An Oracle White Paper
March 2010

Think Globally, Act Locally: Policy-Driven Security for a Trustworthy Service-Oriented Architecture

ORACLE

Executive Overview

The many benefits of a service-oriented architecture (SOA), trumpeted by technology vendors for several years, are driving enterprises to deploy a production SOA framework on a wide-scale basis. However, many of the foundational characteristics that make SOA so appealing from an architectural perspective also introduce security issues. While realizing substantial advantages from exposing services and data to business partners, subsidiaries, and divisions, organizations deploying SOA are taking on additional security risk.

Introduction

SOA has become the de facto standard for distributed enterprise application development. The reason for this is simple: SOA allows businesses to unlock the value of data and applications previously sequestered within the enterprise and across the supply chain. By exposing those data and applications as services, organizations are reducing time to market and increasing ROI for their IT initiatives—while gaining the ability to react quickly to emerging threats as well as new opportunities.

Security concerns continue to be a limiting factor in the expansion of SOA. Accompanying the evolution of SOA is the emergence of regulations for privacy and data security. The financial and legal ramifications for lapses in security are now material and potentially severe. In response, industry groups have developed many technologies and specifications to address SOA security. But SOA, and in particular SOA security, is not merely a technology that can be implemented and deployed. People and processes have to be factored in as well. Therefore, organizations continue to confront challenges with bringing the tools and techniques defined in those standards to bear on real-world SOA. Today, organizations employ governance procedures to regulate operations in areas such as finance and research to comply with corporate policy. Those governance procedures must be transferable to IT.

To address these issues, organizations must adopt a policy-driven approach to security. The goal of the policy-driven approach is to free developers and administrators from focusing on security technology implementation, integration, and configuration. Policy, combined with governance, enables organizations to think globally about security policy, while letting the runtime system act locally to enforce security policy. This white paper addresses SOA security issues, evaluates industry response to the problem, and discusses how governance and a policy-driven approach foster a trustworthy SOA.

Addressing Service-Oriented Architecture Security

In addition to many tried-and-true network layer attacks, the following characteristics contribute to SOA's vulnerability:

- The primary transport for SOA is Simple Object Access Protocol (SOAP) over HTTP. SOA application requests and data flow uninspected through port 80 on a standard network firewall. The network perimeter, long secured by the firewall, is no longer the trust boundary it once was.
- The specifications for describing SOA interfaces, including Web Services Definition Language (WSDL), Web Services Policy Framework (WS-Policy), and Web Services Metadata Exchange (WS-MetadataExchange), are helpful for rapid application integration. In the wrong hands, they provide a blueprint for an attack against the service they describe.
- Loosely coupled applications intrinsically lack tightly coupled security mechanisms; thus, they invite unplanned—and potentially insecure—reuse.
- The complexity of SOA systems can lead to the inadvertent exposure of sensitive operations and data through unmanaged endpoints, often called rogue services.
- The heterogeneous, multivendor environments typical of SOAs make it difficult to integrate security mechanisms consistently, if at all. Change management, root cause analysis, monitoring, and auditing in these environments are formidable tasks.

Approaches to Application Security

Traditionally, enterprises take a bottom-up approach to security for distributed applications, treating security as a technical problem rather than a business problem. Recall the days when you asked, How do I implement a public key infrastructure to secure my Web applications? Today, such a question, and those like it, have changed to, How do I allow my business partners to securely access my services, while adhering to our business agreements and complying with relevant regulations? Security has increasingly become a corporate issue, as industry and government regulations around IT security have not only proliferated but also strengthened. To date, three approaches have driven application security.

Security in the Application

Until recently, there have been few alternatives to implementing security processing directly within the application code. There was no clear separation between the functional code (that which processed and displayed data, for example) and the security code (that which verified identity, checked digital signatures and encrypted data, for example).

Putting security processing in the hands of developers has one benefit. Tightly coupled security implemented in procedural code enforces a strict contract for interacting with an application, and is useful for one-to-one application integrations. Drawbacks to this approach, however, outweigh that lone benefit. One drawback is that it tends to be inflexible. In the event of changes—for example, to

support a new partner or to react to an emerging security threat—a full development cycle is necessary to design changes, modify code, perform regression and integration testing, and complete other deployment procedures. These procedures detract from enterprise agility, negatively affecting time to market and ROI for IT initiatives.

