Service-Oriented Architecture

Byline

Security Patterns within a Service-Oriented Architecture

Heather Hinton, Ph. D
Senior Security Architect

Maryann Hondo
Security Architect for Emerging Technology

Dr. Beth Hutchison
Senior Technical Staff Member and Web Services Architect

November 2005
http://www.ibm.com/websphere/developer/services/

Abstract

A service-oriented architecture (SOA) [SOA],], [SOA&WS] is an approach to designing business applications to be more responsive to a changing business environment. In the context of an SOA, a business service is a repeatable business task; business applications are in turn created by combining a series of business services. In composing business applications, there can be a business justification for separating the business application logic (and an application developer) from the infrastructure. This approach of isolating business functionality at the application level into services can also influence how we think about infrastructure and management applications.

Business application design implementing this type of separation of concerns should encourage the separation of infrastructure and application. In the context of security infrastructure, this allows a business application to be security unaware. The premise is that any security data or security process is provided outside the application itself and is known to and managed by the infrastructure or middleware. An example of this is to think of security mechanisms (SSL, VPN's) as an instance of an implementation of a security channel protection service albeit one that is at the protocol layer.

An SOA provides an opportunity and motivation for taking another look at the security mechanisms currently deployed in a business in support of a set of business services. It can be an opportunity to look at encapsulating or decomposing complex security mechanisms into a set of related security services. To do this, we need to start identifying the points at which security decisions are made and the security control points where these decisions are enforced. If we identify these points, we can then separate out when a security decision can be provided by a security service or when it can be provided by the application code itself. A security decision service can be provided to perform the (repeatable) task of applying the relevant security policy to a service request, so that this functionality does need to be provided by the application code. Likewise, security enforcement, the action taken based on the security decision, can be part of an application design or part of a set of managed infrastructure services.

This paper aims to challenge the reader to think about security-as-a-service within an SOA. In this paper, we focus on an example of security as an infrastructure service in the context of an Enterprise Service Bus (ESB). We discuss the SOA architectural model and how the SOA principles can influence the definition of security as part of an overall service model, the benefits of a SOA based approach to security infrastructure components in a business environment and some typical patterns of the deployment of a SOA-security infrastructure.

Introduction

Introduction to SOA

An SOA is an approach to designing business applications to be more responsive to a changing business environment. The emphasis is on decomposing complex business functions into those elements that are repeatable and which offer the most benefit from standardization across business processes. These repeatable elements act as *inflexion points* that can be replaced by service offerings.

This conceptual modeling of business processes also gives us the opportunity to develop an abstract model of a business' end-to-end application design including its conceptual security model. A services approach encourages us to rethink security requirements for the protection of business applications and data in the context of the business process. In today's environment in which acquisitions and mergers are just part of doing business, it is important to have a service-oriented view of security in any business. This will provide the flexibility to adapt to the challenges of providing security in a heterogeneous environment where speed of integration is critical to a business's success. In such an environment, security processes must be deployed in such as way as to not impede integration.

Security in the context of SOA

Figure 1, below, shows the typical lifecycle of security within any development methodology. Architecting and implementing a SOA is an evolutionary process and therefore the security lifecycle illustrated below has great relevance. In this paper, we will look at some of the security patterns that are typically encountered and show how they map to iterations of the SOA evolution.

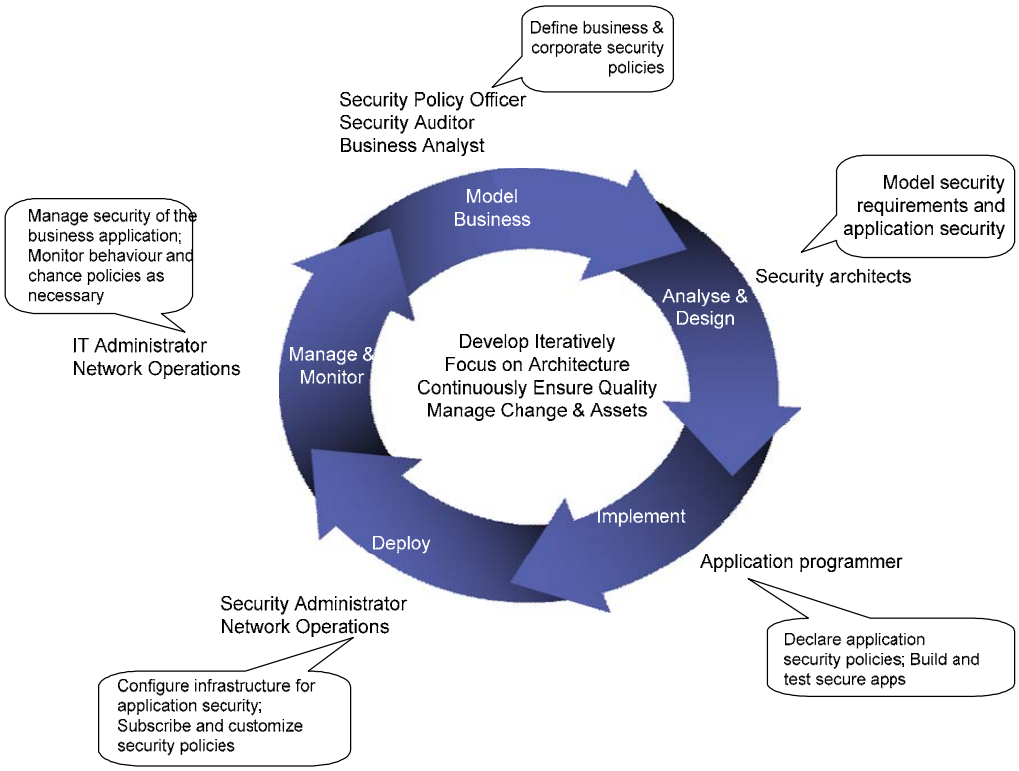

Figure 1: Security in the Context of SOA

The first step in the analysis of security, regardless of the environment, is the definition of a set of requirements. There has been significant work done on a security glossary in the Internet Engineering Task Force (IETF) [IETF] and we draw on that work. In particular, we will focus on user authentication, authorization, data integrity, data confidentiality and accountability. The ability to implement security in an SOA will also come from having a set of cost versus risk parameters for the business. The more a business can articulate the risks to its business, the better it will be able to assess the benefit of countermeasures to protect itself. Businesses must be able to answer questions such as:

- Who needs to have access to what information?
- How is access to data provided? Is it direct or brokered?
- Is there a need for data to be available to external partners as well as internal consumers?
- Are there different requirements on data in transit? In process? At rest?

