Access Control and Session Management in the HTTP Environment

A role-based access-control model is stored as LDAP objects in a security architecture that has been fielded in several implementations.

Kurt Gutzmann
Nervewire Inc.

As the only ubiquitous public data network, the Internet offers business partners a communications channel that previously existed only in unique situations with private, special-purpose networks. Well-publicized security risks, however, have limited the deployment of business-to-business *extranets*, which typically use the Internet's public data network infrastructure. These risks extend behind firewalls to *intranets*, where any user gaining entry to a facility is often implicitly authenticated to access unprotected services by simply plugging a portable computer into an unused network port.

In this article, I describe an approach that uses role-based access controls (RBACs) and Web session management to protect against network security breaches in the HTTP environment. The RBAC and session management services augment network-level security, such as firewalls, inherent in the deployment of any Web-based system with untrusted interfaces. The RBACs are implemented through the Internet Engineering Task Force's Lightweight Directory Access Protocol (for IETF documents relevant to LDAP and other Internet protocols cited in this article, see the sidebar, "IETF Protocol RFCs," p. 28). Session management is implemented through cryptographically secured, cookie-based ticket mechanisms.

The approach was fielded first as part of a project to support the sales force, agency operations, and communications for a large insurance company. It has been incrementally improved in subsequent implementations.

Security in the HTTP Environment

Role-based access controls are not part of the typical Web server software set. The HTTP RFCs specify a "401:WWW-Authenticate" server response—essentially a logon challenge—for authentication and access control.

 1089-7801/00/$10.00

Security Realms

The notion of a *security realm* applies here: a typical security realm comprises a tree or subtree of URLs for a given server. Because each realm must map to unique URL prefixes, security realms are mutually exclusive. When a Web client requests a URL from a server, the server checks the URL against its list of realms for a prefix match. For each realm, there is a corresponding access control list (ACL) that specifies—either explicitly or through a set of rules—which users are allowed access to URLs in the realm, and which users are denied.

Secure realms are useful for gross access control to a Web site. But each realm requires authentication for access, so the user task of supplying a name and password quickly becomes burdensome. The need to differentiate user roles magnifies the problem: few businesses want to maintain distinct and largely redundant Web sites and content for each user role in their authorization base. An additional, more subtle problem arises with the need to dynamically generate content and control the visible link set (that is, those URLs that we know in advance a user is authorized to access, as in a search result).

Given the issues of user complexity and Web site maintainability, secure realms are not feasible in the implementation of an RBAC security model. The approach described in this article shows how to address these issues by using network authentication services—such as LDAP, Sun Microsystem's NIS, and Microsoft's NT domains—together with an RBAC model stored as LDAP objects and secured session ticket.

Session Management for a Stateless Protocol

The problems of entity authentication, resource-access authorization, and session management are not unique to the HTTP environment. In custom client-server systems, sessions are explicitly maintained by persistent network connections and state information shared between client and server applications. The request-response-disconnect nature of HTTP precludes any shared, connection-oriented state between client and Web server, insofar as that state is based on the protocol itself.

RFC 2109 describes a state management mechanism more generally known as a session ticket. RFCs 2068 and 2616 specify HTTP's basic authentication mechanism, which is simply a user-ID and password encoded in Base64 and included as part of the HTTP request headers. From a security viewpoint, Base64 is essentially cleartext. Unless transport layer security (TLS, RFC 2246) or secure sockets layer (SSL) encryption is used, this is not a secure method for authentication.

RFC 2595 recently proposed starting a TLS session to protect what would otherwise be cleartext password authentication for three Internet standard protocols. Following this proposal, a server would augment its advertised capability set to include a "start TLS" capability. A client would issue this start command, redetermine the server's capabilities, and then perform the authentication steps of the protocol with the transport layer encryption protecting the exchange.

The message digest authentication proposed in RFC 2617 is a type of challenge-response authentication protocol that does not transmit any cleartext passwords. At present, commercial server products, such as Netscape Enterprise Server and Microsoft IIS, do not support this protocol.

Role-Based Access Control

Role-based access control provides a rich model for managing information and its accessors. Many other security models can be represented as subsets or simplifications of an RBAC model. Sandhu et al.[1] discuss various forms of RBAC. Using Sandhu's descriptive approach in the insurance company project, we implemented what is essentially $RBAC_0$, that is, role-based access without hierarchical control. However, in our implementation, session termination is system-enforced instead of user-elected, and all users have a single role (both of these are constraints under $RBAC_2$). Furthermore, the notion of sessions is limited in the HTTP environment because of the single request-response nature of the protocol.

Role-based access control provides a rich model for managing information and its accessors.

Using Sandhu's notation, $RBAC_0$ includes:

- *U*, a set of users
- *R*, a set of roles
- *P*, a set of permissions
- *PA*, a many-to-many permission-to-role assignment relation
- *UA*, a many-to-many user-to-role assignment relation
- *S*, a function mapping a session to a set of roles, possibly dynamically

This was the approach used to implement LDAP-based RBAC. Sandhu notes that the permissions are treated like uninterpreted symbols in the model

IETF Protocol RFCs

The Requests for Comments for Internet protocols cited in this article are available online at http://www.ietf.org/rfc/rfcxxx.txt, where xxxx refers to the 4-digit RFC number.

