Pinkas 1995] and Kerberos [Neuman 1994], provides a single list of users on its central domain security server and assigns roles to the users. Hence access to individual Web servers in the Intranet is controlled on the basis of the role rather than the identity of the user. However, to use *TrustedWeb*, the client's browser needs specific software installed in the client's machine to communicate with the *TrustedWeb* servers in the Intranet, while our techniques do not require any specific software on the client side.

9. CONCLUSIONS

In this article we have identified the user-pull and server-pull architectures for RBAC services on the Web. In the user-pull architecture, a user pulls her roles from the role server and then presents them to the Web servers. In the server-pull architecture, each Web server pulls the user's roles from the role server as needed. Each architecture can be made to work—and we provide an analysis of their relative advantages and disadvantages.

We also developed secure cookies and smart certificates to support the architectures on the Web. Secure cookies are constructed by cryptographic technologies to support authentication, integrity, and confidentiality services. Smart certificates have new features, but they are still compatible with X.509 certificates. They are able to support short-lived lifetime and multiple CAs without losing effective maintenance, contain attributes, provide postdated and renewable certificates, and provide confidentiality. Which of these new techniques is selected depends on the applications.

To show the feasibility of our new ideas, we have implemented each architecture by integrating and extending well-known technologies, such as cookies, X.509, SSL, and LDAP, which provide compatibility with current Web technologies. We described how we implemented RBAC on the Web in different architectures using different technologies. We also compared the

tradeoffs of the various approaches on the basis of our our hands-on experiences.

We believe that our contribution is an important step towards offering strong and efficient security management based on users' roles on the Web.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation and the National Security Agency.

REFERENCES

AHN, G.-J. AND SANDHU, R. S. 2000. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur. 3*, 4 (Nov.).

AHN, G.-J., SANDHU, R. S., KANG, M., AND PARK, J. 2000. Injecting RBAC to secure a Web-based workflow system. In *Proceedings of 5th ACM Workshop on Role-Based Access Control* (RBAC '00, Berlin, Germany, July 26 - 27). ACM, New York, NY.

BELLARE, M., CANETTI, R., AND KRAWCZYK, H. 1996. Keying hashing functions for message authentication. In *Proceedings of the Conference on Advances in Cryptography* (CRYPTO '96). Springer-Verlag, New York, NY.

BOOCH, G., RUMBAUGH, J., AND JACOBSON, I. 1999. *The Unified Modeling Language User Guide*. Addison-Wesley Publishing Co., Inc., Redwood City, CA.

CALLAS, J., DONNERHACKE, L., FINNEY, H., AND THAYER, R. 1998. OpenPGP message format. RFC 2440.

DIERKS, T. AND ALLEN, C. 1999. The TLS (Transport Layer Security) Protocol. RFC 246.

DIFFIE, W. AND HELLMAN, M. 1997. ANSI X9.42: Establishment of symmetric algorithm keys using Diffie-Hellman. ANSI, New York, NY.

ELLISON, C., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. 1999. SPKI (simple public key infrastructure). RFC 2693.

ENCOMMERCE. 2000. getAccess. http://www.encommerce.com/products.

FARRELL, S. 1998a. An Internet AttributeCertificate profile for Authorization. Draft. draft-ietf-tls-ac509prof-00.txt.

FARRELL, S. 1998b. TLS extensions for AttributeCertificate based authorization. Draft. draft-ietf-tls-attr-cert-00.txt.

FERRAIOLO, D., CUGINI, J., AND KUHN, R. 1995. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th Annual Conference on Computer Security Applications* (New Orleans, LA, Dec. 11-15). 241–248.

FERRAIOLO, D. AND KUHN, D. R. 1992. Role based access control. In *Proceedings of the 15th Annual Conference on National Computer Security*. National Institute of Standards and Technology, Gaithersburg, MD, 554–563.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. 1999. Hypertext Transfer Protocol—HTTP/1.1. RFC 2616. ftp://ftp.isi.edu/in-notes/rfc2616.txt.

GARFINKEL, S. 1995. *Pretty Good Privacy*. O'Reilly Associates.

GUIRI, L. 1995. A new model for role-based access control. In *Proceedings of the 11th Annual Conference on Computer Security Applications* (New Orleans, LA, Dec.). IEEE Computer Society Press, Los Alamitos, CA, 249–255.

GIURI, L. AND IGLIO, P. 1996. A formal model for role-based access control with constraints. In *Proceedings of 9th IEEE Workshop on Computer Security Foundations* (Kenmare, Ireland, June). IEEE Press, Piscataway, NJ, 136–145.

HOUSLEY, R., FORD, W., POLK, W., AND SOLO, D. 1998. Internet X.509 public key infrastructure certificate and CRL profile. Draft. draft-ietf-pkix-ipki-part1-11.txt.

HOWES, T., SMITH, M., AND GOOD, G. 1999. *Understanding and Deploying LDAP Directory Services*. Macmillan Publishing Co., Inc., Indianapolis, IN.

HU, M.-Y., DEMURJIAN, S., AND TING, T. 1995. User-role based security in the ADAM object-oriented design and analyses environment. In *Database Security VIII: Status and*

Prospects, J. Biskup, M. Morgernstern, and C. Landwehr, Eds. Elsevier North-Holland, Inc., Amsterdam, The Netherlands.

ITU-T. 1993. Information technology—Open systems Interconnection—The Directory: Authentication framework. ITU-T Recommendation X.509. ISO/IEC 9594-8:1993.

ITU-T. 1997. Information technology—Open systems interconnection—The directory: Authentication framework. Recommendation X.509.

