BEA WebLogic Security Framework: Working with Your Security Eco-System

BEA White Paper

Copyright

March 28, 2003

Restricted Rights Legend

Trademarks

CWP0528E0303-1A

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131 U.S.A.
Telephone: +1.408.570.8000
Facsimile: +1.408.570.8901
www.bea.com

Contents

Introduction

We have an application security crisis on our hands. N-tier architectures, enterprise application integration, and Web Services are making the application layer more complex and fragmented. As a result, we have difficultly administering security policies and bridging diverse security models. With greater opportunities to make mistakes and miss weaknesses, the chance of accidental disclosure and the vulnerability to active attack go up.

Most enterprises have adopted one of two tactics to temporarily avert this crisis. One tactic puts the responsibility on application developers. They have to discover the security postures of upstream providers and downstream requesters. Then they have to determine appropriate policies for the application layer and put security code along side business code. This tactic typically results in a confusing morass of security and business logic that is not only difficult to maintain but prone to error.

Another tactic puts responsibility on security administrators. They have to understand the detailed security operations of middleware infrastructure like the J2EE. Then they have to figure out how to statically configure application components to minimize their conflicts with other system elements. This tactic typically results in overwhelmed security administrators and very conservatively configured components. Everyone would be happier if we could cleanly separate security from application development.

The fundamental sources of these problems are twofold. First, middleware paradigms assume that the application is the center of the world. They often don't fully acknowledge the rest of the enterprise security ecology, which includes firewalls, directory servers, Web servers, authentication providers, and databases. Second, the built-in security models of most middleware paradigms don't fully support the dynamic policies necessary to meet the realities of modern business processes. Evaluating who owns an account or the strength of encryption used on a connection is cumbersome if not impossible.

The solution lies in a change of perspective—middleware paradigms own the business logic but cooperate on security. Therefore, they should gather as much information from the rest of the security ecology as possible and enable to security administrators to evaluate this information before it ever reaches the business logic. The BEA WebLogic Server 8.1 Security Framework adopts this collaborative perspective.

The Integrated Security

BEA WebLogic Server 8.1 offers an integrated approach to solve the overall security problem for enterprise applications. This approach is unique in the industry: no other application server vendor, open-source product, or dedicated security solution goes to such length to provide a powerful, flexible, and extensible security architecture. With this framework, application security is never an afterthought: it becomes a function of application infrastructure and is separate from the application itself. With it any application deployed on BEA WebLogic Server can be secured either through the security features included with the server out-of-the-box, or by extending the open Security Service Provider Interface to a custom security solution, or by plugging in other specialized security solutions from major security vendors that customer's enterprise standardizes on.

The subsequent sections of this paper define the major requirements for an integrated application security solution, and explain how BEA WebLogic Security Framework delivers them to customer's application.

Requirements

The goal of application security is rather simple: (1) enforce business policies concerning which people should have access to which resources and (2) don't let attackers access any information at all. Goal (1) causes a problem because it seems acceptable to enforce business policies in business logic. This belief is misplaced because it's much harder to change policies when enforcement occurs in business logic. Consider the analogy to a secure physical filing system. You don't take a document and rewrite it when a security policy changes. You put it in a different filing cabinet. Different filing cabinets have different keys and a security officer controls the distribution of keys. Similarly, application developers should not have to change business logic when security policy changes. A security administrator should simply alter the protection given to affected components.

Moreover, mixing security code with business logic compromises both goals (1) and (2) if developers make mistakes. When the security code in a component has a defect, people may accidentally access information they shouldn't and attackers may exploit the defect to gain unauthorized access. Of course, mistakes are unavoidable. That's why we test software. But it's a lot harder to test the security of every application component individually than a security system as a whole. The difference is somewhat analogous to reading every document in our hypothetical filing system for its fidelity to security policies rather than simply testing the integrity of the locked filing cabinets. However, we shouldn't blame application developers for mixing security code and business logic. We should blame middleware security models. Most of them simply do not support the types of policies many enterprises have such as only an account holder can access his account. Unless these security models begin supporting a much more dynamic type of security, developers really have no choice.

Middleware security models also fail enterprises in goal (2). Keeping attackers out requires a united front from all the elements in a distributed system. Cooperation is the key to this united front. Middleware sits between front-end processors and back-end databases. The middleware security system must be prepared to accept as much information as it can from the front-end processors about the security context of their requests and must be prepared to offer as much information as it can to back-end databases about the context of its requests. Moreover, it must be prepared to cooperate with special security services that work to coordinate the efforts of all these tiers. Middleware security models offer little, if anything, to support such cooperation. This failing affects many aspects of application security.