Security in the Application Container

Another solution has been to delegate security processing to the application container running the service. Application containers provide platform-specific tools for delegating authentication, authorization, integrity, and confidentiality to the platform itself. This approach could work well in a unified environment. However, in a multiplatform, heterogeneous environment, the complexity of managing and synchronizing configurations for diverse containers can be daunting. Although the application code remains untouched when the security configuration changes, the introduction of many of the complexity issues associated with integrating security directly in the application takes place at the administrative level, requiring the management of security through distributed configuration files.

Security Gateways

Yet another solution to this problem has been to deploy security gateways. A security gateway provides application awareness at the network edge and applies security processing on behalf of back-end services. This relieves developers of the burden of implementing security code and enables security administrators to focus on security configuration. Security gateways in the form of hardware appliances provide the added benefit of wire-speed XML transformations and accelerated cryptographic processing.

A gateway can ensure the proper authentication of all traffic entering and leaving the service network, authorized and otherwise secured. Gateways also address security issues such as content injection and denial-of-service attacks before any messages reach the application platform. However, a security gateway cannot secure the service consumer or the service provider. Why is this important? Surveys show that insiders initiate as much as 90 percent of all attacks. To minimize this risk, a "need to know" approach is necessary for SOA intermediaries, even those on the internal network. Although a security gateway is helpful for securing the network edge, a complementary solution is required to implement limited trust relationships in a SOA by providing security at the service endpoint itself.

The general trend has been to move security out of the application and onto the network. As noted, this is essential for protecting against certain types of attacks, while at the same time gaining the benefits of hardware-based XML processing. The problem, then, becomes how to rationalize and manage the loosely coupled relationship between security on the network and applications themselves. The solution to this problem lies with security policy.

Policy and Governance: Aligning Information Technology with Business Drivers for Security

When rolling out SOA, it is important to approach security holistically—taking into account corporate trust relationships and business goals as well as threats and vulnerabilities. In many cases, regulations such as the Sarbanes-Oxley Act, the Health Insurance Portability and Accountability Act (HIPAA), and the Payment Card Industry Data Security Standard will influence application risk profiles.

Whereas application security was once a "nice to have" feature for many organizations, in today's increasingly regulated environment, security can have a direct financial, not to mention legal, impact.

For these reasons, organizations should not allow technologists and point solutions to drive or constrain security strategies. Instead, business analysts, compliance officers, and security officers must collaborate to define security requirements at the business level. These stakeholders require assurance that those security requirements will propagate throughout the organization: at the front desk, in the call center, throughout the extended enterprise—all the way down to the level of application security.

Application security policy needs to reflect corporate policy. For example, a corporate policy states that only administrators and doctors can view electronic protected health information (ePHI) data covered under HIPAA. The embodiment of that corporate policy in an application security policy expresses the requirement that all ePHI must be encrypted using a particular algorithm with a particular key size. Compliance refers to how accurately a corporate policy is implemented and enforced by the many security policies deployed through the organization.

Knowing to what degree the applications comply with corporate policy is a challenge in a complex environment.

In terms of application security, a policy expresses the capabilities and constraints of an application. Codified in specifications such as WS-Policy and Extensible Access Control Markup Language, the policy-driven model abstracts the definition of the security policy from its enforcement in the system.

Challenges for policy-driven SOA security include the following:

- How are security polices created, managed, provisioned, and maintained throughout the service lifecycle?
- How centralized is policy management?
- To what degree does the behavior of the SOA as a whole reflect the intent behind the policies? Can a compliance officer prove that to an auditor?

To address these challenges, the processes and tools used to define and manage security policies should integrate broadly with existing IT procedures. *Governance* is the set of procedures used to manage IT projects from a business perspective. Governance provides guidelines, control mechanisms, and traceability for policies and the way policies evolve over time. Practically, governance specifies who defines and reviews policies, and when and where those policies are applied. Thinking in terms of security policy, integration with a governance solution is critical to a successful and secure SOA deployment.

