SOA Security
by Ramarao Kanneganti
Prasad Chodavarapu
Sample Chapter 1

Copyright 2008 Manning Publications
### brief contents

**PART I**  SOA BASICS .......................................................... 1

1. SOA requires new approaches to security  3
2. Getting started with web services  33
3. Extending SOAP for security  84

**PART II**  BUILDING BLOCKS OF SOA SECURITY ............... 129

4. Claiming and verifying identity with passwords  131
5. Secure authentication with Kerberos  173
6. Protecting confidentiality of messages using encryption  209
7. Using digital signatures  260

**PART III**  ENTERPRISE SOA SECURITY ............................ 307

8. Implementing security as a service  309
9. Codifying security policies  356
10. Designing SOA security for a real-world enterprise  397
SOA requires new approaches to security

This chapter covers
- Functional and nonfunctional aspects of security
- New security approaches for SOA
- SOA security implementation choices
Since you are reading this book, you should be a developer, designer, or an architect. But let’s, for a moment, pretend that you are the CEO of a firm.

As a CEO, you want to innovate and raise the top line of your firm. You can do several things. For instance, you can increase sales by understanding customer needs better and creating self-service channels that induce customers to buy more of your products. You can work with your partners better and make it easier for them to sell your products as part of their offerings. Or, you can make your offerings more compelling by bundling them with offerings from your partners. You can also reach out to new customers through targeted campaigns and venues.

As a CEO, you also want to reduce costs and boost the bottom line. Depending on your business, there are several ways you can approach this problem. Like most other CEOs, you can make it mandatory for your CIO and COO to outsource all noncore competencies and reduce IT and operational costs. To meet this mandate, your CIO can consolidate your firm’s application portfolio, outsource day-to-day IT operations and infrastructure management, pay for infrastructure more on a “per use” basis instead of spending a lot of money up-front buying and setting up infrastructure, and move more things out of applications into infrastructure. Similarly, your COO can outsource low value-added operations.

All of these strategies face several common problems during execution. Your enterprise relies on a large number of applications, each developed in a different silo, assuming particular business models. Every time you want to change your firm’s business model—either to innovate or to cut costs—the assumptions built into your firm’s IT applications get in your way and impede your efforts. To add to your woes, every time you try to open up your firm to partners and customers, technology differences between your IT applications and those of your partners/customers get in the way.

To rectify this situation, enterprises are now turning to a new IT architectural approach called Service-Oriented Architecture (SOA). SOA suggests new ways of thinking about how IT systems are designed, used, managed, and combined.

Now that you understand the business motivations for SOA, put your technology hat back on and imagine the consequences of making a fundamental change in your enterprise IT architecture. As you may expect, you will have to review all aspects of building enterprise IT systems, including requirements gathering, design, and testing of applications. One very important aspect you will need to review is security,\(^1\) and that is what we will help you do in this book.

---

\(^1\) By security, we mean application security. Security in IT can be broadly classified as network security, platform security, and application security. SOA has little bearing on the first two. The discussion
In this opening chapter, we will briefly review traditional approaches to security and evaluate them in the context of changes SOA is expected to bring in IT. We will find that traditional approaches to security don’t work well with SOA. We will identify the specific challenges that need to be solved and suggest new approaches that you can use to solve them.

The rest of this book will greatly elaborate and dig into detail on each of the new security approaches we identify in this chapter. Understand the big picture described in this chapter clearly so that you do not get lost in the detail of later chapters.

One important prerequisite for understanding the motivation behind the new security approaches required by SOA is an understanding of the motivation behind SOA itself. We start this chapter by explaining the need for SOA, what SOA is, and how it accomplishes the goals it is intended for. Once we do that, we will briefly describe traditional approaches to different aspects of security and explain why those approaches are not good enough in SOA. We will then introduce to you new security approaches that address the security challenges introduced by SOA. We will close the chapter with a quick overview of the choices you have today to take care of security in your SOA implementations.

1.1 SOA lowers long-standing barriers

Enterprise IT architectures have been evolving organically for quite some time—as business needs grew, different kinds of applications and application infrastructure evolved. All along, the goal of IT was to enable enterprises to take on complex sets of human activities on a large scale in a very efficient way. IT promised businesses that it will enhance productivity and provide competitive differentiation. Somewhere along the way, IT stopped being part of a solution and became more of a problem.

Enterprises with huge investments in IT found to their dismay that their existing investment locked them into specific ways of carrying out business and created barriers to their success:

in this book is limited to application security. The difference between network security and application security is becoming blurred, with the network starting to take on some of the security tasks traditionally handled by applications. We will discuss more about this in chapter 8 when we introduce Application-Oriented Networking (AON).
Applications, developed independently of each other, provided functionality that can only be used within the boundaries of each application.