As Web Services become the prevailing model for enterprise integration and the preferred way to build interoperable applications issues concerning security become imperative. Transport-level security focuses on providing privacy to communication, but does not address the propagation of identity or control of access to the Web Service.

Authentication

Authentication is the first line of defense. Knowing the identity of requesters enables the application layer to decide whether to grant their requests and poses a barrier to attackers. Fundamentally, all authentication schemes work the same way. They offer a credential to establish identity and provide a means to verify that credential. However, there is a wide variation in the form of credentials and verification mechanisms. Each enterprise's choices of authentication schemes depend on a number of factors, including the sensitivity of protected resources, expected modes of attack, and solution lifecycle cost. In most cases, enterprises already have one or more authentication schemes in place so middleware must work with these schemes by accepting their credentials and engaging their verification mechanisms. Without this cooperation, the enterprise will have to use a lowest common denominator scheme like passwords, potentially limiting the use of such middleware to low value applications.

The problem of Web single sign-on (SSO) is even more difficult. The motivation for SSO stems from the distributed nature of Web applications. From the user perspective, a single application may actually encompass a number of different software components, running on a number of different servers, and even operated by a number of different organizations. Users don't want to have to resubmit credentials every time they click a link that happens to take them to a page running in a different location. Their experiences should be seamless. The previous problem of working with existing authentication schemes requires only understanding credential formats and integrating with verification mechanisms. However, with Web SSO, users don't even want to provide credentials in many circumstances. The trick of establishing a user's identity without seeing his credentials requires sophisticated behind-the-scenes communication between the two servers involved in handing off a user session. There are a number of proprietary solutions and some emerging standards for this communication, but it is likely that a given application may have to support multiple approaches for the foreseeable future so an open model is necessary.

Working with other Web application components involves cooperation on the front-end, but middleware infrastructure must also cooperate on the back-end. Databases have been around a long time and enterprises take database security very carefully. They really don't trust the front-end and middleware layers. If an attacker were to compromise either one of these layers, he could potentially issue a sequence of database requests that would return a large fraction of all the data it maintains. Also, if the front-end or middleware components have defects, they could unintentionally request data for the wrong user, resulting in the potentially embarrassing disclosure of private information. Therefore, many enterprises want to bind each database request to a particular end-user, including the appropriate credentials that establish the user's identity. Applications must be prepared to propagate this information.

Authorization

Once an application has established the requester's identity, it must decide whether the set of existing security policies allows it to grant the request. Typically, middleware infrastructure such as the J2EE uses a static role-based system. During user provisioning, security administrators explicitly assign roles to users and then update these assignments as conditions require. During component deployment, security administrators indicate the roles allowed to access the component. At runtime, if a request comes from a user with the necessary roles, the application grants the request. This static approach ignores the dynamic nature of many business policies. Consider the examples of policies governing bank accounts, expense reports, and bank tellers. For bank accounts, each customer should only be able to access his own accounts. For expense reports, a manager can provide an approval only up to a set amount and never for his own expenses. For bank tellers, they only fulfill the teller role when they're on duty. There are policies that are even more sophisticated where authorization depends on the combination of roles assigned to a user, as well as the content of the request. Middleware infrastructure must explicitly support these kinds of dynamic policies or at least provide enough context to specialized security services that do.

The need for dynamic authorization raises the issue of administration. We definitely don't want to force security administrators to become experts in programming languages like Java. Certainly there will be unusual situations that require some custom programming, but routinely updating the dollar threshold for expense report authorization shouldn't require it. At a more mundane level, we don't want to force them to dig through XML formatted deployment descriptors and then redeploy components to update role assignments. Security administrators need a well-designed, graphical user interface that lets them perform all of their routine tasks and most of their non-routine ones at runtime. Managing user lists and their assigned roles, changing the level of protection for components, and configuring dynamic constraints should all require just a few moments.

A more complicated headache for security administrators comes when migrating from one authorization service to another. Due to the complexity of authorization decisions, many enterprises rely on specialized services and all applications delegate such decisions to them. When it comes time to perform a major version upgrade or switch to a different service, administrators face a quandary. When do they switch over

to the new provider? The concern lies with defects or configuration problems in the new service. They don't want to switch over only to experience a massive case of improper authorizations or mistaken rejections. What they'd really like is to use both systems simultaneously and note when the old service and new service differ in their decisions, but this approach requires an even greater ability for the middleware infrastructure to cooperate with the rest of the security ecology.

Auditing

If an application could simultaneously use two different authorization services, a difference of opinion would be a very noteworthy event and administrators would want to know about it. Unfortunately, most middleware infrastructure neglects this type of security auditing. Proper auditing is not simply a matter of writing information to disk somewhere. To support their duties to *verify*, *detect*, and *investigate*, administrators need records of all security events in a single location, active notification of certain especially important events, and the ability to quickly search the records.

