

RBAC ON THE WEB BY SECURE COOKIES

Joon S. Park, Ravi Sandhu and SreeLatha Ghanta

Abstract Current approaches to access control on Web servers do not scale to enterprise-wide systems, since they are mostly based on individual users. Therefore, we were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments. *Cookies* can be used to support RBAC on the Web, holding users' role information. However, it is insecure to store and transmit sensitive information in cookies. Cookies are stored and transmitted in clear text, which is readable and easily forged. In this paper, we describe an implementation of Role-Based Access Control with role hierarchies on the Web by secure cookies. Since a user's role information is contained in a set of secure cookies and transmitted to the corresponding Web servers, these servers can trust the role information in the cookies after cookie-verification procedures and use it for role-based access control. In our implementation, we used CGI scripts and PGP (Pretty Good Privacy) to provide security services to secure cookies. The approach is transparent to users and applicable to existing Web servers and browsers.

Keywords: Cookies, role-based access control, Web security

1. INTRODUCTION

WWW is commonplace. 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 users; therefore, they do not scale to enterprise-wide systems.

A successful marriage of the Web and a strong and efficient access control technology has potential for considerable impact on and deployment of effective enterprise-wide security in large-scale systems. Role-based access control (RBAC) [14] is a promising technology for managing and enforcing security in large-scale enterprise-wide systems, and it will be a central component of

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35508-5_22](https://doi.org/10.1007/978-0-387-35508-5_22)

emerging enterprise security infrastructures. We were motivated by the need to manage and enforce the strong access control technology of RBAC in large-scale Web environments.

To support RBAC on the Web, we chose a relatively mature technology, cookies - widely used on the Web - and have extended it for our purpose. Cookies were invented to maintain continuity and state on the Web [5, 6]. 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, it is not safe to store and transmit this sensitive information in cookies because cookies are insecure. Cookies are stored and transmitted in clear text, which is readable and easily forged. Therefore, we should render secure cookies to carry and store sensitive data in them.

We will 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 the contents of the cookies might be changed by unauthorized modification. Finally, confidentiality services protect cookies against the values of the cookies being revealed to an unauthorized entity. Details for these techniques have varying degrees of security and convenience for users and system administrators¹.

In this paper, we will describe how we implemented RBAC with role hierarchies [4, 18] on the Web using the secure cookies. To provide security services to secure cookies, we used CGI scripts and the PGP (Pretty Good Privacy) package, which are already in widespread current use.

2. RELATED TECHNOLOGIES

2.1 ROLE-BASED ACCESS CONTROL (RBAC)

Role-based access control (RBAC) [14] has rapidly emerged in the 1990s as a promising technology for managing and enforcing security in large-scale enterprise-wide systems. The basic notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management.

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

¹For secure communications on the Web, we may consider using other existing technologies, such as, SHTTP (Secure HTTP [10, 19]) and SSL (Secure Socket Layer [21]). However, these technologies cannot solve the *stateless* problem of HTTP. Furthermore, none of these can prevent end-system threats to cookies.

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

RBAC is a promising alternative to traditional discretionary and mandatory access controls, and ensures that only authorized users are given access to certain data or resources. It also supports three well-known security policies: data abstraction, least-privilege assignment, and separation of duties.

2.2 COOKIES

Although there are many ways to use cookies on the Web, the basic process and the contents of cookies are similar. The detailed cookie specifications are available in [5, 6].

Cookies contain strings of text characters encoding relevant information about the user. Cookies are sent to the user's memory via the browser while the user is visiting a cookie-using Web site, and are stored on the user's hard disk after the browser is closed. Whenever a browser sends an HTTP request for a URL to a Web server, only those cookies relevant to that server will be sent by the browser. If the server finds any cookies that are related to the server, those cookies are used during this communication between the browser and the server. However, if the server does not find any cookies specified for it, either that server does not use cookies in the communication or the server creates new cookies for subsequent communication between the browser and the server.

Web servers may update the contents of their cookies for any specific circumstance. The cookie-issuer is not important for cookie validation. In other words, a server can create cookies for other servers in the domain. This is an important aspect of cookies that will be used in our implementations described in Section 4.

2.3 PRETTY GOOD PRIVACY (PGP)

PGP (Pretty Good Privacy), a popular software package originally developed by Phil Zimmermann, is widely used by the Internet community to provide cryptographic routines for e-mail, file transfer, and file storage applications [22]. A proposed Internet standard has been developed [2], specifying use of PGP. It uses 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. It 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 [13] for the convenience of the public-key cryptosystem, message digests (MD5 [12]) and IDEA [7] for the speed of process, and Diffie-Hellman [3] for key exchange. The updated version supports more cryptographic algorithms.