Many attempts at introducing new business applications hit roadblocks at deployment because many software developers think of security as something that happens in the "next phase" of the project. So another question businesses need to ask is *What pieces of the security logic can have the best return on investment if they become common services (are they targets for reuse within the business)?*

By creating a conceptual model for the application and defining a conceptual security model for the infrastructure we can achieve some level of platform independence (a key concept in any SOA) in the security model as well as take advantage of the development of common services

(sometimes referred to as code reuse). Businesses can then consider the tradeoffs of a security decision point implemented in the middleware versus a security decision point replicated across each individual application. Generally the implementations of security continue to be based on a perimeter defense strategy; because of this, these existing implementations tend to be very inflexible. And while a perimeter defense offers points of control to the business, it really is (or should be) only part of an overall security model.

Just as SOA is about business transformation, SOA security is about security transformation. The definition of specific security services needs to be a well integrated component of the overall business service decomposition. This will require application developers and business analysts to understand and collaborate on the definition of security requirements in the context of the business applications. Each business application will need to define the scope of "security service" in its own application model. Developers and business analysts together need to decide when common security services will provide them the agility they require.

Adopting common services usually requires some effort. So what security mechanisms are candidates for a service?

- Authentication services?
- Authorization services?
- Trust services?
- Message protection services?
- Key management services?

Many vendors are starting to define and standardize common security services (J2EE, container-based security, OASIS WS-Trust, et cetera). An overall security model doesn't need to impose whether an application is J2EE, C# or CICS but there is a point at which the abstract modeling of security concepts needs to be made more concrete through the mapping to concrete service offerings.

Another facet of SOA is enabling the expression of service requirements and capabilities. When these capabilities can be captured as policy expressions they facilitate interoperability and the dynamic discovery of common (security) services. When we look at what security requirements have been articulated for the business (e.g. ensure user authentication as part of request fulfillment) it is important to think about how these requirements can be expressed as generic policies and how these generic policies act (or not) as input to various enforcement points. This allows us to create a layer of abstraction that can be mapped to implementations choices (instantiations of generic policies that conform to a particular template.

In the rest of this paper we explore how to apply SOA security techniques in the context of a messaging broker business application.

Security as a Service

The "Software as a Service" (SAAS) model is gaining traction within the business world. This model allows for SAAS providers to provide access to software services without requiring the customer to host this service within their local environment. By analogy, SAAS occurs when an application (which may in turn be a service) does not internalize, or locally host, security

functionality. This, logically, is the model that has been adopted by the International Standards Organization (ISO) model of security as implemented through a "Policy Decision Point" (PDP) and a "Policy Enforcement Point" (PEP) [ISO10181]. The ISP model is API based, not service based, but the PDP is the entity responsible for the access control decision required to allow/deny access to a resource. The PEP is the entity responsible for enforcing this decision, appropriately allowing or denying the access in response to the access control decision.

In a security as a service model, the access control decision and (ideally) enforcement functionality is not embedded within an application. Instead, the application often provides enforcement functionality for an external access control decision but relies on a service to acquire the decision. This split of enforcement and decision point has advantages, beyond moving the security complexity out of the realm of the application developer. This separation of concerns approach allows for multiple enforcement points to re-use the same decision point functionality. This in turn promotes component re-use as well as the consistent application of access control decisions across an environment. Moving towards an SOA, this enforcement will in turn move to the service invocation layer, as described in this paper.

Extending the separate decision/enforcement approach for an SOA, we can consider a Security Decision Service (SDS) as a service based PDP, sharing the common characteristics of an ESB brokered service, namely implementation independent invocation. An SDS provides a single service that is easily replicated, scaled, and distributed across an environment to provide PDP functionality for all service enforcement points, regardless of the service's actual instantiation. If, in addition, the ESB itself offers security enforcement services, we move to a security as an (infrastructure) service model.

Intro to ESB

An Enterprise Service Bus (ESB) is an infrastructure component for integrating applications and services. ESBs facilitate the connectivity of business logic, where this business logic is represented as a service. Most businesses have a heterogeneous environment with applications implemented in various application programming models, such as J2EE, .Net, CICS, and so on. One goal of universal connectivity, and an ESB, is to allow these different applications (services) to be connected. An ESB facilitates this connectivity by providing transformations that allow the invocation of a service or a service request, which is presented in one format to be "transformed" to a different format which the service provider can respond to. As shown in Figure 2, below, an ESB may provide support for:

1. Routing messages between services
2. Converting transport protocols between requestor and services
3. Transformation of message formats between requestor and services
4. Handling of business events from disparate sources

At a conceptual level, an ESB allows the application designer/developer to build an application framework that invokes services without having to know where these services are located or how they are invoked. An ESB is responsible for the routing of messages between service requestors and service providers. By handling the conversion of transport protocols, the ESB again allows

an application developer to avoid supporting multiple transport protocol based invocation formats. The application developer is able to concentrate on defining a generic application message format, and leaving the protocol transformations to be managed by the ESB. When open standards are exploited, the same can be true for the message format of external resources – the developer does not need to focus on the differences based on protocols but can rely on the ESB to transform and route the request to a provider that can respond to the request.