Security administrators are responsible for ensuring the enforcement of the enterprise policies regarding information access. Obviously, they must first specify these policies, hopefully using a productive interface as described above. Then they must verify the actual enforcement of these policies by periodically inspecting the audit trail. Government regulations or commercial contracts may require such audits. Administrators sample a representative set of transactions and track their paths through various application components to ensure the correct enforcement of security policies at each step. They need a consolidated audit trail or they'll have to spend a significant effort manually assembling logs from different locations. They need detailed records or they won't be able to determine full compliance.

Responding to potential breaches is the other primary responsibility of security administrators. Responses involve two steps, detection and investigation. First, they need the ability to specify conditions under which the security system will actively notify them. These conditions could involve transaction values, such as transfers over a million dollars, or a pattern of events, such as a spike in the number of clients connecting using weak encryption when accessing sensitive functions. Once they receive notification, administrators must be able to quickly search the logs to determine if there has been an actual breach and the extent of any damage. These searches may involve complex criteria and must execute against the live audit trail so they can track a particular attack as it unfolds. These requirements make the auditing subsystem a substantial piece of software in its own right that middleware providers must devote a substantial effort to perfecting it.

Vision

Obviously, there are many specific challenges in application security. But like the essence of application security itself, the essence of the solution is also rather simple.

1. We should have a clean, elegant abstraction between security policy and business logic.
2. We should have a simple, declarative interface for managing security policies in real time.
3. We should have an open, flexible architecture for integrating with security services.

These practices avoid the problem of mixing security and business logic, streamline security administration, and enable cooperation with the rest of the security ecology. The BEA WebLogic Security Framework delivers these critical capabilities.

Security Framework Architecture

Overview

The goal of the BEA WebLogic Security Framework is to deliver an approach to application security that is comprehensive, flexible, and open. Unlike the security realms available in earlier versions of BEA WebLogic Server, the new framework applies to all J2EE objects, including JSPs, servlets, EJBs, JCA Adapters, JDBC connection pools, and JMS destinations. In addition, the new framework is also used to provide authentication and authorization services required for the development of secure Web Services. It complies with all the J2EE 1.3 security requirements such as JAAS for objects related to authentication and authorization, JSSE for communication using SSL and TLS, and the SecurityManager class for code-level security.

The heart of the architecture is the separation of security and business logic. Business logic executes in an appropriate container, be it for a JSP, servlet, or EJB. When the container receives a request targeted at an object it contains, it delegates the complete request and its entire context to the Security Framework. The framework returns a yes or no decision on whether to grant the request. This approach takes business logic out of the security equation by providing the same information to the security system that is available to the target object. They each use this information to fulfill their dedicated responsibility: the framework enforces security policy and the object executes business logic.

When the Security Framework receives a delegated request, it manages security processing as shown in Figure 1. This processing is very flexible, with fine-grained steps not found in many systems such as dynamic role mapping, dynamic authorization, and adjudication of multiple authorizers. At each step, it delegates processing to an included, third party, or custom provider through the corresponding service provider interface (SPI). This architecture enables BEA WebLogic Server to route all the information necessary to each kind of service provider so that applications can take full advantage of specialized security services.

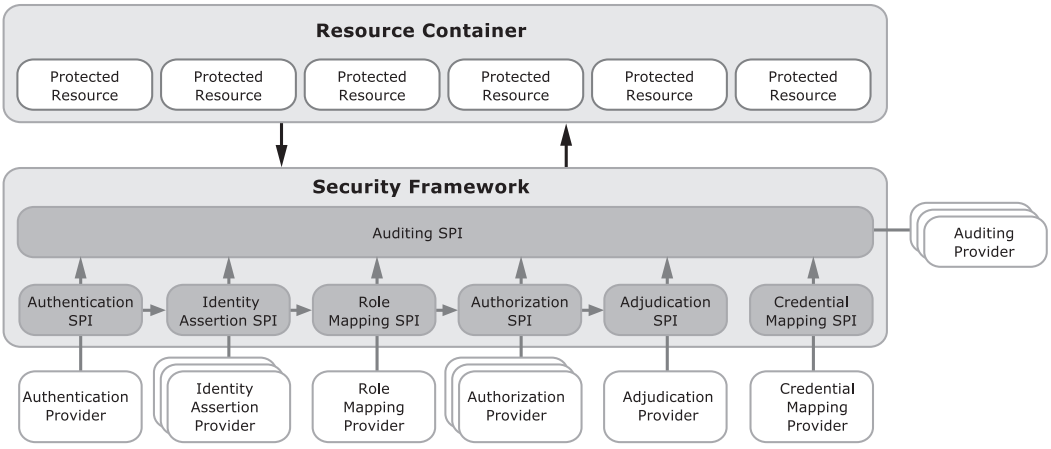

Figure 1. Security Framework Architecture.