For Lightweight Directory Access Protocol

RFC 1777, W. Yeong, T. Howes, and S. Kille, "Lightweight Directory Access Protocol," Mar. 1995.
RFC 2251, M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol (v3)," Dec. 1997.
RFC 2252, M. Wahl et al., "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions," Dec. 1997.
RFC 2253, M. Wahl, S. Kille, and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names," Dec. 1997.
RFC 2254, T. Howes "The String Representation of LDAP Search Filters," Dec. 1997.
RFC 2255, T. Howes, "The LDAP URL Format," Dec. 1997.
RFC 2256, M. Wahl, "A Summary of the X.500(96) User Schema for Use with LDAPv3," Dec. 1997.

For Hypertext Transport Protocol

RFC 1945, T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext Transfer Protocol – HTTP/1.0," May 1996.
RFC 2068, R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1," Jan. 1997.
RFC 2109, D. Kristol, and L. Montulli, "HTTP State Management Mechanism," Feb. 1997.
RFC 2616, R. Fielding et al., "Hypertext Transfer Protocol – HTTP/1.1," Jan. 1999 (obsoletes RFC 2068).
RFC 2617, J. Franks et al., "HTTP Digest Authentication," June 1999.

For Transport Layer Security

RFC 2246, T. Dierks and C. Allen, "The TLS Protocol, Version 1," Jan. 1999.
RFC 2595, C. Newman, "Using TLS with IMAP, POP3 and ACAP," June 1999.

For Time Service

RFC 1305, D. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," Mar. 1992.
RFC 2030, D. Mills, "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," Oct. 1996.

For Dynamic Host Configuration Protocol

RFC 2131, R. Droms, "Dynamic Host Configuration Protocol," Mar. 1997.

definition. In the work described here, the symbol interpretation service is implemented either as an application-level service-access meditation function—which was the case with the insurance company implementation—or as an HTTP server-request intercepts filtering function—which was the case in subsequent implementations. (It may also be possible to implement $RBAC_1$ (role hierarchies) using the object class hierarchy that is part of the X.500 schema employed by LDAP directory servers.)

Security Services Architecture

Figure 1 illustrates the logical architecture for the security services in this approach.

Authentication Services

Authentication verifies a claimant's identity. The architecture in Figure 1 shows authentication services as a configurable service element. In the fielded implementation, an LDAP bind operation with a simple password provided the back-end authentication service with parameters obtained from the user in an HTTP form submitted over TLS.

Figure 1 shows several other common authentication services that may already exist in an enterprise and could also be used.

For a generic HTTP client, the authentication possibilities are limited to what can be accomplished with HTTP Basic Authentication or form submission. Form submission by the HTTP client causes the HTTP server to act as a proxy for the client in executing one of the authentication protocols. This implies that the client trusts the HTTP server in this proxy authentication role.

Session Management Services

The approach requires four session management services, shown in Figure 1 and detailed below.

Time service. The session management services related to session duration and time-out require agreement on the time. Some authentication protocols also use time-varying sources, such as challenge-response types. The required precision of time measurement is usually on the order of a few minutes for session idle time-out. The network time protocol (RFC 1305) and simple network time protocol (RFC 2030) provide close synchronization of system clocks.

User profile service. This service provides user attributes, particularly security roles and distinguished names. Other information that may be useful in the applications or content-tailoring environment may be provided, such as given name, common name, application preferences, and so on.

Ticket issuance service. This service grants a session ticket to an authenticated user. The session

