

Service Oriented Architecture: Security Challenges

By Jonathan G. Gossels & Richard E. Mackey, Jr.

jon.gossels@systemexperts.com; dick.mackey@systemexperts.com

Introduction

It is almost impossible to pick up an IT technical publication these days without seeing articles touting the virtues of Service Oriented Architecture (SOA). SOA offers the promise of reduced development cost and faster time to market primarily through code reuse. The critical topic that is lost in all of this is security—and securing a SOA environment is challenging. This brief article identifies several of the security issues you will need to resolve in securing your SOA infrastructure.

Context

The term Service Oriented Architecture is frequently bandied about, but it is not broadly understood. A recent survey noted that approximately 50% of IT professionals professed to have some familiarity with the term. Yet, those same respondents overwhelmingly failed to associate the defining attribute, code reuse, with the term. Those findings simply underline that we are in the very early stages of a major technological change.

For the purposes of this article, SOA means an environment characterized by:

- ▲ **Service Virtualization:** A reusable set of code with well defined interfaces that performs a well recognized business function (e.g., verify identity or calculate loan balance)
- ▲ **Service Reuse:** Applications draw the bulk of their functionality from a catalog of preexisting services
- ▲ **Service Brokering:** Services register their interfaces with a broker so they are easily accessible by other applications

SOA promises not only reduced development time and faster time to market, but also improved integration of old and new components. It also makes it easier to connect any IT resource, regardless of where it is deployed, to any other. Organizations with these goals include technology such as an Enterprise Service Bus (ESB) in their SOA implementations.

There is a common misperception that SOAs require the use of Web Services technology (Web Services Description Language (WSDL)), Simple Object Access Protocol (SOAP) and Universal Description Discovery and Integration (UDDI). They don't. Heterogeneous SOAs may combine Web Services with other middleware such as WebSphere MQ or XML over HTTP.

Challenge #1: Who Controls the Data?

One of the benefits of developing applications in an SOA is that the applications become much smaller. In the degenerate case, the application

is made up simply of a string of calls to blocks of reusable code (services) made via well structured programming interfaces. The services may also interact with one another, requiring one to pass sensitive data to another. The size of applications shrinks in proportion to the number of services available. In order to accomplish the maximum code reuse, services must be designed to be as general as possible.

This pressure to produce general-purpose services can conflict with application-specific security requirements. Imagine the case of a single, monolithic or tightly coupled distributed application handling sensitive data. In a traditional application environment, the data would be protected to an appropriate level across all networks and systems that it traversed during processing. There would be mutual authentication of principals and authorization levels would be enforced. Audit trails and logging would be part of the infrastructure. You care about who handles the data and how it is handled.

In an SOA environment, implementing those same controls is difficult. The problem is that one of the major benefits of a set of broadly used services is that they are typically designed such that they don't care who they talk to. They treat everyone the same—that goes against a basic tenet of security: know whom you're dealing with and what precisely they are allowed to do.

The application hands off data to a particular service and now that service must properly protect it. Unfortunately, in most cases, the services were developed without security in mind (by design based on the underlying SOA philosophy). The ability to protect data becomes increasingly more difficult, the more abstract or general the functions become. Further, the systems that the services run on were also configured to be general purpose, not hardened to the requirements of a sensitive application.

If you are going to run sensitive applications in the SOA, then EVERY component and service as well as the underlying infrastructure that supports the services and components must be designed to meet the security requirements of the most critical application.

Challenge #2: Who Controls the Addition of New Services?

There is a well-established body of best practices for maintaining a secure processing environment. Formal change control procedures are used when new software or systems are added. Only specifically authorized personnel are allowed to implement changes. The opposite is the norm (and vision) for many SOAs.

Organizations are encouraging the development and deployment of new services. SOAs typically rely on a brokering mechanism that enables services to publicize their service contracts and other descriptive information in a catalog or shared repository. It is not clear how an application would be able to recognize a rogue service.

