Security for Service Oriented Architectures

Walter Williams
# Contents

**Preface**  
IX  

**In Gratitude**  
XI  

**Chapter 1 Introduction**  
1  

**Chapter 2 Four Kinds of Architectures**  
3  

2.1 Architecture  
3  

2.2 Infrastructure  
4  

2.3 Software Architectures  
9  

- 2.3.1 Key Principles  
10  

- 2.3.2 Presentation Layer  
13  

- 2.3.3 Business Layer  
15  

- 2.3.4 Data Layer  
16  

- 2.3.5 Workflow  
19  

- 2.3.6 Communications and Messaging  
20  

- 2.3.7 Service Layer  
21  

2.4 Service-Oriented Architecture  
22  

- 2.4.1 Distributed Computing and Services  
23  

- 2.4.2 Process-Oriented SOA  
25  

- 2.4.3 Web Services or an Externally Focused SOA  
27  

- 2.4.4 Enterprise Service Bus  
30  

2.5 Security Architecture  
30  

- 2.5.1 Construction of a Security Architecture  
33  

- 2.5.2 Risk Management  
34  

- 2.5.3 Organization and Management  
36  

- 2.5.4 Third Parties  
37  

- 2.5.5 Asset Management  
38
## Contents

2.5.6 Information Classification 39  
2.5.7 Identity Management 41  
2.5.8 Security Awareness and Training 44  
2.5.9 Physical Security 44  
2.5.10 Communications and Operations Management 45  
2.5.11 Perimeters and Partitioning 46  
2.5.12 Access Control 48  
2.5.13 Authentication 48  
2.5.14 Authorization 50  
2.5.15 Separation of Duties 51  
2.5.16 Principles of Least Privilege and Least Authority 51  
2.5.17 Systems Acquisition, Development, and Maintenance 52  
2.5.18 Confidentiality Models 52  
  2.5.18.1 Lattice Models 52  
2.5.19 Nonrepudiation 53  
2.5.20 Integrity Models 53  
2.5.21 Service Clark–Wilson Integrity Model 54  
2.5.22 Security Assessments and Audits 58  
2.5.23 Incident Management 58  
2.5.24 Business Continuity 59  
2.5.25 Compliance 60  
2.6 Data Architectures 61  

## Chapter 3 IMPLEMENTING AND SECURING SOA 65  
3.1 Web Services 65  
3.2 Extensible Markup Language 66  
  3.2.1 Signing XML 68  
    3.2.1.1 XML Digital Signature 68  
  3.2.2 XML Encryption 74  
  3.2.3 Key Management 79  
    3.2.3.1 Key Information 79  
    3.2.3.2 Location 79  
    3.2.3.3 Validation 80  
    3.2.3.4 Binding 80  
    3.2.3.5 Key Registration 80  
  3.2.4 XML and Databases 82  
    3.2.4.1 A Database Query Language for XML 82  
    3.2.4.2 XML Databases 83  
  3.2.5 UDDI 83  
  3.2.6 WSDL 84  
3.3 SOAP 87  
  3.3.1 SOAP Roles and Nodes 89  
  3.3.2 SOAP Header Blocks 90  
  3.3.3 SOAP Fault 90  
  3.3.4 SOAP Data Model 91  
  3.3.5 SOAP Encoding 91