KRISTOL, D. M. AND MONTULLI, L. 1999. HTTP state management mechanism. draft-ietf-http-state-man-mec-12.txt.

LAI, X. AND MASSEY, J. L. 1991. A proposal for a new block encryption standard. In *Proceedings of the Workshop on Advances in Cryptology* (EUROCRYPT '90, Aarhus, Denmark, May 21–24), I. B. Damgård, Ed. Springer Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 389–404.

MOHAMMED, I. AND DILTS, D. M. 1994. Design for dynamic user-role-based security. *Comput. Security 13*, 8, 661–671.

MOORE, K. AND FREED, N. 1999. Use of HTTP state management. Draft. draft-ietf-http-state-man-mec-12.txt.

NEUMAN, C. 1994. Using Kerberos for authentication on computer networks. *IEEE Commun. Mag. 32*, 9.

NIXDORF, S. 2000. TrustedWeb. http://www.sse.ie/TrustedWeb.

NYANCHAMA, M. AND OSBORN, S. L. 1994. Access rights administration in role-based security systems. In *Proceedings of the IFIP Working Group 11.3 Working Conference on Database Security*. Elsevier North-Holland, Inc., Amsterdam, The Netherlands, 37–56.

OSBORN, S., SANDHU, R. S., AND MUNAWER, Q. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Security 3*, 2 (May).

PARK, J. S., AHN, G. -J., AND SANDHU, R. S. 2001. RBAC on the Web using LDAP. In *Proceedings of the 15th IFIP WG 11.3 Working Conference on Database and Application Security* (Ont., Canada, July 15–18). IFIP.

PARK, J. S. AND SANDHU, R. S. 2000a. Binding identities and attributes using digitally signed certificates. In *Proceedings of 16th Annual Conference on Computer Security Application* (New Orleans, LA, Dec. 11-15).

PARK, J. S. AND SANDHU, R. S. 2000b. Secure cookies on the Web. *IEEE Internet Comput. 4*, 4 (July-Aug.), 36–44.

PARK, J. S. AND SANDHU, R. S. 1999a. RBAC on the Web by smart certificates. In *Proceedings of 4th ACM Workshop on Role-Based Access Control* (RBAC '99, Fairfax, VA, Oct. 28-29). ACM, New York, NY.

PARK, J. S. AND SANDHU, R. S. 1999b. Smart certificates: Extending X.509 for secure attribute services on the Web. In *Proceedings of 22nd National Conference on Information Systems Security* (Crystal City, VA, Oct.).

PARK, J. S., SANDHU, R. S., AND GHANTA, S. 1999. RBAC on the Web by secure cookies. In *Proceedings of the IFIP WG11.3 Workshop on Database Security* (July). Chapman & Hall, London, UK.

PARKER, T. AND PINKAS, D. 1995. SESAME V4—OVERVIEW: Version 4. SESAME Technology.

RESCORLA, E. AND SCHIFFMAN, A. 1998. Security extensions For HTML. Draft. draft-ietf-wts-shtml-05.txt.

RIGNEY, C., RUBENS, A., SIMPSON, W. A., AND WILLENS, S. 1997. Remote authentication dial In user service RADIUS. RFC 2138.

RIVEST, R. 1992. The MD5 message digest algorithm. RFC 1321.

RIVEST, R., SHAMIR, A., AND ADELMAN, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 2 (Feb.), 120–126.

SANDHU, R. S. 1995. Rationale for the RBAC96 family of access control models. In *Proceedings of the First ACM Workshop on Role-Based Access Control* (RBAC '95, Gaithersburg, MD, Nov. 30,–Dec. 1), C. E. Youman, R. S. Sandhu, and E. J. Coyne, Eds. ACM Press, New York, NY.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1994. Role-based access control: A multi-dimensional view. In *Proceedings of the 10th Conference on Computer Security Applications* (Dec.). IEEE Computer Society Press, Los Alamitos, CA, 54–62.

SANDHU, R. S., BHAMIDIPATI, V., AND MUNAWER, Q. 1999. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur. 1*, 2 (Feb.), 105–135.

SANDHU, R. S. AND PARK, J. S. 1998. Decentralized user-role assignment for Web-based intranets. In *Proceedings of the Third ACM Workshop on Role-Based Access Control* (RBAC '98, Fairfax, VA, Oct. 22–23), C. Youman and T. Jaeger, Chairs. ACM Press, New York, NY, 1–12.

SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. 1996. Role-based access control models. *IEEE Computer 29*, 2 (Feb.), 38–47.

SCHIFFMAN, A. AND RESCORLA, E. 1998. The secure HyperText transfer protocol. Draft. draft-ietf-wts-shttp-06.txt.

STEINER, J., NEUMAN, C., AND SCHILLER, J. 1988. Kerberos: An authentication service for open network systems. In *Proceedings on USENIX Winter Conference*. USENIX Assoc., Berkeley, CA.

VON SOLMS, S. H. AND VAN DER MERWE, I. 1994. The management of computer security profiles using a role-oriented approach. *Comput. Security 13*, 8, 673–680.

WAGNER, D. AND SCHNEIER, B. 1996. Analysis of the SSL 3.0 protocol. In *Proceedings of the USENIX Conference on Electronic Commerce*. USENIX Assoc., Berkeley, CA, 29–40.

YOUMAN, C., COYNE, E., AND SANDHU, R. S., EDS. 1997. *Proceedings of the Second ACM Workshop on Role-Based Access Control*. (RBAC '97, Fairfax, VA, Nov. 6–7). ACM Press, New York, NY.

ZIMMERMANN, P. R. 1995. *The Official PGP User's Guide*. MIT Press, Cambridge, MA.

Received: May 2000; revised: November 2000; accepted: February 2001