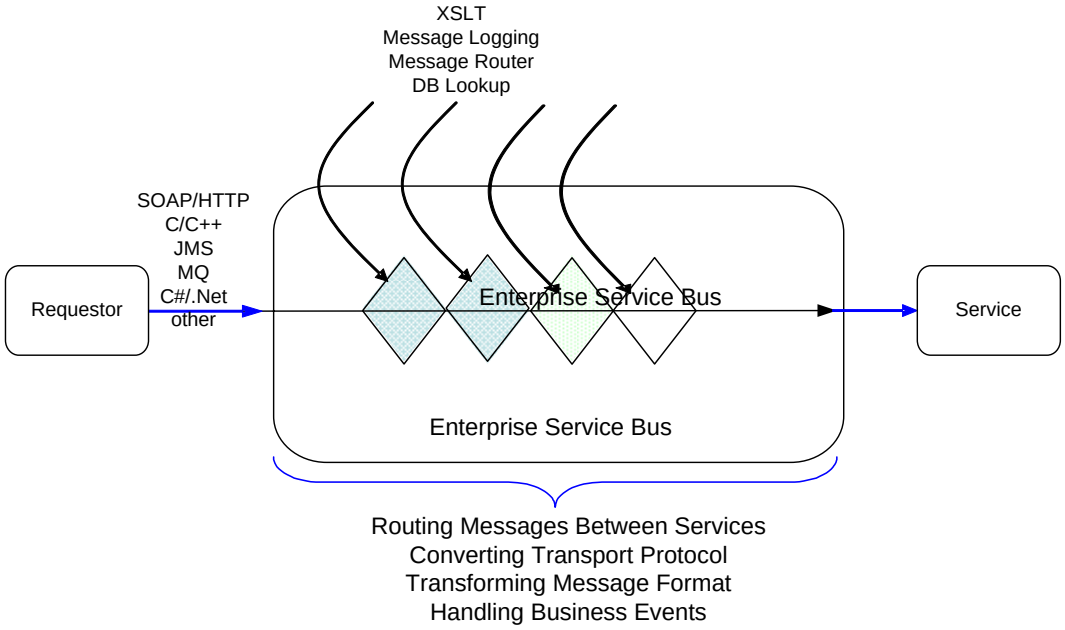

Figure 2: Generic Enterprise Service Bus

An ESB provides application developers a decoupling of the format of a service request and the service implementation. For example, while a particular instance of a service, implemented as a J2EE application may have one message format, a .Net based implementation of a similar service may have a different message format. By supporting both message formats and providing transformations between message formats, an ESB can allow a single message to be used to invoke the independent service based on information other than location and protocol. As more complex transformations are available, other factors such as the identity of the requestor, attributes of the requestor, or business logic surrounding service invocation (including factors such as quality of service) can be used as part of the ESB message invocation framework.

Deploying resources as Web services can exploit these features of universal connectivity through a standardized interface. However, redeployment of existing resources is not always possible. When an application is exposed as service, an ESB can provide the message format transformation required to support the “service-ification” of existing resources. That is, the ESB can handle message transformations of the form where a SOAP/HTTP request is transformed into the required resource invocation, such as a CICS or a .Net resource invocation. Note that while a service interface is typically thought of as being WSDL defined, there are many instances of XML defined services interfaces (e.g. RosettaNet, or custom XML interfaces from “legacy” XML-defined-interface services) that can also be used and handled by an ESB.

The decoupling of interfaces and connections can also impose some additional responsibilities on the application designers. A part of identifying which services can benefit from an ESB is identifying similar services that exist in multiple implementations as a result of different levels of service or class of requestor. An ESB must therefore ensure that this same functionality and differentiation of service access is preserved within an SOA. An ESB does this through the use of mediations. A mediation is logic which operates on a message in flight between a requester and a provider. Mediations have access to the entire message and are able to act on information contained in the message, both at the header and body level.

For example, in Figure 1, above, message logging is identified as functionality provided by the ESB. Within the ESB, message logging can be applied to all messages, or a subset of messages based on the service request, the service and optionally policies attached to the requestor/service combination. As another example, the ESB may provide a mediation to provide authentication and identity mapping on an identity claimed within a message header.

Existing Security Principles applied to the ESB

Moving to an ESB also implies an understanding of how existing security principles apply in this new framework. The principles that this paper focuses on include enforcing authentication, access control and message protection. Our approach to securing ESB applications is built on a layering approach, where security is applied in layers. These layers are based whether data is in transit, in process, or at rest. A typical architecture, and mapping of a data's characteristics, is shown in Figure 3, below. Particular points of concern for security surface when data moves from one state to another, for example data in transit moves to be data in process. At this point, the techniques used to provide security for data in transit may be "broken,"or terminated.

When data is in transit, the appropriate confidentiality and integrity should be applied using encryption and digital signatures. Different levels of in transit security are often applied at different points of data's progress. Data in transit over public networks may require security as applied through techniques such as SSL/HTTP to protect information over an HTTP channel. These same techniques may be used to provide authentication of the request origin, based on mutually authenticated SSL. Within an Enterprise network, techniques such as SSL may be used to provide confidentiality but the advanced confidentiality and authenticity provided through mutually authenticated SSL may not be required. In some cases, the request message itself is also protected, independent of the transport layer protection. This allows the message to continue to have security protection (e.g. confidentiality through message body encryption) even when the transport layer protection is no longer applied.

As part of moving from in transit to in process, additional techniques may be applied to ensure the authenticity of the request and the identity of the requestor. Transport layer protection is typically "broken" when moving across trust boundaries, such as off of the network and into an in process state. While in process, requests may be authenticated (based on the transport layer or based on information included with the request). Authenticated information can also be used as part of the access control decision, which in turn determines if the in process processing is allowed (and if not, processing should be terminated).

While at rest, data typically is not protected, although in many cases data should be protected against alteration and disclosure. Techniques for protecting data at rest are discussed in other papers, including [Cachin2004],[SAN].