Service Provider Integration

The Security Framework simply manages security processing. Each step requires execution by a service provider. BEA WebLogic Server 8.1 includes providers for every step, but they use the framework SPIs. Any other provider has access to the same facilities. These SPIs include:

- **Authentication**. This SPI handles the direct verification of requester credentials. The included provider supports username/password and certificate authentication via HTTPS.

- **Identity Assertion**. This SPI handles requests where an external system vouches for the requester. The included provider supports X.509 certificates and CORBA IIOP CSIv2 tokens. Because the Security Framework can dispatch requests to different providers based on the type of assertion, you can support a new type of external system by simply adding a provider for that system type.

- **Role Mapping**. This SPI handles the assignment of roles to a user for a given request. The included provider supports dynamic assignment based on username, group, and time.

- **Authorization**. This SPI handles the decision to grant or deny access to a resource. The provider included in a future release of BEA WebLogic Server will support many dynamic features such as the evaluation of request parameter values. The Security Framework supports simultaneous use of multiple authorizers coordinated by an adjudicator.

- **Adjudication**. This SPI handles conflicts when using multiple authorization providers. When all the authentication providers return their decisions, the included provider determines whether to grant the original request based on either the rule "all must grant" or the rule "none can deny".

- **Credential Mapping**. This SPI handles the mapping of application principals to backend system credentials. As shown in Figure 1, it is not part of the process leading to an access decision because it's

invoked when an object makes a request rather than when an object receives a request. The included provider supports username/password credentials and is used internally for J2EE calls and Web SSO.

- **Auditing**. This SPI handles the logging of security operations. As shown in Figure 1, it is slightly different from the other SPIs because it is invoked whenever a provider of any kind executes a function. The include provider supports reporting based on thresholds and writes all reported events to a log file. The Security Framework supports simultaneous use of multiple auditors, making it easy to integrate with external logging systems.

These clean SPIs make it possible to plug and unplug different providers as the security ecology evolves, benefiting everyone involved. BEA can individually upgrade the providers included with BEA WebLogic Server. Specialist security vendors can easily make their services available to J2EE applications by coding their products to the appropriate SPIs and many have already done so. Moreover, enterprises can quickly implement customized security processing where necessary. Instead of having to contort your security posture to suit the middleware, the middleware adapts its security processing to suit you.

From an administrator's perspective, selecting from available providers is simply a matter of pointing and clicking. Using the BEA WebLogic Server console, you expand the Realms node and then expand the Providers node. As the screen shot in Figure 2 shows, the result is a set of tabbed panes, one for each type of provider. For a given type of provider, you select one of the available provider instances and then configure its properties.

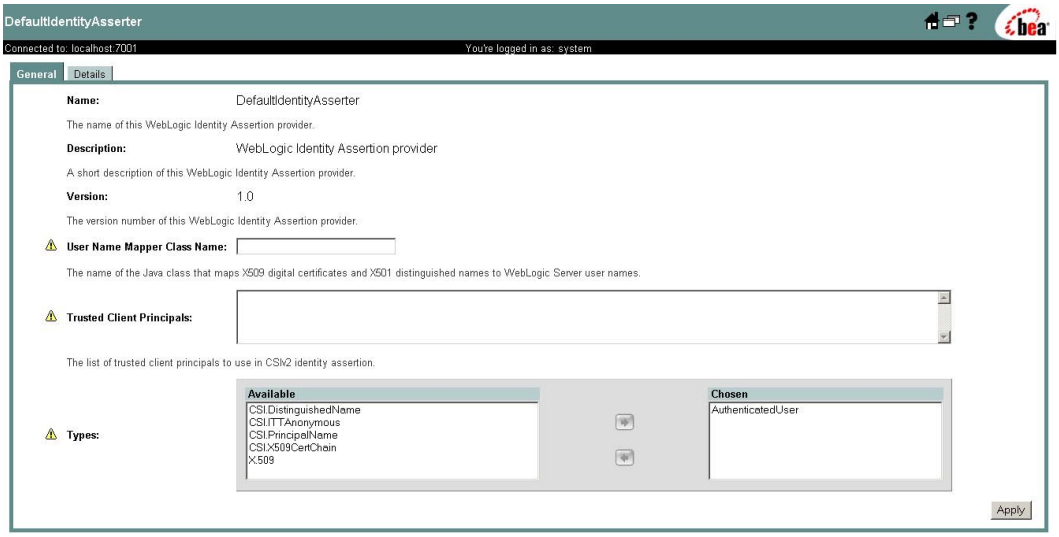

Figure 2. Administering Security Providers.