Although many software security providers are implementing policy-driven security, no one is adequately governing it. Usually, the platform limits the tools provided for remotely managing policies. In the heterogeneous, multivendor "ecosystem" environments that are common to SOA, a different approach is required for providing comprehensive governance. SOA policy provisioning and management are traditionally associated with service registries and repositories. In this arrangement, service provider information is published to a registry, where service consumers can look up and obtain additional information about the service, including interface definitions, endpoints, and policies.

Updating the service registry is usually a manual step in the deployment process. Developers and administrators must coordinate their activities to ensure that the registry and repository provide an up-to-date, accurate portrait of the runtime system. For a variety of reasons, these manual procedures often reintroduce the same class of problems as the implementation of policy in code—a delta between the desired conditions and the actual result. Divergence between the intended design and the runtime reality can result in security holes, inaccurate audits, and a variety of unpleasant surprises.

To address this issue, a SOA governance solution should be capable of automatically classifying services according to high-level business or operational requirements:

- Is the service being deployed to the quality assurance (QA) or production team?
- Is the service externally exposed to business partners?
- Is the service part of a highly profitable business process?
- Is the data exposed by this service subject to regulatory control?

One approach is to leverage metadata associated with services. Organizations already manage high-level descriptions of IT assets. Why not leverage that data to define and provision security policies? Rather than manually applying policies to services on a one-to-one basis, why not apply policies to services in a general and descriptive way, based on classifying metadata? By taking this approach, the loose coupling between security policy and application logic becomes manageable.

A Policy-Based Approach to Service-Oriented Architecture Security

To provide true runtime governance, a solution must support security-related scenarios such as the following:

- For all logistics services in QA, use testing cryptographic credentials.
- For external procurement services consumed by Platinum customers, require the "Platinum" role, require a strong authentication, and forward the user's credentials to the back-end service.

Such a policy-based approach has the added benefit of easing integration with existing infrastructure. First, register all security-related infrastructure with the governance system: identity management systems, user stores, and key stores. Then, transparently use the registered infrastructure within higher-level security policies for authentication, authorization, integrity, confidentiality, and nonrepudiation. To avoid inconsistent policy enforcement and unmanaged endpoints, the policy must synchronize automatically and continually with the appropriate service endpoints. In such a system, changes in

metadata result in changes to applied policies. Move a service from QA to production and the system will automatically provision that service with a new set of production policies, while deprovisioning the set of policies designated for QA services.

Runtime governance delivers value not only by decoupling security coding from application development, but also by decoupling policy definition from its implementation or configuration in the infrastructure. Intelligent policy provisioning makes it possible to rationalize security policy across a heterogeneous SOA.

Enforcement of SOA security policies must be consistent from end to end, from the service consumer to an arbitrary number of intermediaries to the service provider endpoint. For this reason, SOA requires standards-based, interoperable security mechanisms that enable consistent security policy enforcement at every point, and that guarantee persistent integrity and confidentiality from the client to the managed endpoint.

Secure Service-Oriented Architecture Endpoints: Last-Mile Security

Last-mile security refers to communication between a management intermediary and a service endpoint—the final leg of a SOA transaction. Last-mile security protects the service at the endpoint, as the last line of defense, and provides in-depth defense for organizations that have already deployed a gateway solution. The agent architecture ensures no circumvention of security mechanisms to contact protected services in an unmediated fashion. There is no way for an attacker to sidestep the security policies enforced by the service.

Likewise, enforcement of security policies on outbound messages takes place before the message hits the network. Stripping or encrypting sensitive data can occur before network exposure. In the absence of last-mile security, messages move across the network to the management intermediary before any security processing takes place. This leaves the message vulnerable from the time it leaves the endpoint until the intermediary processes it.

Last-mile security is a critical piece of the SOA security puzzle; however, it does not in itself provide end-to-end security. To account for the initial leg of a SOA transaction, you need to understand the concept of first-mile security.

Trusted Service-Oriented Architecture Clients: First-Mile Security

Often overlooked are service consumers, or SOA client applications, as the creation of sophisticated composite services gets more attention. However, who will make use of those services without the accommodation of SOA clients? An often overlooked fact in policy-driven systems is that enforcing security policy on the service immediately places a burden on the client to implement corresponding security features.