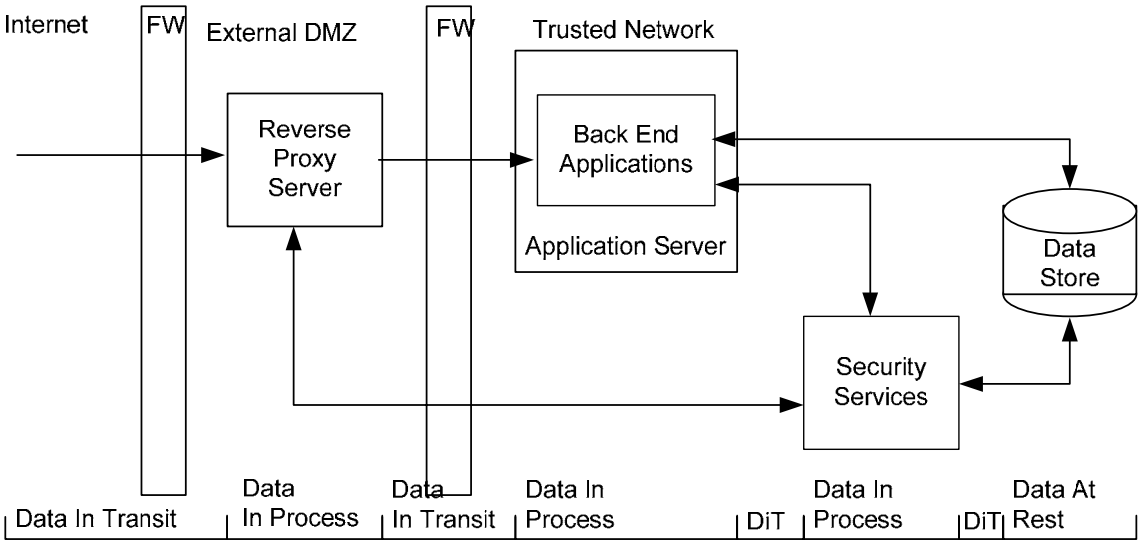

Figure 3: Data In Transit, In Process, and In Rest within an Architecture

Based upon the deployment topology, business requirements may indicate that access control decisions should be made every time a request/data moves from in transit to in process. This is based on the premise that each time a request moves further into the processing stream, there should be more information available to allow for successively more detailed access control decisions. Implementing coarse to fine-grained access control decisions at every transition enables an organization to terminate processing of unauthorized requests at soon as possible. An incremental approach can minimize the unauthorized requests that the target application (the end result of the request) receives. Thus this approach is used not only to improve security, but to minimize the performance impact of security processing at the application layer.

Security within an SOA needs to be seen as an incremental set of services that can be applied at different layers or at different control points. These services can be applied at the transport layer, at the message layer, at the service/resource invocation layer, and at the resource fulfillment layer, or the layers at which requests move from in transit to in process based on the requirements. When moving to an ESB model, we must ensure that these security layers continue to work together and that the overall security model is preserved appropriately for the application.

Routing: ESB Security at the Transport Layer

The ESB has the responsibility to preserve the transport layer security of an incoming message. It follows that in cases where the threats to the ESB are from unauthorized access only, it may

mean that the ESB does not do any additional security processing on an incoming request. In other cases the ESB may offer a mediation which validates that the transport protocol underneath the request is part of a trust relationship. Included in this level of mediation are "typical" transport layer security techniques such as virtual private networks (VPNs) and mutually authenticated SSL used to validate the trust relationship between invoker and service. The ESB, as a trusted component within an SOA, must be able to provide transport layer security, and establish a trust relationship between the origin of the request and the ESB entry point for services.

Allowing the ESB node to mediate transport level security and trust relationships establishes the ESB as a component in the security model. The ESB then supplies a service, that brokers or mediates security for all the services on the bus, removing the need for each service to independently manage and evaluate trust relationships with every possible service invoker.

ESB Security at the Message Layer

As part of the routing of a message, the ESB can offer an additional mediation that will ensure that messages are routed to trusted (or un-trusted) endpoints as required. This level of security may be very coarse-grained, meaning that all requests associated with a particular trust relationship (transport level security evaluation) are routed to service instance A. This ESB-brokered message level security may be very fine-grained, meaning that a request is routed to service instance A if the requestor, acting within a trust relationship, has attributes X and Y and to service instance B if the requestor has attribute Z, or to service instance C if the request is for an operation Q.

Fine-grained security requires that the ESB be able to provide identification and authentication services beyond the transport layer. Recall that techniques such as mutually authenticated SSL can be used to authenticate a requestor at the transport level. This is typically associated with the component that has negotiated the SSL, where this component is typically an edge component at a partner's site. This transport layer security may be applied to many requests, including requests by many different requestors. Fine-grained security requires that the ESB be able to differentiate the requestors within a trust relationship. This level of differentiation is typically provided through the inclusion of security tokens that refer to the explicit requestor and that bind that requestor to the request itself.

Management and mediation of security tokens (message level security) may require that ESB mediations provide additional trust services functionality. Typically a trust service is able to evaluate the identity and authentication, as represented by a security token applied to the incoming request and required by the service implementation. As part of the ESB security, mediations will ensure that the appropriate message protection services are applied. This includes confidentiality (encryption) and integrity (digital signature) checks, as well as identification and authentication (through token evaluation) of the requestor, all applied to an incoming service invocation. This information will be used to establish the context in which message based routing will occur. Note that additional mediations may also be applied to ensure that the routed and transformed message in turn has the required confidentiality and integrity protection, and identification/authentication.

ESB Security as a Service

A Security Enforcement Service, or SES, then remains a potential component of an ESB enabled SOA architecture. The SES allows access control decisions to be applied to a request to access a service, whether that service is implemented as a CICS resource, a .Net resource, a J2EE resource or a forgotten legacy format. Within the context of SOA and ESB, a SES is independent of the transport method of a particular message/service request, as shown in Figure 4, below.

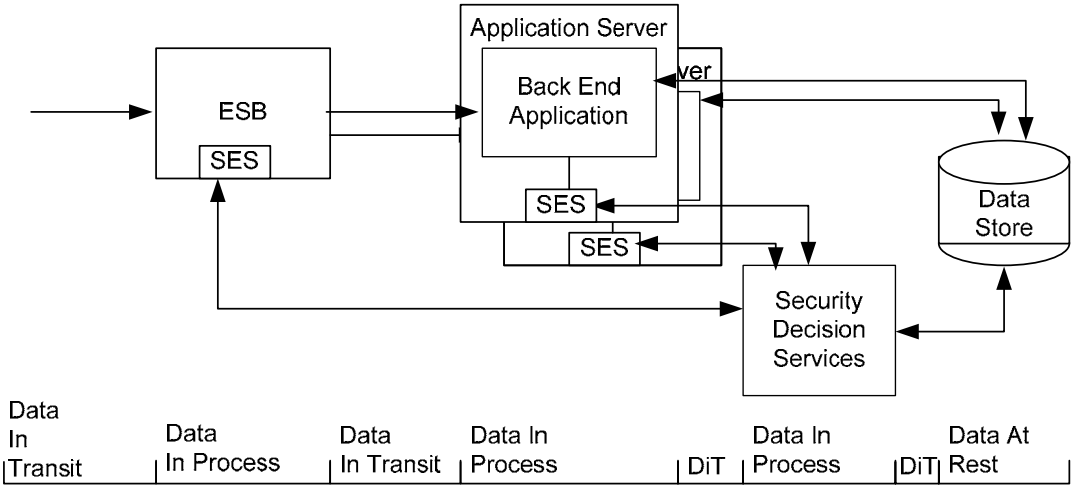

Figure 4: Security as a Service within an ESB

Figure 4 shows how we can logically view security as a service across an ESB. In this figure, the ESB offers a security enforcement service in addition to the back-end applications security. In this picture, different security issues must be addressed at different points in the request cycle.