- Differences between platforms, programming languages, and protocols created technology boundaries that are not easy to cross.
- Security technologies relied too much on enterprise boundaries and created barriers for cross-enterprise cooperation.

All these barriers slowed changes required in business models to react to market changes and increased costs. SOA seeks to solve this problem and return to enterprises the flexibility and the agility they need to win in the marketplace and maximize the return on investments. What magic allows SOA take on this ambitious goal? The magic comes from a few simple principles—basic tenets—that underlie the idea of SOA.

### 1.1.1 Basic tenets of SOA

SOA seeks to solve several issues that come with an application-centric view of IT. The basic principles it enunciates toward this end are:

- Applications must open up their capabilities for use by other existing/new applications. It must be possible to combine the services offered by different applications to create higher-level services or composite applications.
- Technology differences must not matter and interoperability must be a key goal.
- Open standards must be adopted to enable integration across enterprises. Orchestration of business processes across multiple suppliers, partners, and customers must be possible.
- Attention must be paid to governance and manageability in order to make sure that flexibility granted by the first three principles does not lead to chaos.

In other words, SOA insists that traditional barriers to reuse must be lowered. This, of course, has huge implications for security. But before we go into detailing the implications of SOA for security, we should tell you a little more about SOA. We won't go into too much detail, as this book is about SOA security and not SOA. We will simply give you a high-level overview of how it is possible to satisfy the basic tenets of SOA described here.
1.1.2 Idea of a service

Traditionally, applications are designed mainly for use by people. Any provisioning for use of one application’s capabilities by another is often an afterthought, and is usually not backed up by standards. When enterprises need to bring together the capabilities of two or more applications to meet their changing needs, the approaches traditionally used were mostly ad hoc. No one consistent approach was followed. Figure 1.1 depicts this scenario.

The problem with this approach is that it becomes extremely time-consuming and costly to reuse one application’s functionality from another. This limitation severely impacts the ability to combine capabilities of different applications into a new application that may be needed to meet changing business needs.

SOA solves this problem by looking at IT systems as collections of units called services, and not as collections of applications. A service is functionality encapsulated in a form that is readily consumable by other applications and services.

---

**Figure 1.1** In a typical enterprise, applications are built for end users and the mechanisms for applications to interact with each other are ad hoc: database, RPC, files, and so on.
CHAPTER 1

SOA requires new approaches to security

NOTE

What qualifies as a service? Services, if defined correctly, possess desirable attributes that come in handy in overcoming the drawbacks of organizing IT along application boundaries. Some of these attributes are as follows:

- A service is defined at the right level of granularity, from the point of view of a service consumer. In other words, convenience of the consumer drives the service definition.

- A service is self-describing. Potential consumers can learn by themselves how to invoke the service. Service descriptions should include service interface, wire format, transport, location, policies, and SLAs (service level agreements).

- A service is technology-agnostic. Potential consumers are not constrained from using the service because of a mismatch in hardware or software platforms. In other words, an ideal service “interoperates” with every possible consumer.

- A service is discoverable. Consumers looking for a service can discover its presence, usually by looking up a service registry (à la the yellow pages in a phone directory).

- A service can be composed with other services to create a higher-level service. In fact, often several “technical services” are composed into “business services.”

- A service is context-independent. That is, it is usable irrespective of what the caller did before invoking the service.

- A service is stateless, making it easy for service providers and service consumers to create and consume services, respectively.

In contrast to the application-centric view, where application interaction is an afterthought, in SOA, applications are concerned with how to expose services and which services to expose. This allows you to build new applications by pulling together services provided by other applications.

Figure 1.2 shows the state of SOA implementations in enterprises today. Capabilities of existing applications, be they packaged applications or home-grown ones, are exposed as services. Quite a few application vendors are beginning to expose some of their applications’ capabilities as services out of the box. A number of Software as a Service (SaaS) providers such as Salesforce.com are also offering hosted services that enterprises can take advantage of. All of these services are being brought together and managed by a new class of IT infrastructure known as Enterprise Service Bus (ESB). ESB is a term used for distributed omnipresent infrastructure within an enterprise that provides
SOA lowers long-standing barriers

As you can see, SOA shifts IT from an application-centric view to a service-centric view. This shift lets IT build applications that make use of other applications (via the services they provide), making it easy to meet changing business needs by combining functionality from different applications.