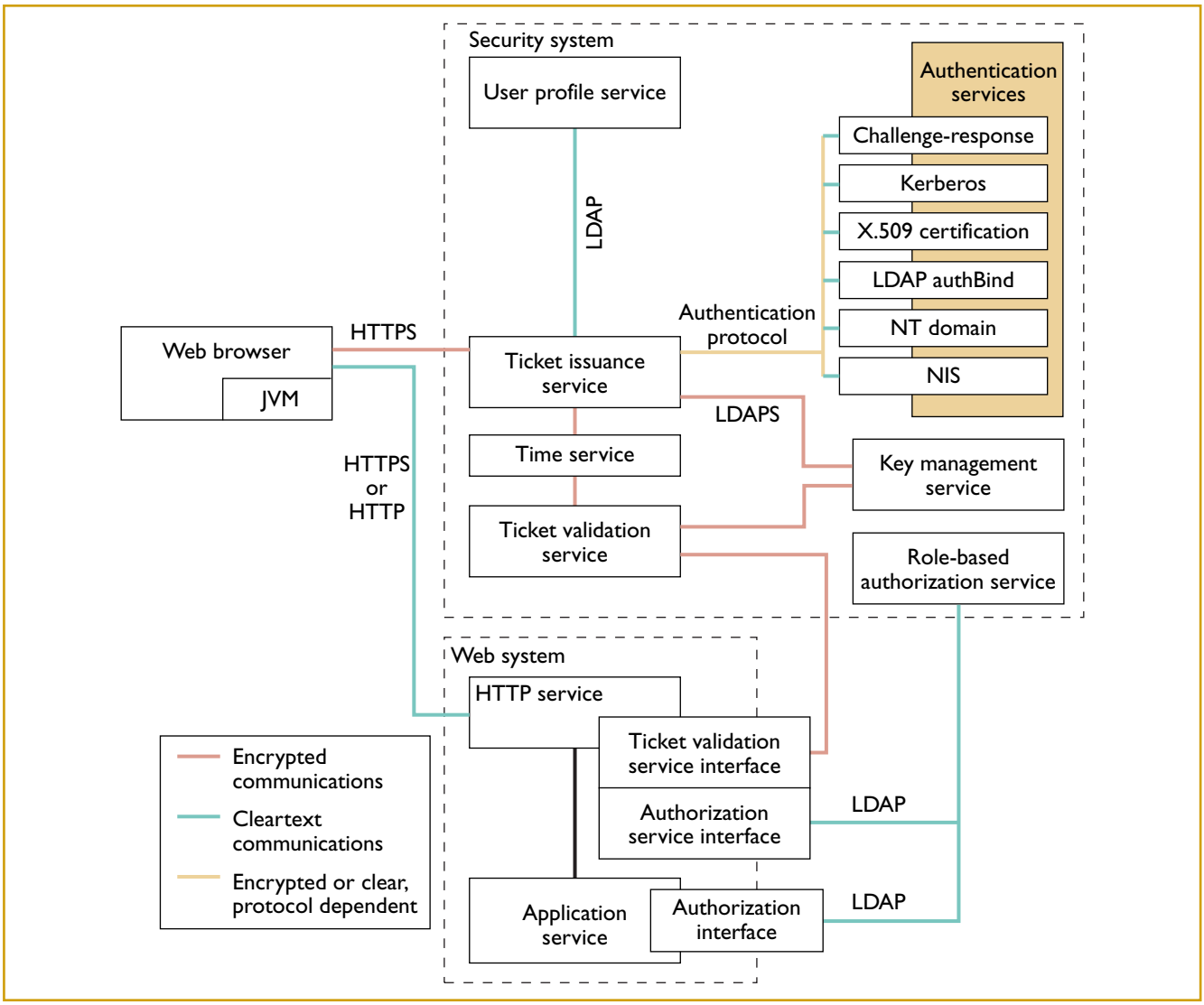

Figure 1. Security services architecture. Multiple services linked by LDAP can be used in an implementation to provide authentication, session management, and authorization.

ticket stores information about the Web site user in a tamper- and spoof-proof format and utilizes session time-out in the normally stateless HTTP environment (for session ticket specifications, see Bellovin[2]).

The session ticket based on HTTP cookies is the only standards-based, scalable method for maintaining state in the HTTP environment. The ticket can be represented either as a single cookie within which a number of values have been catenated or as a collection of distinct cookies. In our fielded implementation, a set of related session tickets was used. For implementation purposes, this collection is referred to as the session ticket (in other words, it is a set of related cookies).

The session ticket comprises a *payload*, consisting of several distinct variables and their values:

- User_IP: the client IP address to which the session ticket was issued. This is used in session ticket validation to detect source spoofing. Note that firewalls should not be configured to hide or remap the requestor address for this to be of use.
- User_ID: the username or distinguished name (DN) that was correctly authenticated to the site.
- Login_Expires: the session time-out and automatic logout function that Web browsers do not inherently support.
- Login_Expires_Absolute: the stated absolute expiration time of a session, even if it has not expired due to idleness time-out.
- Other attributes as required for the particular implementation.
- Ticket_MAC: a digital signature or message

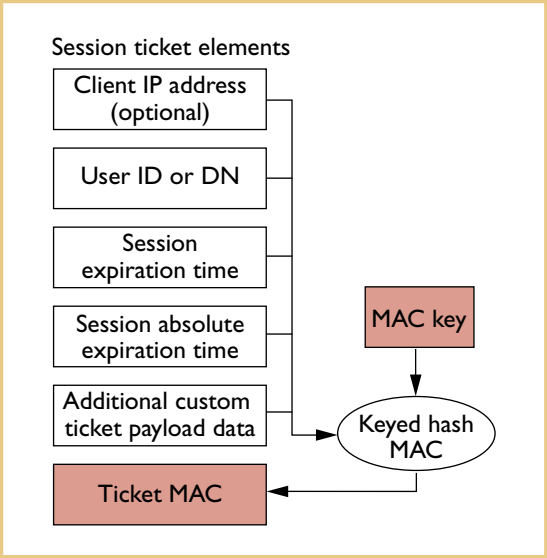

Figure 2. Session ticket secured with keyed hash message authorization code.

authentication code (MAC) computed against the catenation of the session ticket values.