Even though the original purpose of PGP is to protect casual e-mail between Internet users, we decided to use the PGP package. The package is already widely used and satisfies our requirements, in conjunction with Web servers via CGI scripts for our implementation to protect cookies. These cookies have role information of the user.

3. SECURE COOKIES

3.1 SECURITY THREATS TO 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 use of the Secure Sockets Layer (SSL) protocol [21] which is widely deployed in servers and browsers.² 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. Such cookies can be trivially altered and can be easily copied from one computer to another, with or without connivance of the user on whose machine the cookie was 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), later he can use those harvested cookies for all other sites that accept those cookies. We call this the cookie-harvesting threat. These attacks are all relatively easy to carry out and certainly do not require great hacker expertise.

3.2 DESIGNING SECURE COOKIES

In this subsection, we describe how to transform regular cookies - which have zero security - into secure cookies, which provide the classic security

²In many cases, due to export restrictions from USA, only weak keys (40 bits) are supported, but SSL technology is intrinsically capable of very strong protection against network threats.

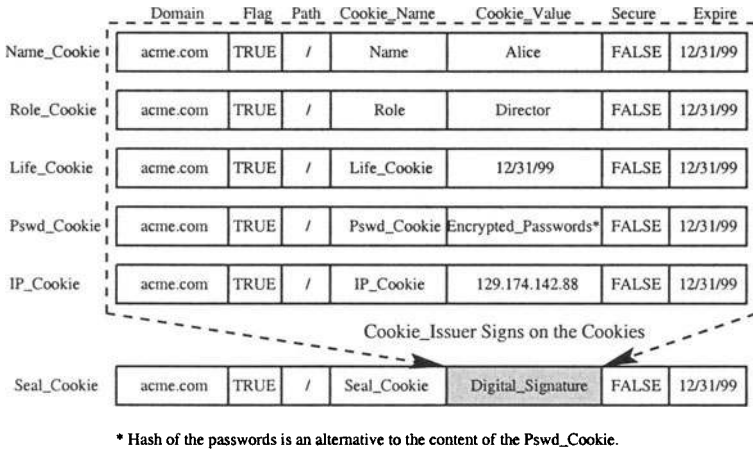


Figure 4.1 A set of secure cookies for RBAC on the Web.

services against the three types of threats to cookies (described in the previous subsection).

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 a given situation.

Figure 4.1 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., Director). 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 will 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 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 unauthorized entity. In our implementation, we used the IP_Cookie and Pswd_Cookie together to show the feasibility, but only one of those authentication cookies can be used to provide the authentication service. The choice of an authentication cookie depends on the situation.³ Finally, the Seal_Cookie - which has the dig-

³It is also possible for authentication to be based on use of RADIUS [11], Kerberos [20, 8], and similar protocols. Our focus in this work is on techniques that make secure cookies self-sufficient rather than partly relying on other security protocols, which is always possible.

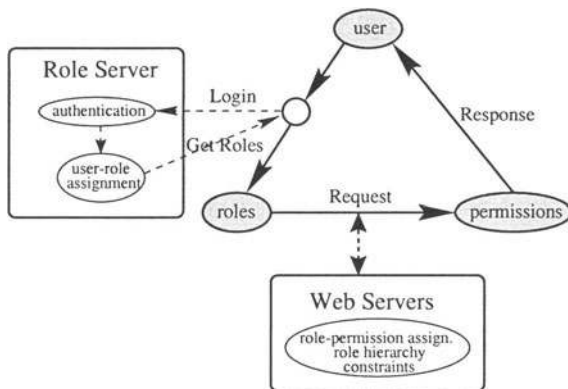


Figure 4.2 A schematic of RBAC on the Web.

ital signature of the cookie-issuing server on the secure cookie set - supports integrity service, protecting cookies against the threat that the contents of the cookies 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 scripts. In the next section, we will describe secure cookie creation, verification, and use of the role information in the Role.Cookie for RBAC with role hierarchies, in turn.

4. RBAC IMPLEMENTATION BY SECURE COOKIES

Figure 4.2 shows a schematic of RBAC on the Web. The role server has 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 identity. The Web servers may have role hierarchies or constraints based on their policies.