Within the ESB, the Security Enforcement Service itself can be shown to have the same type of mediation model as the ESB. This is shown in Figure 5, below:

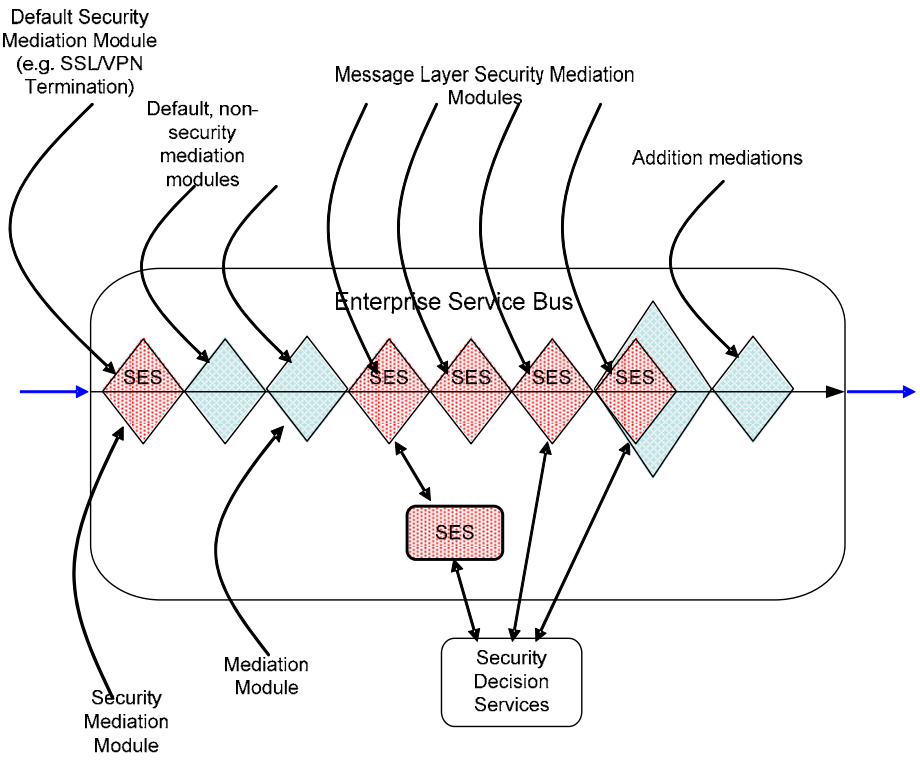

Figure 5: Blow-Up of Mediations in an ESB

In this logical model, security enforcement services appear to be repeatedly applied. This is because security services are composed through the ESB mediations, thus, a single mediation module may be used to provide message level encryption/decryption. A separate mediation module may be used to provide digital signature validation on a signed message. Yet another mediation module may provide requestor authentication and identifier mapping. This model of integrating security services as part of the ESB will provide many benefits such as end-to-end security for service implementations. It will however require the ESB to support additional services like key management if intermediate service points are required to view or route messages based on a request that has had confidentiality or integrity techniques applied to it.

Usage Patterns

One feature of SOA is enabling the development of common code and providing code re-use. A SOA can also help identify common policy definitions and enable common policy development and reuse. Moving to SOA, however, will be an evolutionary process. In this section, we discuss the security implications of some of the basic deployment patterns that can be employed as Enterprises move to a full, ESB-based SOA environment. As we move through these deployment patterns, we are able to move the security layers (introduced above) into the ESB functionality.

Application Managed Security

A common application deployment pattern is one in which applications implement and manage their own security. In this case, security is application specific. Each application has its own

security policy for authentication and authorization, from coarse-grained to fine-grained security. This type of application managed approach to security is often built on proprietary call-outs to a proprietary security provider (access control decision point). The application managed pattern, an example of which is shown in Figure 6, below, results in the replication and duplication of information across applications and data stores.

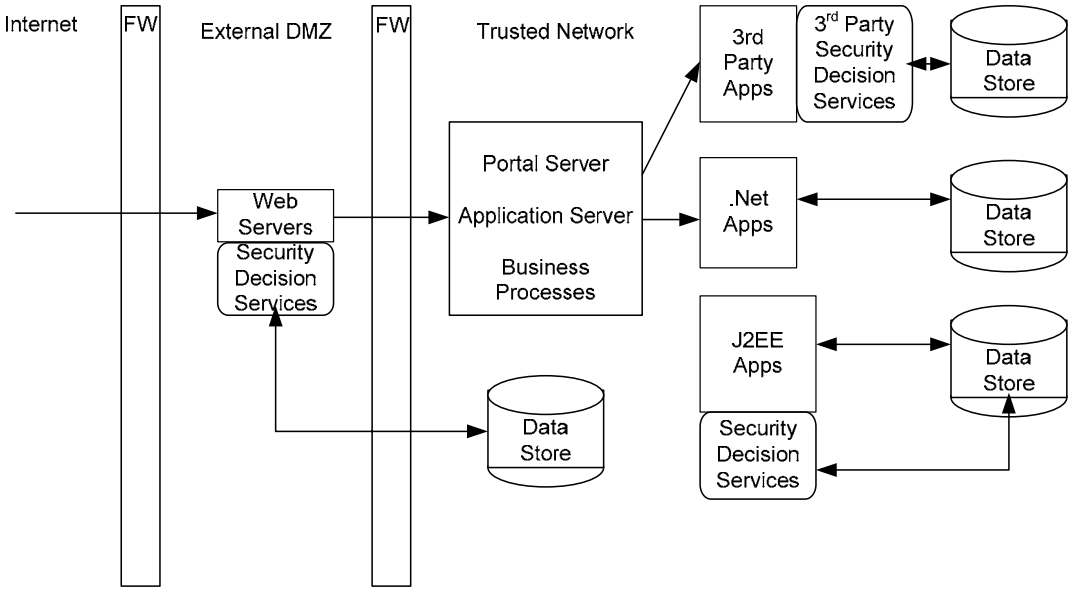

Figure 6: Application Managed Security Functionality

A way to move towards an SOA, is for the application managed approach to re-focus on security as an API/SPI interface with SPI providers adhering to standardized interfaces (e.g., JAAS, JACC, web services based authorization). Within an SOA, the applications can then move to an infrastructure managed approach to security as a first step. Migrating security is not something that is achieved overnight (if ever). It is important to move towards this approach in a logical manner. The first step is to move as much of the application managed security functionality towards implementation of standardized interfaces for security providers. This starts the process of common security functionality, as shown in Figure 6, below.