The server issues the session ticket after a successful authentication protocol execution, which in most cases transpires over an encrypted SSL connection between the Web client and the Web server. Once the ticket arrives at the Web browser, it must be secured from tampering, as shown in Figure 2. A digital signature or Message Authentication Code (MAC) across the session ticket makes tampering detectable. If, for example, a user attempts to maliciously modify his role, the message represented by the session ticket will not be authentic.

The insurance company implementation used a hash-based MAC referred to as HMAC-SHA-1-160, as all 160 bits of the SHA-1 output are retained. (For an extensive discussion of keyed hash MACs, see Bellare et al.[3]) Alternative implementations of the secured session ticket are possible using symmetric cookie encryption or public key methods for digital signature.

The session ticket expiration time is determined by the earlier of the Login_Expires or Login_Expires_Absolute values in the secure session ticket. These values are determined at the time the session ticket is generated by adding the system configuration parameters of Session_Duration and Session_Duration_Absolute to the current time, obtained from the time service.

A valid session ticket is refreshed as it is used; this involves the update of the Login_Expires and Ticket_MAC values. The values are validated by a server and returned to the user in response to an HTTP request, which prevents ticket expiration while a user is active. Users would otherwise need to re-authenticate unnecessarily.

Figure 3 illustrates the logic for implementing the ticket-issuance service and its interaction with the authentication and user profile services.

Ticket validation service. After a ticket is issued, the HTTP server must validate it as presented in the request headers. Three checks are performed to validate a session ticket transmitted from a browser user to a server:

- The IP host address from which the session ticket was transmitted must match the User_IP value.
- The Ticket_MAC value (as a cookie header) in the request from the browser user must match the result of the same server-side calculation performed on the presented session ticket using the MAC key (excluding the Ticket_MAC value).
- The time provided by the time service must be earlier than the times specified in the ticket's Login_Expires and Login_Expires_Absolute values.

If a session ticket is not valid, the user is asked to reauthenticate and thereby establish a valid session. When a user successfully authenticates, the session ticket transitions state to "Valid and Not Expired." From this state, a number of possible transitions can be made:

- A ticket refresh may retain the "Valid and Not Expired" state. This is most common event.
- If the ticket is deleted (for example, the representative cookie file is deleted, or the browser application execution terminated and restarted), it arrives in a "No Ticket" state.
- If the ticket is tampered with or the machine IP address does not match the User_IP value, it arrives in an "Invalid Ticket_MAC" state.
- If the ticket is presented after it has expired, it arrives in an "Expired" state.
- A tampered and expired ticket arrives in the "Invalid and Expired" state.

From any invalid state, a transition back to the same state based on a failed reauthentication is possible. If reauthentication is successful, the state transitions back to "Valid and Not Expired."

The User_IP may have been established by a dynamic host configuration protocol (DHCP) service (RFC 2131); this occurs if sessions time out or expire in a much shorter time than a DHCP address