However, 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. Therefore, 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 [9]. In this paper, as one possible solution, we will describe how to protect the role

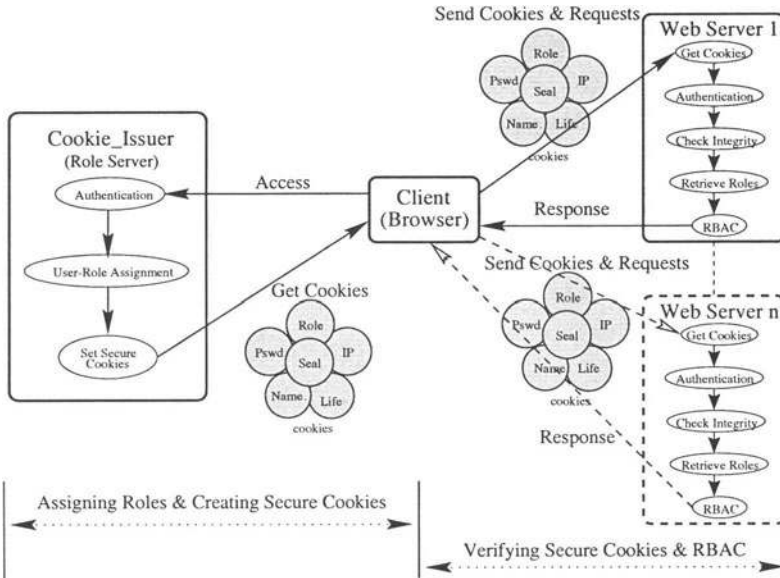


Figure 4.3 RBAC on the Web by secure cookies.

information from possible threats using secure cookies, and how we implemented RBAC (Role-Based Access Control) with role hierarchy on the Web. Figure 4.3 shows how the 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 an 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 (User-Role Assignment [17, 15]) database and creates a set of secure cookies. Then, those secure cookies are sent to and stored in Alice's hard drive securely so that Alice does not need to go back to the role server to get her assigned roles until the cookies expire. Namely, she can use the roles in 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 (Permission-Role Assignment [16]) 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 IP_Cookie and Pswd_Cookie, comparing the value in the cookies with the values coming from the user. The user's passwords are encrypted in the Pswd_Cookie using the Web server's public key. The Web server decrypts the value of the Pswd_Cookie by using the corresponding key to read the user's passwords. Finally, the Web server checks the integrity of the cookies by verifying role

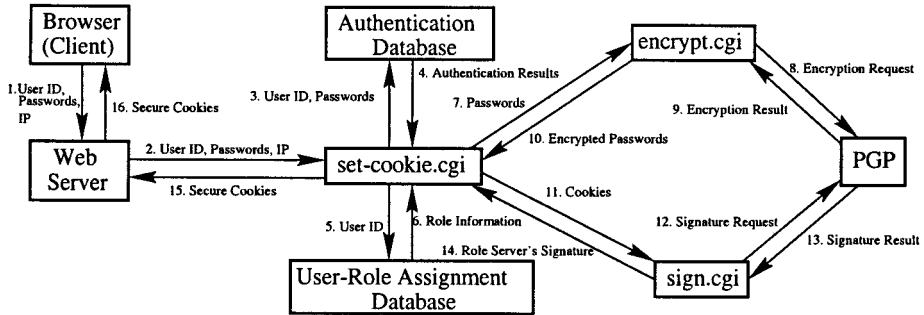


Figure 4.4 Creating secure cookies.

server's digital signature in the Seal.Cookie using the role server's public key. If all the cookies are valid and verified successfully, the Web server trusts the role information in the Role.Cookie and uses it for RBAC with a role hierarchy and permission-role assignment information in the Web server.

4.1 SECURE COOKIE CREATION

Figure 4.4 is a collaborative diagram in UML (Unified Modeling Language [1]) style notation for secure cookie creation. This diagram shows how we create a set of secure cookies for our implementation (refer to the left side of Figure 4.3).

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 the client machine. The program authenticates the user by comparing the user ID and passwords with the ones in the authentication database.⁴ 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. Then, the “set-cookie.cgi” program creates IP.Cookie, Pswd.Cookie, Name.Cookie, Life.Cookie, and