Brokering Security through a Reverse Proxy

Another step along the path to an SOA is to introduce a specialized component, in this case a reverse proxy server to the architecture, as shown in Figure 7. This component is typically already a part of many Enterprise architectures, where it is used to provide perimeter-based security services for the Enterprise. A reverse proxy moves authentication functionality to the *edge* (or perimeter) of the network, so that only authenticated users are allowed into the Trusted Network. This front-end component may also provide coarse-grained access control decisions, eliminating unauthorized requests to the back-end resources. These access control decisions may be coarse-grained (the user is authenticated and therefore access can be granted) or fine-grained (the user is not a member of the group/does not have the role required to access the requested resource) if the information required to make the fine-grained decision is available to the edge-based decision point.

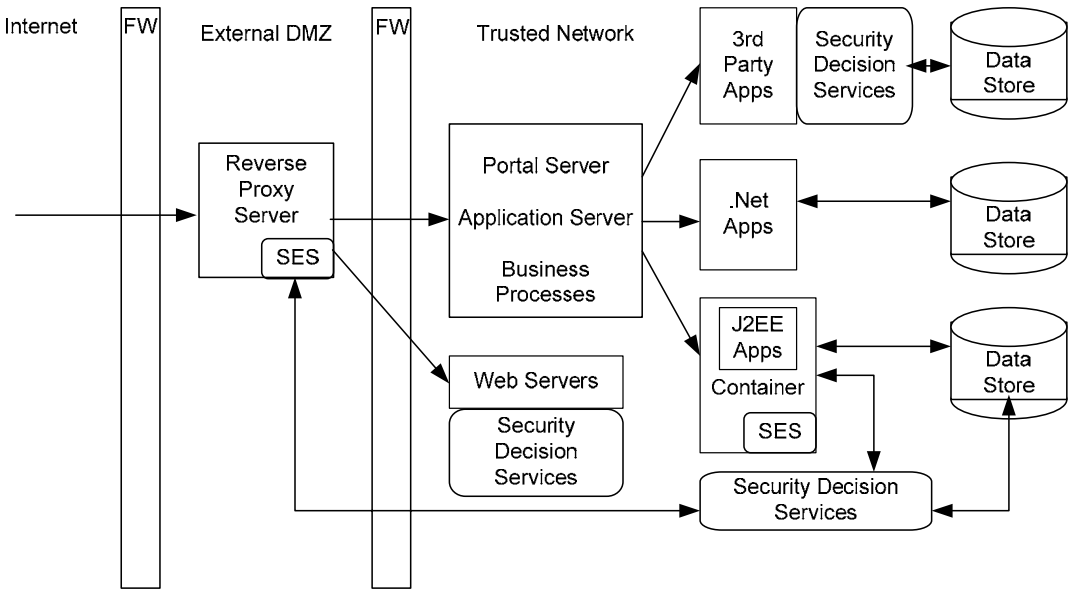

Figure 7: Moving Towards Infrastructure Managed Security: Reverse Proxy Pattern

This additional step consolidates security functionality to a single logical point (the reverse proxy server) thus identifying a common security point. This common security point in turn provides an opportunity to define a common security service, a core component of SOA. A common security service can also help with systems management issues eliminating the need for detailed knowledge of authorization decisions in the back-end application as part of the fulfillment of that application.

This approach also allows for some level of platform independence. Since the authentication is now done at the edge, it doesn't matter if the service is implemented as a type X resource or a type Y resource since the reverse proxy has determined user access to the resource based on the message request and the authentication context. Similarly, a reverse proxy can provide some addressing transparency (another characteristic of SOA) through "routing" by mapping between the Enterprise's internal architecture and a publicly requested URL.

Gateway Pattern

Another step towards an SOA is the adoption of a gateway pattern, as shown in Figure 8, below. Introducing a gateway also continues the evolution of an environment towards an ESB-based SOA. The gateway can provide an intermediary step towards a full ESB, allowing the Enterprise to move to a deployment model in which the application focuses on business logic and the infrastructure provides the base, common messaging functionality needed in SOA.

The gateway pattern is similar to the reverse proxy pattern shown above in Figure 6, with additional features. These features include the ability to handle different transport layers, unlike the HTTP-only nature of the reverse proxy, and enhanced message transformation, based on message-level introspection. The gateway can be configured to also perform security decisions & enforcement, and to support the necessary protocol trans-coding to establish connection with the partners and to ensure the overall quality of service required (e.g., reliable for order handling, best effort for order status updates) in the message exchange.

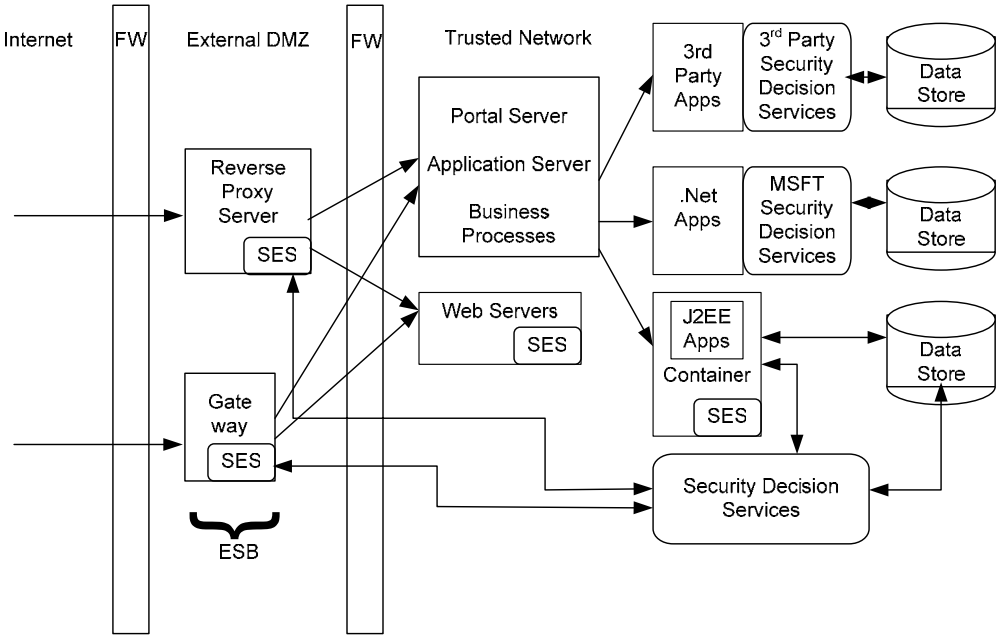

Figure 8: Gateway Based Migration to Infrastructure Managed Security

Thus the gateway provides an incremental step towards full ESB functionality. What are the advantages of a gateway-based approach to a full ESB? As part of its enablement of an ESB model, the gateway approach:

- Supports a single point of control for security enforcement
- Enables a layered approach to security
- Provides a single point of control for identity mapping
- Advanced gateway implementations support transactional integrity

Gateways are typically deployed at the edge of a network, where a request is routed from the Internet/public network to an Enterprise's trusted network. At this edge, typical security functionality includes

- Evaluation/establishment of transport layer security
- Authentication of the requestor and determination of requestor's local identity
- Validation of the request format
- Coarse-grained authorization of the request message and origin
- Crossing trust boundaries from a public form (e.g. public URL of web resource) to a private form (e.g. internal URI address of the resource)

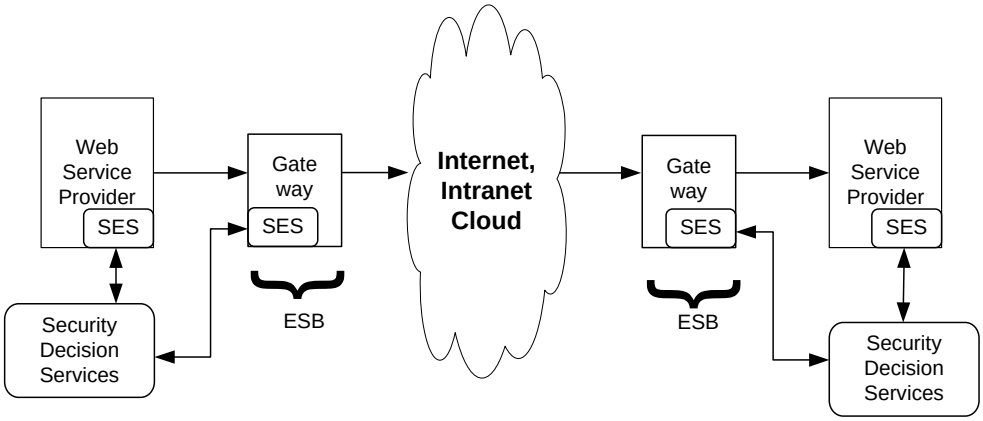

Figure 9: Gateway Based ESB Pattern

Gateway with Security Enhancements/Full ESB

As gateways add more functionality and become more advanced in their processing, they move toward supplying the functionality of a full ESB, so that the ESB logically replaces the gateway, as shown in Figure 10, below. As this occurs, the gateway begins to act as a common switch for all messages crossing into the enterprise. At this level, the gateway/ESB is able to:

- Provide support for messages from multiple inbound transport protocols
- Transform requests from one transport layer protocol to another
- Potentially normalize the request to a canonical format for the appropriate resource
- Authenticate requestors and appropriately map authenticated identity to a locally valid identifier
- Route requests based on metadata in the request contents, including identifier, message contents, and system load
- Transform requests from one message format to another, normalizing the request to the appropriate format based on the routed-to back-end application

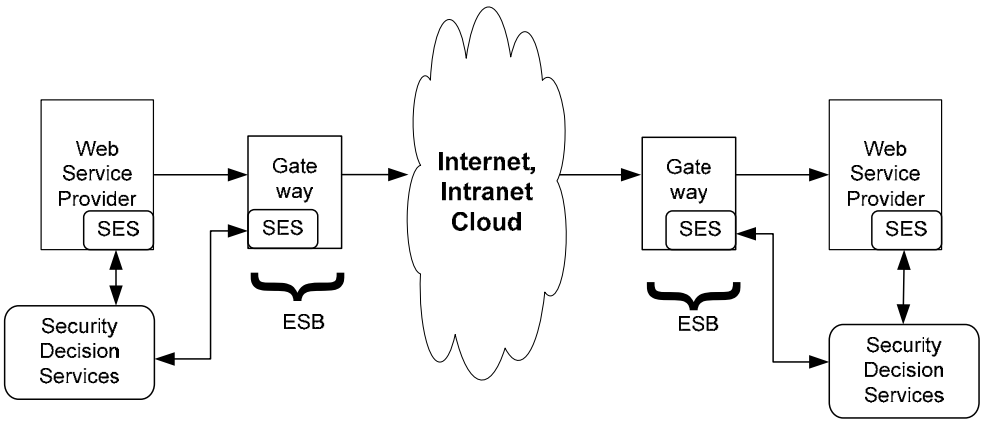

Figure 10: Gateway Based ESB Pattern

Not all advanced functionality is required for all requests, and the number and type of features applied to any one message exchange may differ based on the requested resource, requestor and message contents. A full ESB approach achieves this through defining mediations. Mediations include specific functionality that can be re-used across service requestor and implementation pairs to encourage code re-use within an ESB. The flow shown in Figure 11, below, includes a blow-up of the ESB functionality, where the diamonds represent mediation modules and the implementation of the ESB mediation function. Each mediation module represents a specific, repeatable function and is invoked as required.