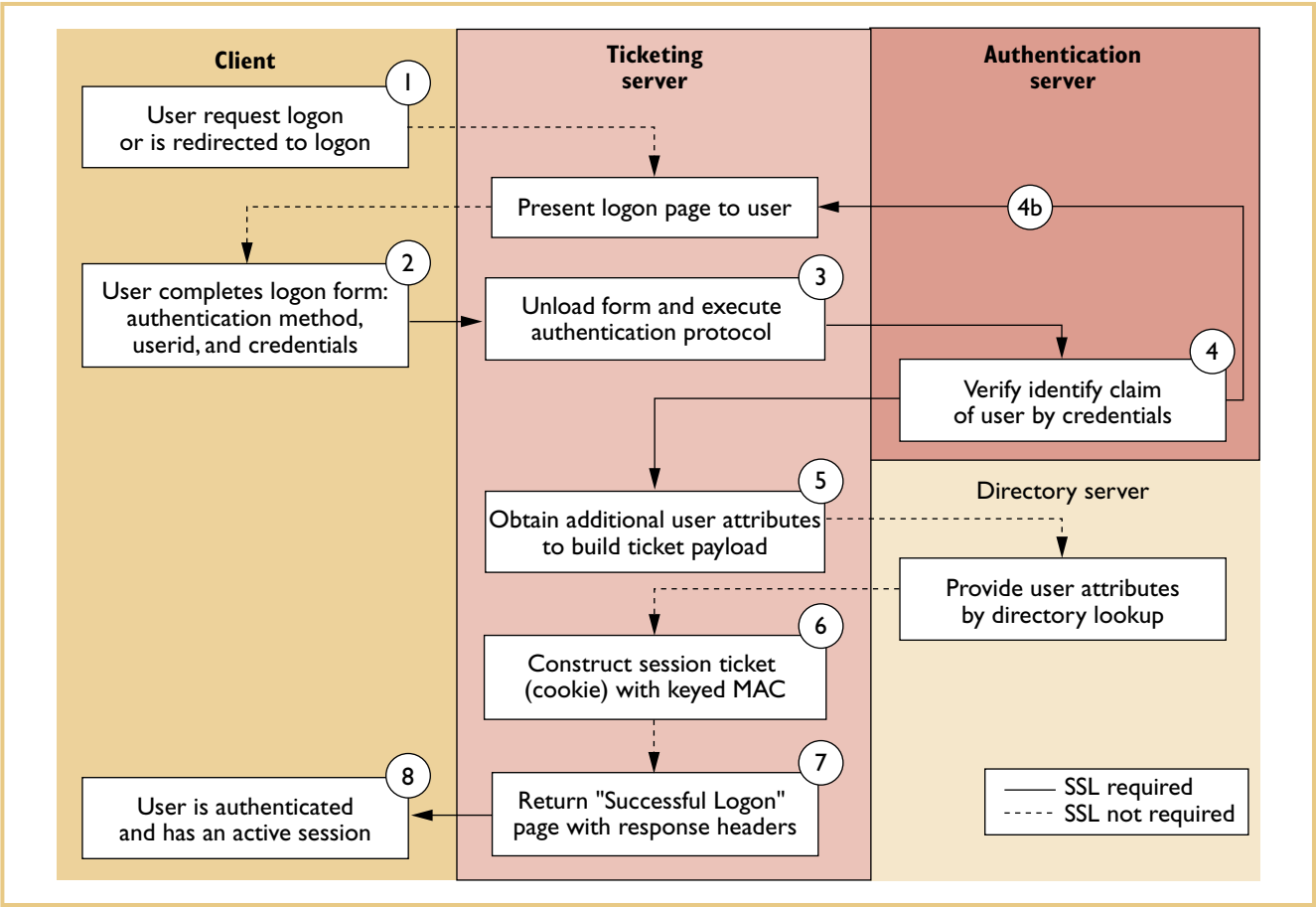

Figure 3. Ticket issuance service. The service implementation begins with a client request for logon and interacts with authentication and user profile services to open an active session.

lease does. Session time-outs are usually in the range of 5 to 20 minutes, while DHCP leases tend to have durations of 24 to 72 hours (a few environments with very short leases provide exceptions to this). If a DHCP lease is lost and renewed with a different IP address while a session is active, then the user will need to re-authenticate. (In other words, the user will have been logged off by the address change; this would be true for any socket-based services in use as well).

The use of proxy servers raises another issue related to User_IP. In the case of many users and a single shared proxy, all users appear to have the same IP address—that of the proxy. This limits the effectiveness of User_IP in binding a session ticket to a particular host. In the case of many users and an array of proxy servers with different IP addresses, the User_IP generally will not match the actual IP address of the rotating proxies. In a case where you can control the proxy systems, one solution is to activate proxy generation of the Client_IP HTTP request header and use this value instead of the host IP address.

Key Management Service

The keyed message authentication code stored as the Ticket_MAC value requires the provision of some key management services. Menezes[4] discusses the complexities of cryptographic infrastructures and methods for implementing them. Key management services required for this approach are:

- secure distribution of the MAC key to all servers requiring it, and
- MAC key renewal or regeneration.

MAC key updates cause all currently valid session tickets to become invalid. This forces users to re-authenticate, which can be irksome if key updates are frequent. In practice, a nightly key update schedule is often adequate for typical business-oriented—as opposed to military or diplomatic—security policies. In a network of servers requiring MAC key knowledge and renewal, more elaborate key distribution methods are needed to protect the key during transmission. The literature includes