Backwards Compatibility

As described above, the new BEA WebLogic Security Framework revolutionizes application layer security. However, you may have invested a substantial effort in configuring the security realms used in WLW 6.x. You might not want to upgrade your security model immediately so the framework offers a realm adapter for backwards compatibility. Essentially, this adapter is the complete security subsystem from BEA WebLogic Server 6.x and the framework treats it just like any other service provider that implements the authentication and authorization SPIs. At server startup, the adapter extracts access control definitions from the deployment descriptor just as before. At runtime, it accepts authentication and authorization requests delegated from the framework through the corresponding SPI. From your perspective, BEA WebLogic Server 8.1 security behaves just like 6.x security. From the server's perspective, the realm adapter is fully integrated into the 8.1 Security Framework. Once you decide to move to the Security Framework, you can easily import the security information from 6.x definitions. You can even perform simultaneous authorization with the realm adapter and the Security Framework's native provider to verify proper behavior of the upgrade.

In some cases, you may be using the 6.x realm's integration with the distributed user management systems in Unix or Windows.NET. In these cases, you might want to continue using this integration for authentication but want the benefits of dynamic authorization offered by the Security Framework. Therefore, it has an option to use the 6.x realm adapter only as an authentication provider. It's interesting to note how the flexibility of the framework's SPI architecture cleanly addresses what might otherwise be a very tricky backwards compatibility issue.

Security Integration Scenarios

The Security Framework offers a lot of flexibility. To see how this flexibility addresses the application security challenges discussed above, it helps to pick a few specific examples and show how the framework helps overcome it. It some cases, the solution may not be completely finished, but the planned design demonstrates the superiority of an open framework approach.

Perimeter Authentication

In many cases, a party other than BEA WebLogic Server's own authenticator vouches for the identity of a requester. It may be the SSL layer of BEA WebLogic Server. It may be a Kerberos system. It may be an intermediary Web Service. In these cases, the third party provides a token that the application can verify. As long as it trusts the third party, it can accept a verified token as if it were the original user credential.

The Security Framework employs a very straightforward mechanism for working with such systems. All a third party has to do is put its token in an HTTP header. The Security Framework examines the token and dispatches an appropriate service provider based on the token type. If an X.509 certificate from mutual

SSL authentication comes in, the framework dispatches a provider that can verify the certificate chain to a root certificate authority and perhaps check the current validity of the certificate using the Online Certificate Status Protocol. If a Kerberos ticket or WS-Security token comes in, the appropriate provider decodes the token and performs the necessary verification.

Once the provider performs this verification, it maps the identity in the credential to a local user. The framework calls back to the JAAS with this local user, which then populates the Principal object as specified in J2EE 1.3. This approach is therefore fully compliant with the appropriate standards yet still offers tremendous flexibility. A third party provider or enterprise development team can integrate any authentication technology with BEA WLS as long as they can populate an HTTP header. Integrating BEA WLS applications with Web SSO solutions is easy because most of them, including SAML, already use cookies or HTTP headers.

Role Associations

Most application security models employ the concept of roles. Roles provide a layer of indirection between users and resources that increases the ease of administration. They are like groups, but are more dynamic. Typically, a security administrator assigns a user to a group upon provisioning and then changes this assignment only when the user's job responsibilities change. Roles change more often, perhaps even from request to request based on specific conditions. The Security Framework supports both Groups and Roles. The screen shot in Figure 3 shows how easy it is to graphically configure Groups and Roles.

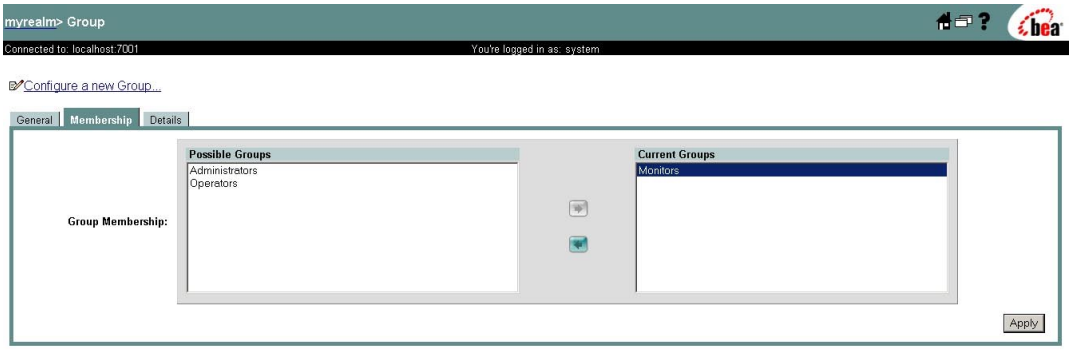

Figure 3. Administering Groups and Roles.