BUY Security for Service Oriented Architectures  
http://www.crcpress.com/product/isbn/9781466584020
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.6</td>
<td>Bindings</td>
<td>92</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Documents and RPC</td>
<td>93</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Messaging</td>
<td>95</td>
</tr>
<tr>
<td>3.4</td>
<td>WS-Security</td>
<td>99</td>
</tr>
<tr>
<td>3.4.1</td>
<td>WS-Trust</td>
<td>107</td>
</tr>
<tr>
<td>3.4.2</td>
<td>WS-Policy</td>
<td>116</td>
</tr>
<tr>
<td>3.4.3</td>
<td>WS-SecureConversation</td>
<td>129</td>
</tr>
<tr>
<td>3.4.4</td>
<td>WS-Privacy and the P3P Framework</td>
<td>133</td>
</tr>
<tr>
<td>3.4.1.1</td>
<td>POLICIES</td>
<td>135</td>
</tr>
<tr>
<td>3.4.5</td>
<td>WS-Federation</td>
<td>144</td>
</tr>
<tr>
<td>3.4.5.1</td>
<td>Pseudonyms</td>
<td>153</td>
</tr>
<tr>
<td>3.4.5.2</td>
<td>Authorization</td>
<td>162</td>
</tr>
<tr>
<td>3.4.6</td>
<td>Authorization without WS-Federation</td>
<td>173</td>
</tr>
<tr>
<td>3.4.7</td>
<td>WS-Addressing</td>
<td>178</td>
</tr>
<tr>
<td>3.4.8</td>
<td>WS-ReliableMessaging</td>
<td>183</td>
</tr>
<tr>
<td>3.4.9</td>
<td>WS-Coordination</td>
<td>191</td>
</tr>
<tr>
<td>3.4.10</td>
<td>WS-Transaction</td>
<td>193</td>
</tr>
<tr>
<td>3.5</td>
<td>SAML</td>
<td>195</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Assertions</td>
<td>197</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Protocol</td>
<td>205</td>
</tr>
<tr>
<td>3.5.2.1</td>
<td>Assertion Query and Request Protocol</td>
<td>207</td>
</tr>
<tr>
<td>3.5.2.2</td>
<td>Authentication Request Protocol</td>
<td>209</td>
</tr>
<tr>
<td>3.5.2.3</td>
<td>Artifact Resolution Protocol</td>
<td>212</td>
</tr>
<tr>
<td>3.5.2.4</td>
<td>Name Identifier Management Protocol</td>
<td>212</td>
</tr>
<tr>
<td>3.5.2.5</td>
<td>Single-Logout Protocol</td>
<td>213</td>
</tr>
<tr>
<td>3.5.2.6</td>
<td>Name Identifier Mapping Protocol</td>
<td>214</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Authentication Context</td>
<td>214</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Bindings</td>
<td>218</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Profiles</td>
<td>226</td>
</tr>
<tr>
<td>3.5.6</td>
<td>Metadata</td>
<td>229</td>
</tr>
<tr>
<td>3.5.7</td>
<td>Versions</td>
<td>240</td>
</tr>
<tr>
<td>3.5.8</td>
<td>Security and Privacy Considerations</td>
<td>241</td>
</tr>
<tr>
<td>3.6</td>
<td>Kerberos</td>
<td>244</td>
</tr>
<tr>
<td>3.7</td>
<td>x509v3 Certificates</td>
<td>246</td>
</tr>
<tr>
<td>3.8</td>
<td>OpenID</td>
<td>246</td>
</tr>
</tbody>
</table>

Chapter 4 Web 2.0

4.1 HTTP | 249  
4.2 REST | 250  
4.3 WebSockets | 251  

Chapter 5 Other SOA Platforms

5.1 DCOM | 253  
5.2 CORBA | 253  
5.3 DDS | 254  

BUY Security for Service Oriented Architectures
http://www.crcpress.com/product/isbn/9781466584020
Chapter 6 Auditing Service-Oriented Architectures 271
6.1 Penetration Testing 272
  6.1.1 Reconnaissance 272
  6.1.2 Injection Attacks 277
  6.1.3 Attacking Authentication 278
  6.1.4 Attacking Authorization 284
  6.1.5 Denial-of-Service Attacks 286
  6.1.6 Data Integrity 286
  6.1.7 Malicious Use of Service or Logic Attacks 288
  6.1.8 Poisoning XML Schemas 289

Chapter 7 Defending and Detecting Attacks 291
7.1 SSL/TLS 291
7.2 Firewalls, IDS, and IPS 294

Chapter 8 Architecture 297
8.1 Example 1 297
8.2 Example 2 300
8.3 Example 3 305
8.4 Example 4 307

Bibliography 317
Index 323
Preface

As applications become more complex and distributed, it is increasingly important that security be considered during the design phases. While there are a lot of books and articles on point solutions that would flow from this integration, such as threat profiling and how to block injection attacks, there is more to consider in the design of an application than how to leverage some of the excellent tools that have been developed to enhance the security of our applications.