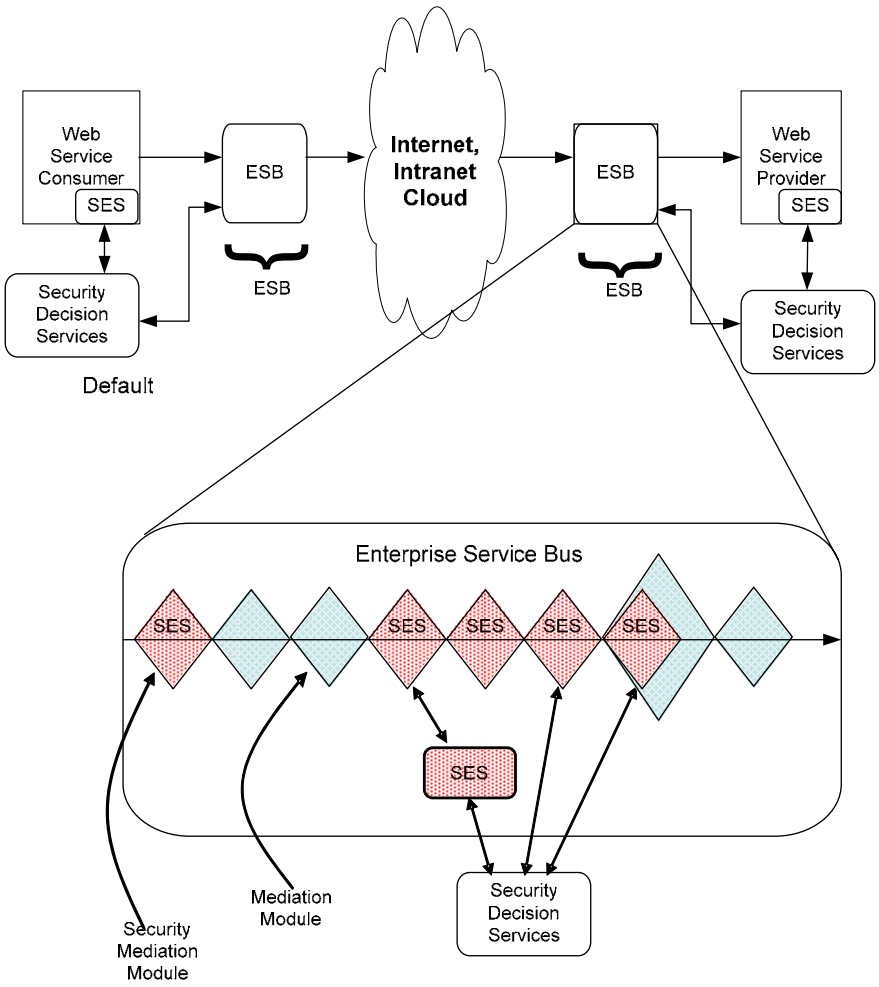

Figure 11: Gateway with Full Enterprise Service Bus

When security is added as a mediation within the ESB layer, mediation modules can be implemented and re-used. This is in turn made possible by the security-as-a-service approach described in this paper.

Conclusions

Implementing an SOA requires a process of transformation. Each organization that deploys a SOA will go through this process in its own way based on that organization's unique requirements. Many organizations will begin to enable SOA by starting with a broker model. This will allow these organizations to initially focus on deployment instead of architecture, or application re-design.

The SOA process encourages the re-use of existing security functionality as appropriate for local application deployment. Not all applications will be immediately deployable in a SOA model. Legacy functionality may need to be encapsulated within business logic instead of being exposed with a normalized interface. The model of the ESB will help ease this process. Application brokers implementing ESB functionality will provide a valuable piece of a SOA ecosystem; they will enable the service-ification of proprietary systems so that these systems can operate within an SOA.

Implementing security across an SOA requires that we focus on the conceptual security model across infrastructure components. Breaking down security functionality into security tasks allows us to build security services that can outlast application level implementations of security. A key part of this process is the establishment of a timeline to take an architecture from a proprietary approach (fully application managed security) to a full, interoperable ESB based SOA. This timeline will often include the migration away from a proprietary application managed approach to security, through a reverse proxy pattern for HTTP-accessible resources, to a gateway pattern, for HTTP and non-HTTP resources, to a gateway as a full ESB.

The ESB is designed to work within an existing environment. While it will allow for the consolidation of common services and functionality, it will not require the replacement/redeployment of a new infrastructure. For example, buses and message brokering have existed for many years and will continue to be deployed in products such as IBM's MQ series. These messaging services function well and do not need to be replaced just because a new architectural style has been introduced, or because new technology is available. Customers who already deploy MQ should assess whether the existing topology already supplies sufficient security for the business while also assessing how to leverage the ESB to extend the reach of their MQ assets.

An ESB provides the decoupling of service request and service invocation to support an SOA. It also includes the business events normally bound to the service invocations that can be logically associated with the message functionality, such as routing, message transformation and the underlying transport protocol conversion. The ESB is therefore a key enabler of a security-as-a-service model, in turn enabling a service-oriented approach to security infrastructure.

About the Authors

Heather Hinton is a senior security architect with IBM in Austin, Texas. She has 12 years of experience in computer and information security. She has a PhD in electrical and computer engineering from the University of Toronto. Her areas of expertise include Federated Identity Management, access control, composition of policy, wireless, network, and systems security.

Maryann Hondo is the security architect for emerging technology at IBM, concentrating on XML security. She is one of the coauthors of the WS-Security, Policy, Trust and Secure Conversation specifications announced by IBM and other business partners. Before joining the emerging technology group she managed the IBM Tivoli Jonah team (IETF PKIX reference implementation) and was security architect for Lotus e-Suite participating in the development of Java Security (JAAS).

Dr. Beth Hutchison is a senior technical staff member and a web services architect working on IBM's ESB technologies. She has consistently worked on leading-edge technologies, initially as the lead developer for the first release of WebSphere MQ on the distributed platforms. Subsequently, she took on the role of performance architect for IBM's Java Virtual Machines. She has now rejoined the MQ family and is working on systems management across the ESB.

Bibliography

[Cachin2004] C. Cachin, Security in Storage Networks -- A Current Perspective. *Invited talk at ZISC Information Security Colloquium*, Nov. 2004, http://www.zurich.ibm.com/~cca/talks/storage-2004nov-zisc.pdf

[ISO10181] ISO 10181, The Access Control portion of the ISO Security Framework

[IETF] R. Shirey, IETF Internet Security Glossary, Network Working Group, Request for Comments: 2828, http://www.ietf.org/rfc/rfc2828.txt

[SAN] J. Tate et al, Introduction to Storage Area Networks, IBM Redbook, http://www.redbooks.ibm.com/abstracts/SG245470.html

[SOA&WS] New to SOA, http://www-128.ibm.com/developerworks/webservices/newto/index.html

[WS] New to Web Services, http://www-128.ibm.com/developerworks/webservices/newto/index.html