Administrators can set up roles so that they embody a logical identity such as Teller or name a set of logical permission such as Deposit, Withdraw, and Transfer. It's really a matter of design. The first approach is more focused on the logical role of the user and the second approach is more focused on the logical role of the resource. The Security Framework distinguishes between globally-scoped roles, which apply to all resources in an installation, and resource-scoped roles, which apply only to specific resources. Globally-scoped roles are intended primarily for managing different levels of administrative privileges. Administrators will mostly configure and manage resource-scoped roles.

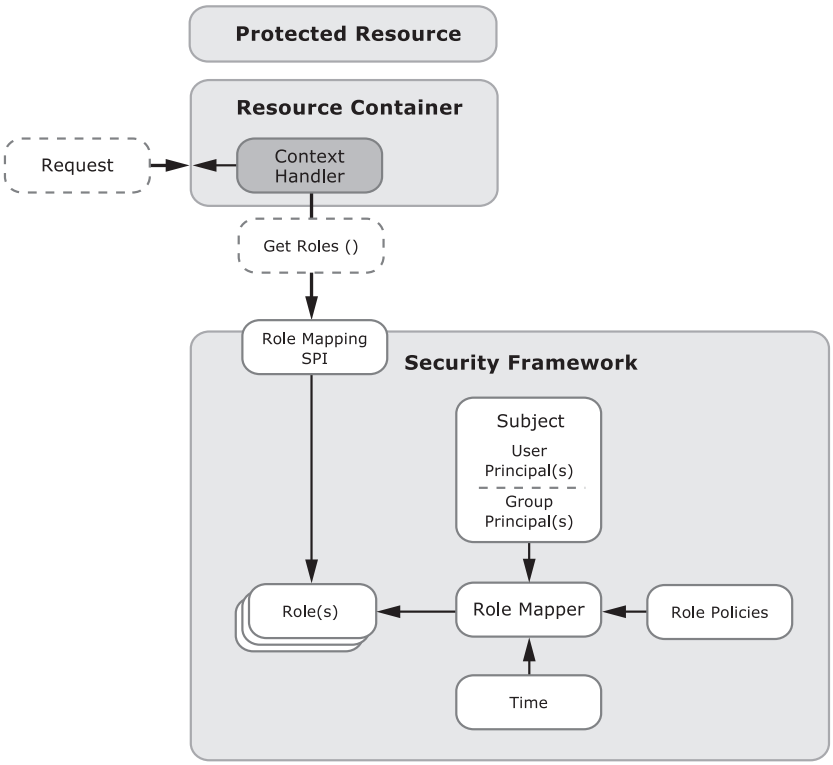

Figure 4. Dynamic Role Mapping Process.

The Security Framework enables service providers to dynamically assign roles based on context. Figure 4 represents this process. The included provider can take into account username, group, and time. For example, suppose a bank had an AccountManagement EJB restricted to users with a Teller role. A user would fulfill the Teller role if they were a member of the DayTeller group and the local time was between 8am and 6pm. Administrators could also use this time feature to set up role assignments that automatically expire, which might be especially useful for very sensitive information such as human resources data. Figure 5 is a screen shot that shows how easy it is to set up this dynamic assignment. The role mapping SPI actually supports the use of additional information such as the parameters of the method call. Therefore, custom providers could offer even more flexible role mapping Consider the case of a CFO and expense accounts. A custom provider could grant the CFO an Approver role unless the Employee parameter of the request were the CFO himself. That way, he couldn't approve his own expenses.