If you are going to run sensitive applications in the SOA, then there has to be a formal process for reviewing the security of new services and a structured change control process for adding new services to the environment.

Challenge #3: How is Authentication Handled?

In many implementations of SOAs using SOAP and MQ Series, by default, no authentication is performed. However, even if a developer enables Web Services Security, he still must determine what authentication means in the loosely coupled SOA environment. Whether a developer chooses to enable point-to-point authentication or not, there are at least two architectural approaches to consider:

Preserve the Basic No-security SOA

Preserve the basic no-security SOA by building a wall around a set of services and supporting infrastructure that all deal with each other in a trusted manner. In complex SOAs, this may take the form of implementing a second (or third) ESB with each component and service tuned to an agreed upon security level.

Mutual Distrust

Mutual distrust or trust and verify. This model applies well-established open networking principles to SOA design. Every principal has to authenticate to the infrastructure and to each other. Either the infrastructure or each component has to maintain authorization state about which principals can use which services or access which resources. So, imagine a world where every application trusts the infrastructure, registers its services, and allows the infrastructure to control who can talk to whom. Naturally, the infrastructure would also support logging and auditing.

Challenge #4: No Concept of End-to-End

In larger SOAs, software infrastructure is used to create a bus-processing model that aids in dynamically connecting, mediating and controlling services and their interactions. The term Enterprise Service Bus was coined by Gartner to define a new type of application integration middleware that is intended to act as a lightweight, ubiquitous integration backbone through which software services and application components flow.¹ Its concept evolved out of Enterprise Application Integration (EAI) tools.

One of the problems organizations face with SOAs is that they provide no end-to-end security. Let's look at an example of an organization using an ESB to connect a set of modern applications to services offered by a legacy financial application. The ESB (implemented with TIBCO Business Works) supports both Web Services and MQ Series protocols and is designed to allow various filters and translators to intercept messages, operate on them, and re-queue them for further downstream processing. This bus model of processing is flexible and powerful and as we noted above, is consistent with an SOA. One of the main benefits of the ESB is that it supports a centralized translation function. This will allow the organization to adapt to the introduction of a replacement application by making changes in a single location rather than modifying each of the applications that create transactions.

Once the ESB translation components have completed their tasks, the application messages are queued in a dedicated transmission queue for retrieval by a bus adaptor that front-ends the legacy financial application service. From there, the legacy application takes over and does its processing. When the financial transaction is complete, the legacy application returns its results to the adaptor, which, in turn, enqueues the results as messages in the ESB.


The beauty (and danger) of this model is that each of the components in the chain is (relatively) unaware of the processing that occurs in the other components. Even if the system requires components to authenticate to one another, there is no concept of end-to-end control over an SOA processing path. The closest examples we've come across have jerry-rigged these controls by requiring components to traverse certain services so that controls are executed. It is not pretty, but it is a start.

Challenge #5: Meeting Requirements with Off-the-Shelf Components

It is not uncommon for organizations deploying SOAs to begin by selecting popular infrastructure products. While this approach maximizes application integration, it eliminates business requirements and information security requirements from the selection process—this is a mistake. Often, these requirements cannot reasonably be retrofitted into the environment.

Last Word

While implementation details varied, the basic rules of developing security architecture have been well established for the past several years. Service Oriented Architectures are a new ballgame and require creative solutions to a wide range of problems. The most important of these solutions are architectural in nature—common infrastructure or replicated infrastructures by security level, how to accomplish mutual authentication, how to manage keys, how components are added to the environment, and who controls data. This article has touched on just a few examples of security challenges with SOAs.

As an industry, it appears that SOA is more than a buzzword and is here to stay; our challenge over the next several years is to develop practical solutions to the inherent security problems of SOAs to enable our organizations to reap the benefits of code reuse, shorter time to market, and any-to-any processing interaction. 

Jonathan Gossels is President and Richard Mackey is Principal of SystemExperts Corporation, a recognized leader in network security and regular contributors to The ISSA Journal. www.systemexperts.com

¹ Roy Schulte, Gartner