⁴If 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.


```

# Netscape HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file! Do not edit.

list.gmu.edu TRUE / FALSE 924951853 Name Alice
list.gmu.edu TRUE / FALSE 924951853 Role Director
list.gmu.edu TRUE / FALSE 924951853 Life 12/31/99
list.gmu.edu TRUE / FALSE 924951853 IP 129.174.40.15

list.gmu.edu TRUE / FALSE 924951853 Pswd hEwDNMBB1eJQr
WEBAF3/dJRFK/Y2Bry30YgnK6rJJQ1xqRzbeyLYkuWU0bFwK5Xy0XIX8dT8s4k1cy#Z2nI1CzJJn1
F8pPUqR89xSYm7pgAAACBw76iNNPT51vSHxxF+3dBRRN/I78pwwrTzjTcOX7cXtQ==94Cd

list.gmu.edu TRUE / FALSE 924951853 Seal owEBigB1/4kAV
QMFAdb6U+s0wEGV41CtYQEBcUoB/1kGcmZqRSFxbHsdNVFBxFeq6QOFRUCTrwy7/kaNynsRheX

```

Figure 4.5 An example of secure cookies stored in a user's machine.

Role_Cookie, giving each cookie the corresponding value: IP number of the client machine, encrypted passwords, user's name, lifetime of the cookie set, and assigned roles.

To support the integrity service of the cookies, the "set-cookie.cgi" program calls another CGI program (sign.cgi), which uses PGP to sign the message digest 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 message digest of the cookies.

Finally, the Web server sends the HTTP response header, along with the cookies, back to the user's browser, and the cookies are stored in the browser until they expire. These secure cookies will be verified and used in the Web servers as described in the following subsections. Figure 4.5 is an actual snapshot of a set of secure cookies from our implementation that are stored in the user's machine after the cookies are generated by the cookie-issuing Web server. The contents of the cookies exactly reflect the ones presented in Figure 4.1.

4.2 SECURE COOKIE VERIFICATION

Figure 4.6 is a collaborational diagram in UML style notation for secure cookie verification. This diagram shows how we verify (corresponding to the right side of Figure 4.3) the set of secure cookies that we generated in the previous subsection for our implementation. When Alice connects to a Web server (which accepts the secure cookies) in an RBAC-compliant domain, the con-

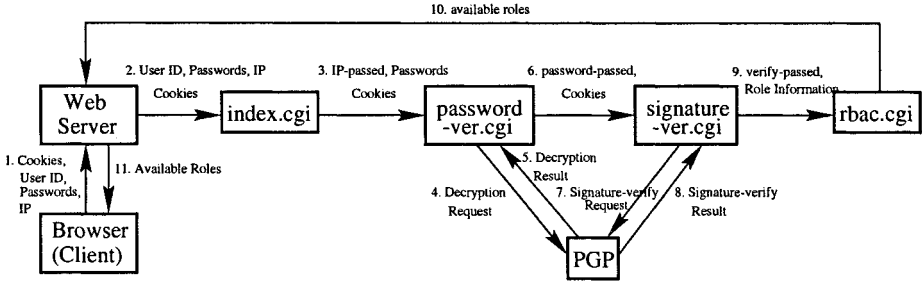


Figure 4.6 Verifying secure cookies.

nection is redirected to the “index.cgi” program. The related secure cookies are sent to the Web server and she is prompted by the 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 “status” with the value of “IP-passed” is created to indicate that this stage was passed⁵. 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 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 the password verification is completed, another CGI program, “signature-ver.cgi,” is activated to check the integrity of the cookies. Like the

⁵We used a hidden field to check the completion of the previous stage, which is passed on to the next program. This 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 can access the pages directly with forged cookies.

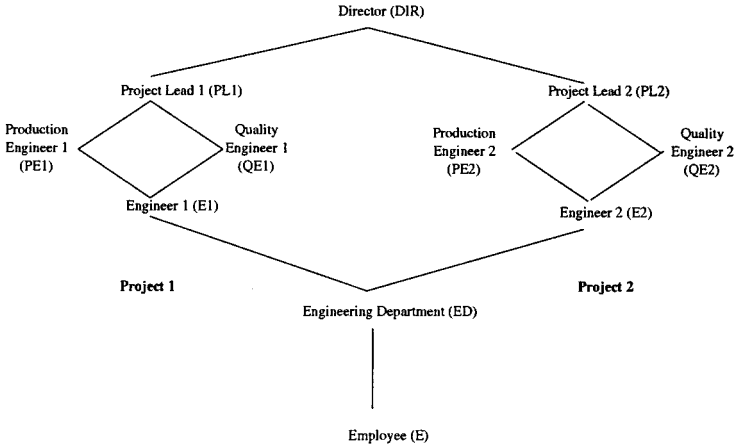


Figure 4.7 An example role hierarchy.

other programs, it first checks the value of “status” passed on from the previous program, and it proceeds only if it shows 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 of “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 role-based access control in the server as described in the following subsection. However, if the signature verification is failed, the user has to start again by receiving new cookies from the role server.