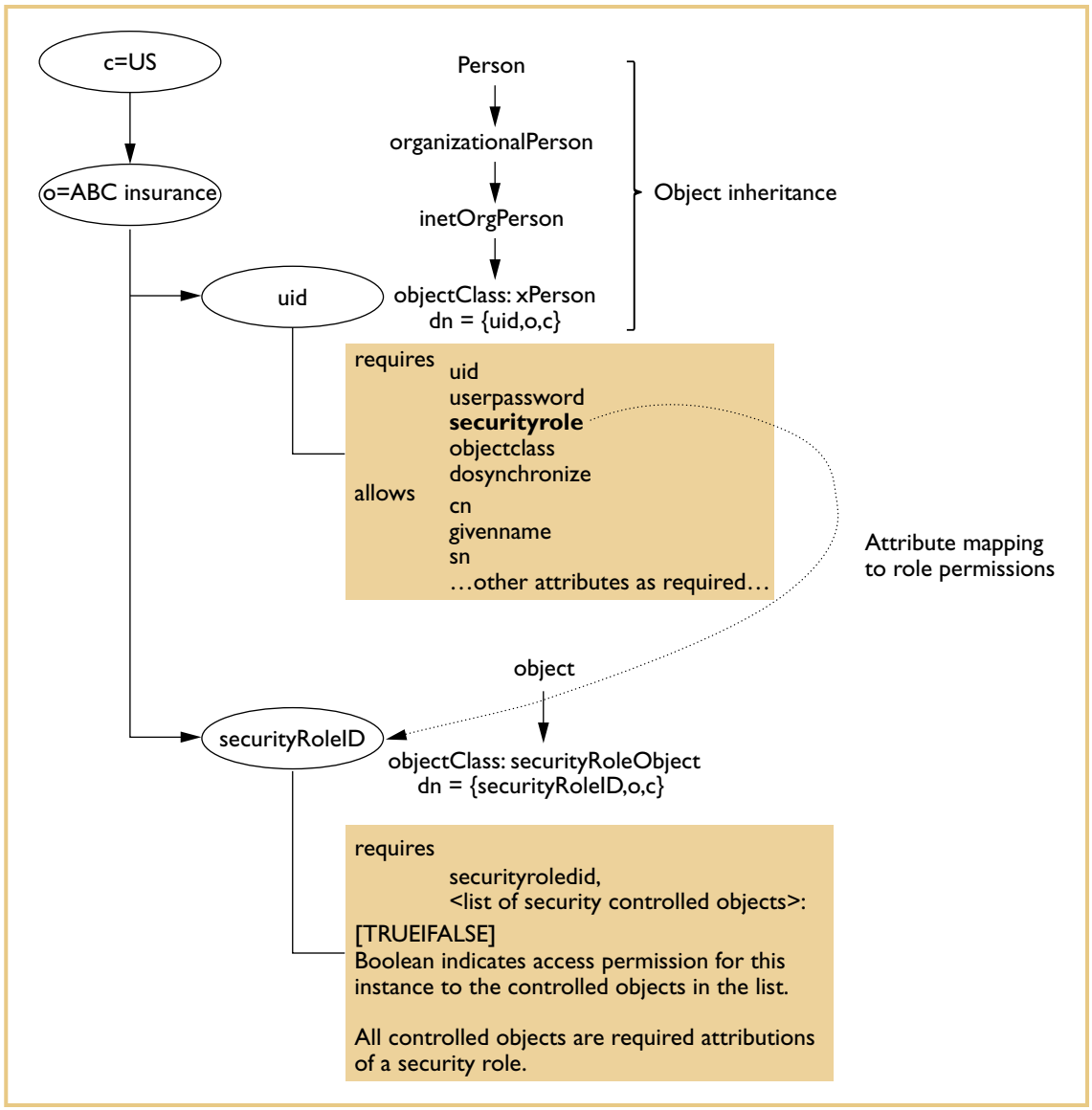

Figure 4. LDAP schema extensions for RBAC. The extensions in the directory service establish a mapping between users and security roles.

many robust key establishment, agreement, and distribution protocols.[4]

Authorization Service

This service mediates user access to resources; its primary clients are HTTP servers and other network-based, LDAP-aware applications. When a user requests resources, the HTTP server asks the authorization service if the user is authorized for them. The response is either *true* or *false*, and the HTTP server or application processes the request accordingly. In this way, the HTTP server acts like an application access firewall, where filtering rules are based on an RBAC model accessed over LDAP.

LDAP Schema Extensions and Object Instantiations for RBAC

LDAP is defined in several IETF documents. RFC 1487 (July 1993), now obsolete, was the earliest definition of a lightweight access protocol for X.500 directories. RFC 1777, released in March 1995, is known as LDAPv2 and remains the current draft standard. At the time of writing, RFCs 2251 through 2256 are proposed standards and collectively constitute what is known as LDAPv3.

The LDAP RFCs describe a network protocol for communication between directory user agents (DUAs) and directory server agents (DSAs), supported by an underlying set of data structures referred to as a directory. The directory data struc-

tures are in accordance with the CCITT X.500 standards[5] and provide a simple, object-oriented organization. The objects are not complex and do not have any executable code attached to them; they often look like simple database rows with the exception that attributes or columns may be designated as *required* or *allowed*. Objects may be designated distinct classes, the attributes of which may be inherited by any object therein.

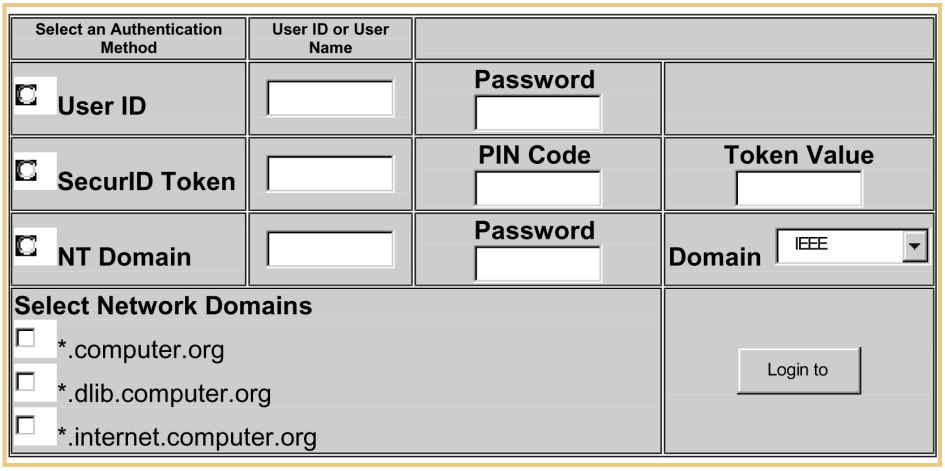

Figure 5. Initial login screen for the multiple authentication services and domains example.