Applications, especially those that are distributed across corporate boundaries, benefit from being developed within a comprehensive design or an architecture. While there is a lot of literature on how to develop these software architectures and service-oriented architectures (SOAs), their treatment of security is focused on the use of tools within the architecture.

Information security also benefits from an architecture. However, traditional security architectures are most often focused on infrastructure and consider software as no more than applications that require integration into the policies and standards of an organization, leveraged within approved procedures.

This volume seeks to provide both security and software architects with a bridge between these two architectures, with the goal of providing a means to develop software architectures that leverage security architectures.
Each of the chapters within this volume could be and has been covered in book-length manuscripts. This volume does not attempt to duplicate or condense those texts but rather to draw them together to show how building software architectures within security architectures allows the development of more scalable and resilient applications, which become a trusted platform for their execution of business functionality.

Because architectures are only effective when designed with the capabilities and limitations of the technology in mind, as well as the business goals and design philosophy, this volume will look at, in some detail, how protocols and application components work and interact in the context of the decisions that need to be made in a distributed application. I will explore the various web services standards as well as SOAs which are not web services based but only to expose sufficient detail about the operation of each of the technology frameworks, standards, protocols, messages, and assertions so that an architect is made aware of the decisions that need to be made regarding their approach to application design. I will look at not only the construction of an architecture but also the possible ways by which to attack an architecture by exploiting those same frameworks, standards, protocols, messages, and assertions that businesses depend upon to conduct their business.

None of the standards is explored in sufficient detail to allow this book to be used as source material for application development. Such a book would be beyond any one person to pull together and would be outdated when it is published. In fact, the chapters on penetration are likely outdated as the publisher and I work on bringing the book to print. All good security architects keep themselves up to date on such matters, and this volume represents a point in time, and it is of the past.

However, architecture and the ideas that go into architectural design are not so fleeting. I hope that this book brings together both software and security architects and facilitates their collaboration during the design phases of the next generation of applications.
The service should be able to detect and manage invalid requests as well as repeated messages and handle these gracefully. Most important, and to reiterate, a service must be able to handle messages in an asynchronous manner. Much like the data layer, a service layer benefits from processing message requests in bulk, or in batches, rather than in small discrete quantities.

As a unique interface into the application, a service layer must be concerned with many of the same issues as a presentation layer:

- Authentication and authorization
- Communication
- Exception management

However a service must also manage many of the same concerns as the messaging layer of the application:

- Message construction
- Message end point
- Message protection
- Message routing
- Message transformation
- Message validation

Message validation is a key security issue. Messages should be validated for length, range, format, type, as well as expected data. If all messages are not validated, then the application may not be reliable but nevertheless secure. Schemas can be used to validate data, and in an SOA, a validation service can be developed and exposed.

2.4 Service-Oriented Architecture

To properly understand what an SOA is, it is helpful to understand where it sits as an architecture and the problem that an SOA tries to solve. An SOA, first, is an application architecture. Unlike other applications, in a pure SOA, all application layers comprise services. This is not to imply that the entire application runs at the service layer.

SOA was developed to solve the problem of how to connect disparate systems in a way that they could function together in a systematic manner. These systems, or applications, each provide to the organization the value for which they are designated, but often this
value would be enhanced if either the processing or the results could be integrated with the information processing of other applications.

SOAs are not the first attempt to solve the problem of how best to leverage the capability of disparate systems in a unified and comprehensive way.

2.4.1 Distributed Computing and Services

The message-oriented model (MOM) was developed with the idea of distributing the components of a system among existing and emerging applications and platforms. Messages were used to connect these systems, implemented through specialized platforms called message queues. Message queues are very reliable; however, they require not only a system to manage them but also a staff to maintain the messaging system. Most important, support for the message queues had to be built into all the application components, restricting the implementation of a system built on an MOM architecture to the internal needs of a single corporate infrastructure.