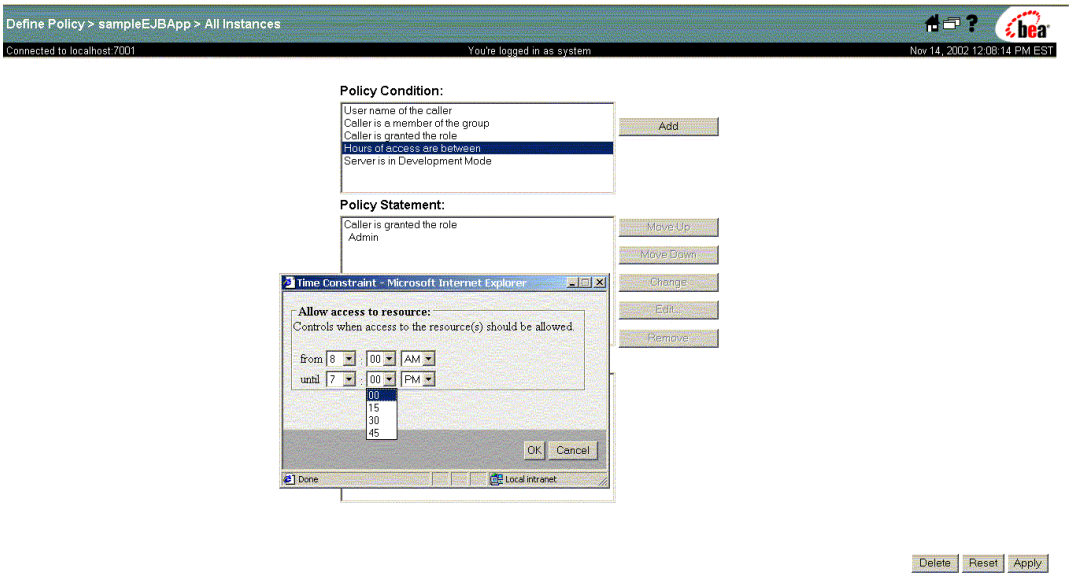

Figure 5. Administering Role Mapping.

Credential Mapping

As discussed above, enterprises often want to tie each request of a backend database, packaged application, or legacy system to the ultimate user. Therefore, when a J2EE object accesses a backend system on behalf of a user, it has to supply the appropriate credentials to the system. The basic problem is mapping a J2EE Principal to backend system credentials. The included service provider solves this problem for the most common case of username/password credentials. Each BEA WebLogic Server instance has an embedded LDAP directory in which it can store the encrypted username/password pair for every valid combination of Principal and backend system.

The increasing attention paid to security may create the need for more sophisticated third party or custom providers. The latest versions of some databases can use Kerberos tickets. Also, the latest versions of some packaged applications can use various forms of strong authentication and many mainframes use RACF. The Security Framework can easily accommodate third party or custom providers that support these alternatives.

Parametric Authorization

One of the classic application security problems is making an authorization decision based on the content of the request or the target object. Approval thresholds are a common case where you want to evaluate the value of these parameters. First, you would create a set of roles such as Manager, SeniorManager, and Director. Then, you would create a set of policies that authorizes approval requests for each role based on the value of the Amount parameter, such as $5,000 for a Manager, $10,000 for a SeniorManager, and

$20,000 for a Director. Most middleware does not currently address this issue. A forthcoming version of the included authorization service provider will allow such decisions based on the content of the request. Figure 6 represents this process. In fact, the authorization SPI already supports the use of method call parameters in access decisions so you could build a custom provider with these capabilities today.

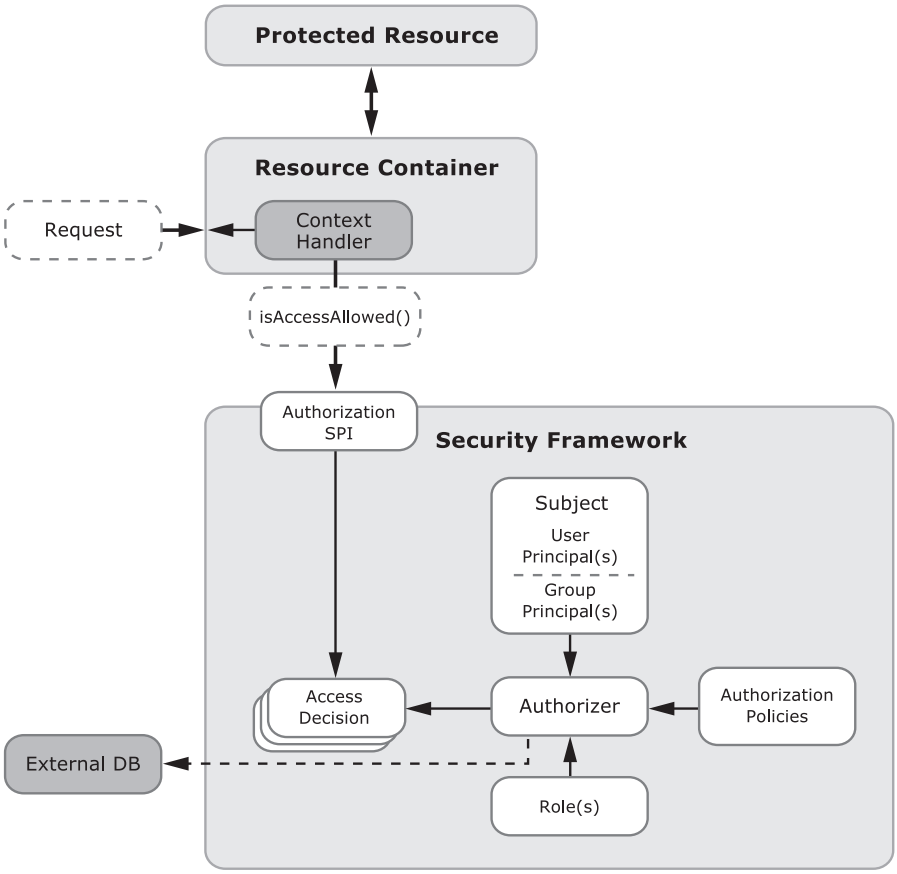

Figure 6. Parametric Authorization Process.