Using a compliant LDAP DSA (a Netscape Directory Server in the implementation described here), the $RBAC_0$ model defined previously can be implemented by making two schema extensions. First, the default user object is subclassed to a new object class with at least one additional required attribute: securityRole. At this point, if the client has other attribution requirements, those attributes are also defined for the new default user object.

Second, the object class securityRoleObject is defined. There is one instance of securityRoleObject for each defined role in the system. This collection of objects defines the relation PA, as described for $RBAC_0$ (a many-to-many permission-to-role assignment relation). In practice, PA may be represented as a Boolean matrix of dimensions corresponding to the number of roles (rows) and permissions (columns) in the system. Each securityRoleObject then corresponds to a row of the permission matrix.

Next, users are created in the LDAP directory using the new object class. This object class inherits all the usual attributes—e-mail address, fax number, street address, first name, last name, and so on—plus any additional new attributes the client requires. A typical distinguished name (DN) for a user would have the form `uid=userid,o=organization,c=countryName`.

After all of the information access functions are identified, user roles are defined. An instance of the object class securityRoleObject with an identifier that included the role name and true or false values for each attribute that matched an information-access function was created in the insurance company implementation. Figure 4 shows the schema extensions for the directory service. In practice, identifying Web-based services and developing user roles and authorizations are challenging tasks.

Domain Ticketing and Single Web Sign-On

Because HTTP cookies contain a return domain that may include a wild-card type of specification, it is possible to use this session ticket scheme for a single Web sign-on (SWSO) capability. SWSO allows a user to authenticate once to the ticket-issuing service and obtain a session ticket that establishes the user's session and authentication throughout an entire DNS subdomain of hosts. An HTTP cookie contains a name, a value, a path, a domain, an expiration, and a *secure-only* attribute. By designating the domain to be, for example, *.computer.org, the cookie will be returned to any server in the subdomain of computer.org, such as www.computer. org, dlib.computer.org, ftp.computer.org, and so on. A single session ticket thereby provides the user's authentication and session management across a number of hosts related by domain name.

Example User Interaction

The example outlined below describes how a user sees the interaction with Web-based systems employing this approach to security. The initial login form shown in Figure 5 illustrates all possibilities for authentication service selection and domains for SWSO; in practice, this login form would be simpler.

The following steps describe this hypothetical session, illustrating the user's view of the system's security aspects:

- The user launches a Web browser and enters a URL, such as http://dlib.computer.org.
- Any cookies representing session tickets from previous sessions have expired, so no cookies

are presented in the request headers. (Expired cookies may be present in the request, but the user does not see this.)
- The Web server examines the request headers looking for a name-value pair, which serves as the session ticket. If the session ticket is invalid or expired, or if none is found, the user is redirected to an authentication HTML form, delivered over a secured SSL HTTP connection from the ticket-issuing server. This form may be delivered by the same Web server or by a different Web server dedicated to this purpose.
- The user fills in the authentication form, entering a user-ID and password, domain selections, and authentication method selection, and submits the form to the ticket-issuing server.
- The ticketing server operates as a proxy for execution of the selected authentication protocol with an authentication server.
- The ticketing server prepares a ticket for the user's session. The user ID, DN, session expiration time, absolute expiration, and possibly a client IP address form the ticket payload. A keyed MAC value is computed against the payload and appended to it; the payload and its MAC are then delivered in the HTTP response headers as cookies. The return domain of the cookies corresponds to the domain selections made earlier on the form (*.computer.org in this case). The user is now authenticated and has an active Web session with all Web servers in the *.computer.org domain.
- The user is presented with the initial navigation screen of the Computer Society's Digital Library.
- Subsequent requests against any server in the *.dlib.computer.org domain will have the session ticket cookie values in the HTTP request headers. All Web servers will extract that header and validate it using the shared MAC key. The user does not need to re-authenticate until the session expires.
- The session expires after some time, say 30 minutes. A valid MAC but expired ticket results in a re-authentication as described above. Now re-authenticated, the user continues using the Digital Library.
- Finally, the user is done with the session, and instead of letting it time out, she logs out explicitly. The logout establishes a session ticket with the expiration time so that any re-use of the computer's currently running browser will require re-authentication.

The approach secures an HTTP network environment in a simple way . . . .