As a software architecture, MOM depends on every component being designed with MOM in mind, instead of leveraging the capabilities of the existing applications in a synergistic manner.

To provide this synergy, each application in an SOA is treated as if it can only perform one specialized function or as if it were an element in the larger software. This specialized function is called a service. Thus, each service functions as an element of the SOA, where the structure and the defined relationship between the elements must transcend application and often corporate boundaries. This allows for the creation and deployment of large distributed applications.

SOAs can leverage messages the same way an MOM can, as a means to transport information from one service to another for processing.

To be considered as a service, the functionality of an application must possess distinct characteristics. First, the functionality to be considered as a service must have complete autonomy; more specifically, this must be limited to a narrow function. It must be able to complete this function without the need for any other component. Second, this function must be directly accessible remotely via some protocol. It is not sufficient for the functionality to be embedded deep within the
application and be dependent on other components to provide the business value. If the application functionality depends on, as an example, additional user input to provide the desired business functionality, it is ineligible to participate as a service unless this can be abstracted and presented to the application as part of the message.

The service must be able to present an informal specification of the purpose, functionality, constraints, and usage of the service. This specification, often called a contract, is often represented in a formal language such as Interactive Data Language (IDL) or Web Services Description Language (WSDL). This contract may provide (though is not required to do so) a detailed semantic description of how to call the service. This semantic description does not need to be limited by WSDL or IDL constraints, even when wrapped by WSDL or IDL.

A service must expose an interface to service clients. This interface will consist of service stubs or application front ends. Front ends may be a web page or an API.

Behind this front end is the actual implementation, the code that performs the actual business logic consisting of application code, configuration, and perhaps data structures.

SOAs that began to be developed as businesses realized that other companies’ applications provided a better solution to meet their requirements than their own internally developed systems. They realized that they could provide better products to their customers if they found a way to leverage the capabilities and expertise of other companies for components that were not part of the core competency of their own company.

As an example, a travel company might recognize that MapQuest’s maps (or Google Maps) were vastly superior to their own, and because their homegrown map application was not how they brought in new business but still provided a desired component to their customers, it would benefit everyone if there was a way to leverage the capabilities of the other companies’ superior product.

SOAs abstract the diverse applications, protocols, systems, and data into four key concepts:

- Application front end
- Service
- Service repository
- Service bus
An application front end is the owner of the business process the application provides and other services can use. It is the interface, the API, the graphical user interface (GUI).

A service is an implementation that provides business logic and data; a service contract that specifies functionality, usage, and constraints, which must be observed by any client; and an interface that exposes the functionality. A service repository stores the service contract of the individual services.

A service bus connects all the services and application front ends with each other. If two participants need to communicate, they do so via the service bus, which uses the service descriptions found in the service contracts to govern how to format the messages, which call the application front end or interface. While similar to the concept of a software bus as defined in Common Object Request Broker Architecture (CORBA), the components of the service bus do not have to have been designed to interoperate in the fashion in which the service bus defines their relationship within the context of the implementation. Each service may have been designed around distinct and alternative business requirements, but because the implementation is exposed as a service, it may be reused to fulfill emerging and new needs.

Thus, each service may serve different roles across different service buses while providing the same discrete unit of business logic, which allowed it to be defined as a service.

There are various technologies that can be applied in the implementation of an SOA. Some of these are more appropriate for internal enterprise-specific projects; others may be applied to any project with any scope. Precisely because of the common architecture, there are issues that transcend the individual technologies that must be considered when establishing a security architecture appropriate for the protection of the business and its objectives as supported by the specific implementation of an SOA. Before we can explore the relationship between the SOA and security architectures, we need to explore specific variations of SOAs.