First mile security means the transactions that span the client application, or service consumer, and the service network. The key problem in first-mile security is enabling the client application to execute the policy enforced by the service. The client should be shielded from service and policy updates of any

kind. First-mile security applies to all clients that consume enterprise services, whether those applications are local or remote.

To support dynamic client enablement, a service must be able to communicate its policy requirements to prospective service consumers. The client application must then be able to execute those requirements. Additionally, there is a need for policies that divulge only the information needed by the client, keeping secret additional processing, such as auditing, that might be taking place on the enforcement point. This policy conversation must occur between products from heterogeneous vendors, and formatting should be according to the WS-Policy and WS-MetadataExchange specifications, allowing clients to consume this policy in a standards-based, interoperable fashion.

The benefit of this architecture is that modification of security policies can take place on the service side without the fear that a policy update will break all clients that consume the service—a constituency that will expand as your SOA becomes more successful. This feature becomes increasingly important as your SOA evolves and more clients begin consuming your services.

Nevertheless, client applications present additional challenges. An unmanaged client is often unpatched; lacks a firewall; and is, therefore, the target of malware distributors around the globe. In short, the client can be a security administrator's nightmare. In today's world of Trojan horses and keystroke loggers, it is sometimes necessary to ensure that a request comes from an approved, enterprise-managed machine. Otherwise, a password or a digital signature, although perfectly valid, could come from the wrong source—an imposter with the intent to undermine or defraud your business.

A client agent must be able to consume some form of policy description dynamically from a managed service. The policy indicates which security processing to perform on the request and which security features to expect on the response. The client agent then executes policy automatically on behalf of the client application, seamlessly providing the client application with secure access to the target service.

By providing support for multiple authentication methods on a single request, client-side agents can enable "trusted clients." A client policy can specify both username/password and a digital signature. The username/password identifies the user, whereas the key used to generate the signature might be bound to a specific machine. This strong authentication ensures an unmanaged computer cannot use a stolen password.

Security Across the Service-Oriented Architecture Lifecycle

To reap the full benefits promised by SOA, enterprises must address security from the outset. Security cannot be an afterthought, applied only when services move into production. The policy-based approach can provide security across the SOA lifecycle, from development through QA/staging and on to production. This might seem counterintuitive. Why do developers need access to security policy in development?

When creating applications, developers need visibility into services available for reuse, monitoring tools that provide the data that enables performance optimization. Developers might be composing applications by leveraging existing services, which in turn might enforce security policies of their own.

In QA, tools are required to assist in capacity planning and might be required to optimize performance for applications that are performing security operations such as authentication or encryption. From the management perspective, services must move through the cycle with governance. Security across the SOA lifecycle prevents the inadvertent exposure of data and services by ensuring that development activities always take place within the context of the overall governance tools and processes implemented within an organization. SOA security needs to be future-proofed through dynamically adaptive mechanisms, while remaining backward compatible throughout the evolution of the ecosystem. As new services and partners come online—as new clients consume enterprise services—the application and enforcement of security policies must be performed in an intelligent, consistent, and rational fashion that enables end-to-end SOA security.

Security Policy: Codifying Best Practices for a Secure Service-Oriented Architecture

A policy-driven security model must provide a foundation for best practices, while also enabling customization and extensibility to reinforce institutionalized processes and taxonomies seamlessly. Therefore, it is important to keep in mind that the policies described here, although powerful, are really a starting point for security administrators to develop a policy library that responds to the unique security challenges confronted by their organizations. A SOA governance solution should provide support for the following security mechanisms in the form of an extensible security policy library.

Authentication

Enterprises must ensure that users prove their identity before granting them access to protected services and regulated data. However, it can be challenging to repurpose authentication mechanisms designed for human-based interaction to support SOA's machine-to-machine environments.

It is also challenging to match the strength of an authentication mechanism to the risk profile of the application. Too strong a mechanism will incur needless expense; too weak a mechanism exposes an application to compromise. As application reuse accelerates, more users seek access to more services.