Attacks and Defenses

A comprehensive approach to security must consider numerous potential attacks on network services. Relevant issues include security policy, information labeling, user administration, physical security, operating system configuration and hardening, network topologies for service locations, firewall configuration and filtering rules, intrusion detection, penetration testing, and more. The following analysis of threats is restricted to those specific to the security services identified in the service architecture presented here.

MAC forgery. Both SHA and MD5 produce a fixed number of bits from an arbitrary size input: SHA produces 160 bits and MD5 produces 128 bits. The MAC can be defeated only by a forgery. To succeed, an attacker must find a useful hash collision—a computationally daunting task. To guard against this attack, the MAC key is regenerated daily, thus limiting the time available for calculating a useful hash collision to 24 hours. This interval may be reduced as computer power available to attackers increases.

Session ticket theft. The primary defenses against session ticket theft attacks are the Login_Expires and User_IP elements. An attacker has only until the Login_Expires time to steal the session ticket and move it to another machine; in practice, this window of opportunity is usually between 5 and 20 minutes. (If the attacker were in possession of the victim's password, this session ticket theft attack would be unnecessary.)

The attacker's machine must also engage in an IP address spoof so that it appears to have the same network IP address as the victim's machine. Since both machines are active at the same time, this routing issue poses an additional problem for the attacker.

A successful session ticket theft requires that an attacker read a user's disk-based cookie file, change the victim's IP address or take the victim's machine off the network, assume the victim's IP address on a subnet such that the IP routing of the stolen address will operate correctly, and finally, access

the Web-based resources while impersonating the victim. This is a relatively complicated attack, which would generally require physical proximity to the victim's network and execution within a short timespan.

This attack is very unlikely to succeed when the cookies are *memory-resident*, that is, if they are never written to the cookie file by the browser software. This memory-only cookie treatment by browsers is not guaranteed, however. If SSL is used only for access to the ticket-issuance service and the session tickets are subsequently transmitted in cleartext over the network, the tickets are vulnerable to recovery by an eavesdropper. When the HTTP traffic is SSL-encrypted, successful eavesdropping to steal a session ticket is unlikely.

Tampering with the session ticket. If an attacker tries a different approach, such as extending the Login_Expires attribute or changing the User_IP of the session ticket, this will be detected in a Ticket_MAC computation mismatch. (The attacker does not have access to the MAC key secret used in the signature-generation algorithm, as it is protected by a firewall and operating system security measures.) A Ticket_MAC mismatch causes the server to immediately request re-authentication with the correct user ID and password, and generates an auditable event. Similarly, if an authenticated user seeks to modify some signed attribute, this will also be detected by a Ticket_MAC mismatch.

Accessing the ticket-issuance service. The function that generates the session ticket is another point of attack. Direct execution of this function would allow an attacker to revive an expired session on a machine that an authenticated user has left unattended. This function is protected by operating system methods, application server methods, and Web server access controls. No unauthenticated or direct execution or viewing of the session ticket generator is permitted.

Attacking the authentication and authorization services. LDAP-based authentication and authorization services are also vulnerable to attack. This could involve repeated attempts to guess a user-ID and password for an LDAP bind with simple password operation. Discovery of a user's password would allow an attacker to impersonate an authorized user. An attacker might also discover a system administration account, and thereby be able to change security role definitions. Further, a user may seek to modify his security role attribute to gain greater access to resources.

Conclusions

The approach described here for securing an HTTP network environment is simple and effective in a practical sense for many applications. Since the original implementation for a large insurance company, the approach has been incrementally improved in subsequent field implementations, and its effectiveness has been demonstrated in extensive evaluations, analyses, code walkthroughs, and penetration testing.

Acknowledgments

I would like to thank the project team members for their contributions: Ron Raymond, Norbert Nowicki, Eric Bazerghi, Dean Miller, Walter Kuketz, Pete McNair, Heather LeJeune, and Rob Bell.

References

1. R.S. Sandhu and E.J. Coyne, "Role-Based Access Control Models," *Computer*, vol. 29, no. 2, Feb. 1996, pp. 38-47.
2. S. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *Computer Communication Review*, vol. 19, no. 2, Apr. 1989, pp. 32-48.
3. M. Bellare, R. Canetti, and H. Krawczyk, "Keyed Hash Functions and Message Authentication," *Proc. Crypto96*, LNCS 1109, pp. 1-15; available online at http://www.research.ibm.com/security/keyed-md5.html.
4. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Fla., 1997.
5. CCITT, "The Directory: Overview of Concepts, Models and Service," CCITT Recommendation X.500, 1988.

Kurt Gutzmann is chief architect for financial services with Nervewire, a business-to-business applications consultancy. At the time the work reported in this article was performed, he was a consultant with Computer Science Corporation. Since 1992, his research interests have focused on the Internet, specifically in the areas of security, performance engineering, and distributed systems design. Gutzmann graduated with distinction from Virginia Tech, where he earned a B.S. and M.S. in industrial engineering and operations research.

Readers may contact the author via e-mail at kgutzmann@nervewire.com.