2.4.2 Process-Oriented SOA

Data integrity is a key factor in the success of many SOAs, as data are transferred from service to service. However, the integrity of data is not sufficient to guarantee that the service returns the correct results.
due to the complexity of business processes that span multiple systems and often involve multiple corporations. A process-oriented SOA is an SOA designed with the primary goal of managing the integrity of process. Process integrity involves not only the integrity of the assets but also their utility throughout the architecture.

The principles of entity, domain, and referential integrity are borrowed, where appropriate, from relational databases. Entity integrity requires that each row in a table be uniquely identified. Domain integrity requires that certain data be within a valid range (such as the date of purchase of an item not being in the future or before the date of which the item was first placed on sale). Referential integrity refers to the validity of the relationship between different data sets, preserving as an example the names of the residents in relationship to their appropriate residence.

Where data must be processed across multiple systems through the use of their services, there is the risk of inconsistencies that impact the validity of both the data and the use of the data in all services. There might be technical failures, business exceptions, and special cases that impact the integrity of the process. Because the process is not centrally controlled, the impact of the failure of any particular component may be high.

There have been a number of techniques applied to solve this problem, each with their own merits and issues. The most common approach is to apply logging and tracing. This is similar to the use of transaction logs in a transactional system, allowing for recovery to a valid state in the event of a failure. The problem is that it is difficult for logging and tracing to resolve issues that relate to logical processes, which span the multiple systems involved in an SOA.

Online transaction processing systems were developed to enable large numbers of users to manipulate shared data concurrently. Such systems are based on the concept of transactions on a unit of work that transforms data from one state to another. A unit of work is presumed to be atomic or indivisible, consistent (move from one consistent state to another), in isolation where no process is visible to any other, and durable where committed updates are permanent. Such systems depend on a central control mechanism that resolves conflicts. Such a central control mechanism is often unavailable to an SOA that leverages services from multiple organizations.
The two-phase commit protocol (2PC) was developed to allow online transactional processing to span multiple systems. A transaction coordinator is implemented as part of a transaction monitor. This enforces that in the first phase of the processing all relevant locks have been acquired and that the state of the data has been properly captured. Dependent on the results of this examination, the transaction coordinator informs the participating systems if they should roll the transaction forward or back. These systems “vote” on how to handle the data; a single abort vote will cause the entire transaction to be rolled back.

All of these mechanisms of tracking changes to data are predicated on some assumptions that often do not apply in an SOA. One is that it is possible to ensure the isolation to the data; another is that the transaction is short term. Neither can be assumed in an SOA, where the various services may be entirely ignorant of access and use of data by other services, and transactions are often long lived. SOAs are also often implemented on discontinuous networks, and none of the above mechanisms are designed to operate under such conditions.

Two techniques scale well to address the issues of process integrity. The first is that of persistent queues with transactional steps. Persistent queues, which follow the data, can guarantee consistency of the individual steps of the process, where errors are handled by a dequeue, and the error is returned with the data. Such systems depend heavily on the presence of a message-queuing system and are more often implemented in internal SOAs where such systems are present.

The second is transactional chains and compensation. Complex workflows are created through individual process steps (transaction chains), where failures are dealt with through compensating transactions that logically undo a prior transaction. Each transaction is made visible to each service so that data may be made available to a compensating control.

Another approach has been the development of a process-centric service, which may or may not use persistent queues or transactional chains. This service operates as both client and server within the service bus and maintains the process state throughout the entire service bus.

2.4.3 Web Services or an Externally Focused SOA

Unlike an internal SOA, a web service-based SOA cannot rely on a single monolithic service bus. To this end, web services are
based on slightly different principles than a traditional SOA. Each service needs to be reusable, stateless, autonomous, abstract, discoverable, and loosely coupled. Instead of a formal service bus, you have a service integration layer, which operates as a logical or virtual service bus.