User Stores, Access and Identity Management Systems

When securing a SOA deployment, a turning point is defining strategy around user stores. Typically, a user store of one sort or another is in place and might include access and identity management solutions. The question is how these solutions will integrate consistently across a heterogeneous SOA. Moreover, because deployment of these systems is typically for human users, can repurposing enable the support of machine users?

X.509 Certificates and Digital Signatures

The Web Services Security (WS-Security) specification has had the effect of making X.509 digital certificates a cornerstone of SOA authentication. Digital signatures do away with vulnerable passwords, providing instead the benefits of publicly verifiable private passwords; in other words, users can prove

their identity without ever revealing their password. Digital signatures also provide persistent integrity—a permanent mark that proves modification of a document's contents since the application of the signature. (Some say this makes documents tamperproof—it does not. You can tamper with digitally signed documents. The difference is that, with a digital signature, you can demonstrate document tampering. However, you cannot show where or how the modification occurred.) Such integrity is invaluable when performing audits of any sort. The binding of a private password to an identity provides the chain of evidence needed to prove unequivocally which entity performed an action.

The complexity associated with deploying a public key infrastructure tends to trump the claims for digital signatures. Although the security of the RSA digital signature algorithm is unimpeachable, the system's vulnerabilities derive from all of the processes around credential provisioning. First, only authenticated individuals or machines should receive certificates. There are a variety of approaches; yet vulnerability is always an issue of process, rather than technology. Second, secure the private key itself. Although computationally expensive, the direct attack or cracking of private keys can occur. However, it is far easier to steal or misuse the private key itself, even if the key is stored securely in hardware. For example, Trojan horse software on a machine could be capable of accessing a key to produce seemingly authentic documents, even if the key is stored on hardware. Again, it is the infrastructure around the private key—and not the private key itself—that determines the trustworthiness of the signatures it produces. This tends to make X.509-based security an expensive proposition.

With these caveats, X.509 digital certificates and RSA keys play a critical role in SOA. Simply because a compromise of digital signatures can occur does not mean that they do not play a central role in building a trustworthy SOA. In SOA, the message is the application, because it is the message that drives the composition of the various interacting services. Therefore, message integrity is a critical feature of a trustworthy SOA. If an intermediary can intentionally or unintentionally modify a critical piece of a request or response message, the entire composite application is at risk. For this reason, SOA governance solutions should make it easy, and essentially transparent, for SOA security policies to make use of digital signatures as part of normal processing.

Signatures provide persistent and strong authentication and integrity. With reasonable security and procedures, a digitally signed request or response message provides a high degree of assurance that data comes from the correct party, unmodified in transit. Authentication of a digitally signed, archived message can occur and be validated offline for auditing purposes. This provides for strong chains of evidence tying identities to actions long after a transaction has been completed—a key to auditing compliance in a complex SOA.

Identity Assertions

Assertions provide a general-purpose mechanism for sharing information about an entity within a distributed system. Conceptually, an *assertion* is any statement made by one entity about another entity. For example, an assertion might be, "I am Bob, and I am an expert carpenter." The value of the assertion depends on the integrity of the assertion itself and the trustworthiness of the one asserting it. If I know Bob well, he makes the assertions in person, and I trust him, then the assertion is valid to me. If I do not know Bob, I require a different type of assertion, one from a party I trust for

contractual or institutional, rather than personal, reasons. For example, "We, the Bonded Carpenters Certification Association, assert that Bob is an expert carpenter." If we can prove the integrity of this assertion—that is, if we can unequivocally show that the assertion comes from the Bonded Carpenters Certification Association—then, even if we do not know Bob, we can confidently sign contracts with him to build a deck in our backyard.

Assertions in SOA work the same way. Because applications in SOA are loosely coupled, they require flexible mechanisms for sharing identity information. One service in SOA cannot directly trust all of the other services and consumers. Therefore, we must delegate the trust for many services and users to an authority. This is "transitive trust." I trust you because the authority, or someone else I trust, trusts you.

A practical way of looking at the problem is to ask, If I am authenticating all users at the perimeter via my security gateway, why do I need to reauthenticate those users at the service endpoint? It is far better to authenticate the user at the perimeter, and then create an assertion that shares with back-end services.