Authorization based on the content of the target is a little trickier. For example, you may want to inspect an Account object to get the value of AccountHolder before you decide to authorize a withdrawal. Unfortunately, in the general case, this type of visibility can break encapsulation and present a security vulnerability. If the security system can access any data in the system, it presents a tempting target for compromise. It will soon be possible to write custom code to perform this type of operation in select cases. A future version of the SPI will enable you to access context besides the method call parameters, such as the EJB primary key. You could then create a very specialized provider that examined the primary key of the Account object targeted by a Withdrawal request to determine the correct row in an account database. The provider would then make its own call to this database, using special credentials of course, to retrieve the account holder and compare it to the Principal. This solution might require some manual mapping of account holder values to Principal values, but it would work if used judiciously. The dotted line from the authorizer to the external database in Figure 6 illustrates the possibility of such a solution.

Note that you would have to write almost the exact same code to perform this check in the Account object itself. However, the service provider approach partitions the security logic and the business logic. Different people can maintain the two types of logic and changing one does not introduce the risk of potentially breaking the other. This flexibility is important if you consider the actual complexity of bank accounts where there are minor accounts, joint accounts, and business accounts. Maintaining the security policies for such an application could be a full time job in and of itself.

Securing Web Services

Using a combination of the following three techniques, customers can build Web Services that enforce security at all levels: SOAP message body elements, transport, and control of access.

Message-based security—Data in a SOAP message is digitally signed or encrypted. Message-based security in BEA WebLogic Web Services uses the WS-Security specification. This specification provides three mechanisms: security token propagation, message integrity, and message confidentiality. These mechanisms can be used independently (such as passing a username security token for user authentication) or together (such as digitally signing and encrypting a SOAP message). Message-based security provides end-to-end security, ensuring that a message is secure even when using intermediaries, caches, or queues, in contrast to connection-based security that provides point-to-point security between two endpoints.

Connection-based security—SSL is used to secure the connection between a client application and the Web Service. Connection-based security uses protocols such as SSL to protect against eavesdropping and ensure the integrity of data sent between the Web Services client and server. While it provides a secure transport for SOAP messages, connection-based security lacks the granularity and flexibility of WS-Security.

Authorization—Specifies the rules that control access to a Web Service. This kind of security protects the implementation of the Web Service just like any other component deployed on BEA WebLogic Server. The security framework handles control of access to the Web Service after the SOAP message is received and decoded.

Conclusion

The BEA WebLogic Security Framework does not impose a rigid security model that hinders security integration with other system elements and forces the costly workaround of mixing security code with business logic. Instead, it adopts an open processing model so that application components can seamlessly cooperate with the rest of the enterprise security ecology. Moreover, its processing model delivers a clean abstraction of policy enforcement from business logic that lowers the cost of administrating security policies and decreases the chance of security breaches.

Both application developers and security administrators benefit from the BEA WebLogic Security Framework. Developers no longer have to shoulder the responsibility and potential embarrassment of mixing application and security code. Administrators do not have to become experts in middleware paradigms to meet security requirements. When someone has to write special security code, it only has to be done once—everyone can use and easily maintain it.

The key to the BEA WebLogic Security Framework's benefits lies in its open service provider model. Third party security vendors can easily integrate their solutions with BEA WebLogic and enterprises can quickly create custom security modules. Most importantly, an open model means that enterprises do not have to wait for the middleware vendor to adopt new security technologies because there are plenty of hooks for specialists to hang future innovations.

About BEA

BEA is the world's leading application infrastructure software company, with more than 13,000 customers around the world including the majority of the *Fortune* Global 500. The BEA WebLogic Enterprise Platform provides an industrial strength and easier to use software foundation that makes an enterprise more agile, productive, and connected, resulting in dramatically increased IT productivity and faster time to value. BEA's platform is the de facto standard for more than 1,700 systems integrators, independent software vendors, and application service providers who partner with BEA to ensure the successful deployment of customer solutions. BEA can be found on the Web at *www.bea.com*.

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131 U.S.A.
Telephone: +1.408.570.8000
Facsimile: +1.408.570.8901
www.bea.com

CWP0528E0303-1A