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, the Web server does not care about the user’s identity for authorization purposes. This resolves the scalability problem of the identity-based access control, which is being used mostly 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 4.7.

If the “rbac.cgi” program in Figure 4.6 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. Therefore, the Web server can trust the role information in the Role.Cookie, and uses it for role-based access control 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. 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, when his Seal.Cookie is being verified, there would be a problem, as the signature verification would fail. Therefore, he would not be allowed to move beyond this stage. On the other hand, he could try accessing the pages directly by typing the URLs. This would not be 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, which require roles senior to his, because he cannot activate the senior roles, which 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. This access also is possible only if the users have the required roles and activate them among their available roles based on the role hierarchy.

5. CONCLUSIONS

In this paper, we have described how we implemented RBAC with role hierarchies on the Web using *secure cookies*. To protect the 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. This access control mechanism solves the scalability problem of existing Web servers. The use of secure cookies is a transparent process to users and applicable to existing Web servers and browsers.

Acknowledgments

This work is partially supported by the National Security Agency under its IDEA program.

References

- [1] Booch, G., Jacobson, I., and Rumbaugh, J. (1998). *The unified modeling language user guide*. Addison-Wesley.

- [2] Callas, J., Donnerhackle, L., Finney, H., and Thayer, R. (1998). *OpenPGP message Format*. RFC 2440.
- [3] Diffie, W. and Hellman, M. (1977). *ANSI X9.42: Establishment of Symmetric Algorithm Keys Using Diffie-Hellman*. American National Standards Institute.
- [4] Ferraiolo, D., Cugini, J., and Kuhn, R. (1995). Role-based access control (RBAC): Features and motivations. *Proceedings of the Eleventh Annual Computer Security Application Conference*, pp. 241–248.
- [5] Kristol, D. M. and Montulli, L. (1998). *HTTP state management mechanism*. draft-ietf-http-state-man-mec-8.txt.
- [6] Kristol, D. M. and Montulli, L. (1998). *HTTP state management mechanism*. draft-ietf-http-state-man-mec-10.txt.
- [7] Lai, X. and Massey, J. (1991). A proposal for a new block encryption standard. In *Advances in Cryptography - CRYPTO'90 Proceedings*, Springer-Verlag, pp. 389–404.
- [8] Neuman, B. C. (1994). Using Kerberos for authentication on computer networks. *IEEE Communications*, **32**(9).
- [9] Park, J. S. and Sandhu, R. (1999). RBAC on the Web by smart certificates. *Proceedings of Fourth ACM Workshop on Role-Based Access Control*.
- [10] Rescorla, E. and Schiffman, A. (1998). *Security Extensions For HTML*. draft-ietf-wts-shtml-05.txt.
- [11] Rigney, C., Rubens, A., Simpson, W. A., and Willens, S. (1997). *Remote Authentication Dial In User Service RADIUS*. RFC 2138.
- [12] Rivest, R. (1992). *The MD5 message digest algorithm*. RFC 1321.
- [13] Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, **21**(2), pp. 120–126.
- [14] Sandhu, R. (1998). Role-based access control. *Advances in Computers*, **46**.
- [15] Sandhu, R. and Bhamidipati, V. (1997). The URA97 model for role-based administration of user-role assignment. *Database Security XI: Status and Prospects* (eds. T.Y. Lin and X. Qian), North-Holland.
- [16] Sandhu, R., Bhamidipati, V., Coyne, E., Ganta, S., and Youman, C. (1997). The ARBAC97 model for role-based administration of roles: Preliminary description and outline. *Proceedings of Second ACM Workshop on Role-Based Access Control*, pp. 41–50.
- [17] Sandhu, R. and Park, J. S. (1998). Decentralized user-role assignment for Web-based intranets. *Proceedings of Third ACM Workshop on Role-Based Access Control*, pp. 1–12.

- [18] Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, **29(2)**, pp. 38–47.
- [19] Schiffman, A. and Rescorla, E. (1998). *The Secure HyperText Transfer Protocol*. draft-ietf-wts-shhttp-06.txt.
- [20] Steiner, J., Neuman, C. and Schiller, J. (1988). Kerberos: An authentication service for open network systems. *Proceedings of the Winter USENIX Conference*.
- [21] Wagner, D. and Schneier, B. (1996). Analysis of the SSL 3.0 protocol. *Proceedings of the Second UNIX Workshop on Electronic Commerce*.
- [22] Zimmermann, P. R. (1995). *The Official PGP User's Guide*. MIT Press.