Not all SOA services will support the same authentication mechanisms, and it is sometimes impractical to reauthenticate an identity repeatedly. Identity sharing enables SOA applications to interact efficiently and securely.

To address these needs, Security Assertion Markup Language (SAML) is a standard that enables the reuse and propagation of identity information across multiple applications. Services are capable of consuming SAML assertions generated by another application. For example, a service intermediary might authenticate a service consumer using a digital signature. In turn, the service intermediary can generate a SAML assertion that contains an authentication statement. The service intermediary can then forward the SAML assertion to a service endpoint, which can simply use the authentication statement, rather than going to the trouble of reauthenticating the service consumer.

Authorization

Authorization is distinct from, though dependent upon, authentication. Once positive identification of a user occurs, the system must determine what operations the user is entitled to perform. You can imagine a SOA architecture where these two procedures are distributed: a security gateway performs authentication, and then forwards an assertion containing authentication information and additional information (such as roles or other attributes) to a back-end service. The back-end service can then use that information to make an authorization decision autonomously.

Flexible authorization is a critical component of SOA, due to the potential for unintended reuse of services. When a service is deployed in SOA, you might find it adopted by very different groups of users—some more trusted than others, some completely unknown and unaccounted for when the service was initially deployed. For example, you might release a service to support your internal administrative users, only to find next quarter that the service is to be included in a composite application exposed to your overseas subsidiary. The goal of SOA should be to leverage existing security infrastructure to enable enterprises to apply existing processes and tools to authorize users of SOA services, with no need to replicate user stores or to deploy additional infrastructure.

Cryptographic Message Security

Message security, as defined in WS-Security and related specifications, plays a vital role in SOA. Message security has specifically addressed the needs of complex SOA service environments. A single SOA transaction can involve many interacting services. Messages sent across multiple intermediaries must enforce a limited model of trust.

Such transactions require persistent integrity and confidentiality to secure data as it moves and transforms across multiple services. Although the Secure Sockets Layer protocol provides integrity and confidentiality, it does so for only the duration of the session and for only a single network hop. Even if SOA is relatively simple today, security mechanisms must be available to enable SOA to evolve and scale to support multiple, potentially partially trusted, intermediaries.

The management of credentials (that is, public/private key pairs and X.509 certificates) is one challenge of message security. Adding to the complexity of message security, different platforms manage these credentials in different ways. It might be necessary to apply a message security policy consistently across multiple platforms. An effective way to do this is to define key stores as infrastructure components that are available to the system as a whole.

Cryptographic operations encompass the use of XML Signature, XML Encryption, and WS-Security in the context of Web services transactions with SOA. Message security works in two modes: receiving and forwarding operations.

Receiving operations are responsible for

- Verifying any digital signatures on requests
- Decrypting confidential content in requests
- Digitally signing response messages
- Encrypting response content

Forwarding operations are responsible for

- Digitally signing outgoing request messages
- Encrypting sensitive content in the outgoing request message
- Verifying any digital signatures on the response message
- Decrypting response content

Credential Mapping

To derive the full value from your SOA, it is critical to support legacy applications, which includes supporting legacy authentication mechanisms. From a security perspective, that means flexibility in exchanging one credential type for another.

Consider an application that generates SAML assertions, but which must access a legacy application hard coded to require a username and password. For these applications to interact securely, an

intermediary must validate the SAML credential, and then generate the appropriate credential (in this case, a username and password) to access the legacy application on behalf of the service consumer.

Credential mapping capabilities take three forms:

- **Pass-through.** This mode enables credentials taken from the transport (for example, HTTP) or message body (SOAP) to be passed on in a format suitable for the target service. For example, a username token can be mapped from an incoming SOAP request and placed into the HTTP header for a request to the target service. In this way, pass-through credential mapping provides interoperability between WS-* enabled Web services and legacy applications.
- **Impersonation.** This mode enables an identity to use a set of credentials supported by the target service. The credential mapper effectively logs in on behalf of the requestor. Usually, this will take the form of using a single username and password to log in to a legacy server on behalf of multiple user accounts.
- **Versioning.** Given that the version of a supported specification might not have synchronized applications, a credential mapping policy can map one version to another. For example, one application might support the SAML 1.1 token profile, whereas another might only support SAML 2.0. Credential mapping can be used to securely reformat a credential, and either up-version or down-version it as necessary.