Services engage with this service integration layer and with each other through a formal contract, which defines the terms of information exchanged and provided supplemental service description. Since services need to be discoverable, they make themselves known through a service provider. Services also need to know which service to call and thus will have a service requester. These roles can be and often will be reversed as the role of the service changes within the larger workflow from client to server. There may be more than one service provider through which a workflow must pass before it arrives at its ultimate destination; these are called intermediary services. Intermediary services may or may not do more than discover the next step in the workflow, depending on the nature of the service and the contract it has as a service provider.

Web services tend to be broken down into one of a set of roles:

- Utility service
- Business service
- Controller service
- Proxy service
- Wrapper service
- Coordination service
- Process service

| Exhibit 2.1 | Web 2.0 and SOA (web services). Abbreviations: BPEL, Business Process Execution Language; REST, Representational State Transfer; SOA, Service-oriented architectures; SOAP, Simple Object Access Protocol; UDDI, Universal Description, Discovery and Integration. |
The nature of the service offered, how to engage it, and the results to be expected are all defined in a specialized XML document present on the service provider. This document will be written in the WSDL. The WSDL functions as the integration layer of the web service, providing the basis for other services to discover how to engage the particular service.

Some implementations of web services will register themselves in a central registry of services using a specification called the Universal Description, Discovery and Integration (UDDI). UDDI repositories provide a marketplace of generic services and are often hosted by major corporations.

Many protocols can be used and are used to communicate between the various web services over TCP/IP. The most common is SOAP. SOAP provides a standard message format that consists of an XML document capable of hosting remote procedure call (RPC) and document-centric data. SOAP can be easily leveraged by both synchronous and asynchronous data exchange models. SOAP, as a raw protocol set, does not define a preset language, allowing the application designer to create a language specific to the architecture.

An alternative to SOAP-based web services is the Representational State Transfer (REST)-based web services. REST leverages the existing vocabulary of the HTTP or other robust application layer protocols with existing vocabularies. SOAs based on REST are easier to implement but less flexible.

With second-generation web services, or Web 2.0, a limited vocabulary was developed to provide a common framework for common constructs that all business services rely on, such as business process or workflow, security, reliability, policies, and attachments. These standards, managed by the Organization for the Advancement of Structured Information Standards (OASIS), are called the WS or web services standards.

The most common kind of web services is the XML web service. Like all web services, the communications of an XML-based web service leverages the Internet protocols for communications but most commonly uses SOAP. Equally important is that all data are expressed and communicated via documents formatted in the XML.

Another commonly found version of web services is called Web API (often called Web 2.0). Web services in the Web API model do not depend on XML for data communications, often leveraging HTML
instead, utilizing HTTP as the transport protocol. Applications built on this version of an SOA have one key distinct assumption not always found in other SOA implementations: that the services are stateless. They use REST, instead of messaging or operations like a more traditional SOA and limit operations to the well-known operations of HTTP (GET, POST, PUT, DELETE).

2.4.4 Enterprise Service Bus

An enterprise service bus is the technical infrastructure of the distributed environment. It is composed of an XML-based communications protocol with a message-oriented middleware core to perform the actual message delivery. There are a variety of message bus frameworks in common use. Message bus frameworks such as Enterprise JavaBeans within the J2EE specification and Microsoft’s .net are based on the capabilities of an application architecture. Others rely on either message queues or object-oriented communication infrastructures such as CORBA. In practice, a successful enterprise service bus is not a single product, no matter how flexible or how many communications protocols it supports, but one that supports accessing services on a meta level, which can leverage the capacities of all application architectures, allowing .net, Enterprise JavaBeans, and other diverse applications to function within a single business process.

2.5 Security Architecture

Security architectures are an abstraction of the protections provided to software and infrastructures to provide confidentiality, availability, and integrity of information. These can be layered on top of software and infrastructure as complementary, or they can be integral to the design and implementation of either or both.

Unlike in many application architectures, SOAs are usually designed with security in mind. However, it is an internally focused security rather than the integration into a larger security architecture.

In the absence of an architecture, security is often relegated to be no more than the security components of an infrastructure, namely, the controls on protocol usage and flow, with a nod to the need to authenticate. Sometimes, software applications will also consider the requirement for authorization as well as authentication.