Censorship, Redaction, Content Filtering

SOA services are often responsible for transmitting sensitive or regulated data across the network. As a SOA evolves, more and more consumers might come to rely on that data. However, from a governance standpoint, the control of access to that data must reflect corporate policy as it relates to the various regulations around data sharing. Censorship, or content filtering, policy ensures that, unless a consumer has the appropriate entitlements, sensitive or regulated data never leaves the container where the service is running.

Unlike encryption policies, censorship or content filtering policies provide a lightweight mechanism for stripping out data entirely, or for replacing sensitive data with a "cover story" to throw would-be attackers off the trail. This is not a replacement for encryption, which is necessary when the receiver requires the sensitive data or message content. Censorship is important for data covered by regulatory rules, especially those related to privacy.

Interoperable, Standards-Based Security

The basic goal of standards is to shield customers from the complexity and expenses associated with proprietary, noninteroperable software. At the same time, standards efforts should not simply reintroduce those expenses by forcing software customers to implement immature specifications or to follow the moving target of evolving standards.

Beginning in 2000, industry leaders recognized the need to standardize security for XML Web services. That recognition led to the creation of the WS-Security specification, which provides for interoperable,

SOAP-formatted security tokens, digital signatures, and encryption. WS-Security has been widely implemented, and extensive interoperability testing between vendors has been successful.

The WS-Security effort has expanded to encompass a complete vision for application-level security under the umbrella of the OASIS Web Services Secure Exchange (WS-SX) Technical Committee, whose work encompasses Web Services Secure Conversation, Web Services Security Policy, and Web Services Trust specifications. The stated purpose of the OASIS WS-SX Technical Committee is to "define extensions to OASIS WS-Security to enable trusted SOAP message exchanges involving multiple message exchanges and to define security policies that govern the formats and tokens of such messages."

Some of these specifications are finalized standards, whereas others are forward-looking specifications:

- **Extensible Markup Language Signature.** Provides for persistent integrity and authentication using digital signatures in conjunction with XML.
- **Extensible Markup Language Encryption.** Provides for persistent confidentiality using encryption in conjunction with XML.
- **Web Services Security.** Provides SOAP support for SOAP-based security, and defines support for username, X.509 certificate, and Kerberos tokens.
- **Security Assertion Markup Language.** Interoperable format for sharing authentication and authorization tokens.
- **Web Services Trust.** Standard for exchanging security tokens of one type for another via a Security Token Service.
- **Web Services Security Policy.** A component of the Web Services Policy Framework that allows a service provider to describe its security requirements to a service consumer.
- **Web Services Secure Conversation.** A standard for application-level security contexts encompassing long-running, multiparty transactions.
- **Web Services Federation.** A standard for interenterprise identity sharing.

Conclusion

Many organizations are seeking the business benefits offered by transforming enterprise applications into SOA services. To do this, they require flexible yet centralized management of policies, coupled with consistent enforcement across the SOA ecosystem. In many cases, organizations have already invested in security infrastructure, and they now seek a way to leverage that infrastructure within their SOA ecosystems. However, traditional methods of securing applications have come up short, lacking the comprehensiveness, consistency, and flexibility required by SOA.

Oracle provides a uniquely effective solution to the SOA security problem. Its intelligent security policy provisioning ensures the consistent application of existing security infrastructure, processes, and tools across the heterogeneous environments typical of SOA. By providing governance over the loose

coupling between security policy and application logic, Oracle's SOA security capabilities allow organizations to think globally about security, while letting the runtime system act locally to enforce security policy according to whatever local conditions prevail. This policy-driven approach to runtime governance provides the key to unlocking the benefits of a secure SOA.

Think Globally, Act Locally: Policy-Driven Security for a Trustworthy Service-Oriented Architecture
March 2010

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

SOFTWARE. HARDWARE. COMPLETE.