**NOTE**  
Web services and SOA  
The idea of a service, as discussed so far, can be implemented in multiple ways. Currently, most service providers prefer to implement their services as web services.
A web service can be defined as any software offering that can be described using the Web Services Description Language (WSDL). WSDL specifies these details about a service:

- What operations the service provides.
- How to invoke those operations, specifically: the format and protocols we can use to send the data for those operations.
- The location (address) of the service.

There is a companion standard awkwardly called UDDI (Universal Description, Discovery, and Integration) that provides for discovery of services and retrieval of their WSDL descriptions. Together, these standards let applications discover and use web services. Several frameworks use the details from WSDL at runtime, providing loosely coupled integration between service consumers and service providers.

The most popular data format and protocol in use for web services are, of course, XML and SOAP, respectively. In chapter 2, we will describe XML and SOAP at the level needed for following the rest of the book. There are alternatives for XML and SOAP that are capable of providing web services, the most popular being REST, described by some of the references provided at the end of this chapter. In our book, we are going to focus on SOAP-based web services exclusively, as they are the most popular ones. There are also other, less-popular ways web services can be implemented. For example, EJBs and database queries can be exposed as services described using WSDL.

Now that we understand what SOA is and what its building blocks are, let’s return to the security problems introduced by SOA’s lowering of traditional barriers to reuse—application boundaries, technology boundaries, and enterprise boundaries.

### 1.2 Lowering of barriers forces us to rethink security

Barriers may be good for security, but they get in the way of business. That does not mean security can be compromised to meet the business goals; we just have to find smarter ways to secure our applications. Traditional security approaches assumed and took advantage of barriers. Since SOA lowers barriers, we must rethink the security approaches.

Figure 1.3 shows the traditional application security architecture. An application manages its own security and relies on secure channels to protect data it exchanges with client applications.
What’s wrong with this picture? The answer is nothing! At least in the context of the traditional application-centric view, this kind of security works well. Before seeing what aspect of this security breaks in SOA, we can observe that there are two implicit assumptions being made here.

- The server application is assumed to know what the appropriate security model is. By security model, we mean who makes the decisions regarding security, and when and how.
- The server application is assumed to be trustworthy enough to see all the data, including any sensitive data that the client is sending.

Now, let us consider an application that is composed of services from multiple applications, as shown in figure 1.4.

First, let us see how the server applications in figure 1.4 differ from the server application in figure 1.3.

- Thanks to SOA, the functionality within an application can be easily recombined with functionality in other apps to create composite applications (such as composite application 1 in figure 1.4).
- Some of these composite applications (such as composite application 2 in figure 1.4) may even combine services in an enterprise with services of partners.
- In addition, services of an enterprise may be invoked directly (like the way client application 3 invokes service 2b) or by partner applications (like the way a partner service p2 in figure 1.4 invokes service 2b).

It is difficult for an application designer to foresee all possible situations in which a service may be invoked. For instance, how is service 2b invoked? It is
CHAPTER 1

SOA requires new approaches to security

invoked in different contexts by different client applications, which means it can never tell how it should handle security. Applications can no longer be in charge of security. That is, security models cannot be hard-coded into applications, as shown in figure 1.3.

Because some or all parts of a message intended for one enterprise’s application may end up in another’s applications, we need to correctly limit the data exposed to each application. For example, parts of messages submitted by client application 4 in figure 1.4 may be sensitive, intended for use only by service 2b. In such cases, a secure channel between client application 4 and service p2 will not be enough, as service p2 will get to see everything submitted by client application 4.

In other words, as application and enterprise boundaries are no longer impediments to reuse, traditional approaches to security no longer suffice. We need to rethink how we secure our applications.

Of course, any new security approaches we come up with should be in alignment with the philosophy of SOA. That is, security should keep the services as open as possible, and as easy to use as possible. Technology boundaries should not be re-erected. Interoperability should not suffer because of security.

Going one step further, it’ll be even better if security architecture is itself service-oriented. To be specific, security infrastructure should be accessible independent
of technology, using open standards. It should be manageable, preferably via policies, a SOA standard.

So far, we are talking in abstraction about security. What exactly are the security concerns we need to address? We can broadly classify them into two groups:

1 Functional aspects of security: These aspects of security are standard in the sense that they exist even with traditional applications as well. These are:
   a Authentication—Verifying identity of users.
   b Authorization—Deciding whether or not to permit action on a resource.
   c Data confidentiality—Protecting secrecy of sensitive data.
   d Data integrity—Detecting data tampering and making sure neither the sender nor the receiver can deny the message they sent or received.
   e Protection against attacks—Making sure attackers do not gain control over applications.
   f Privacy—Making sure the application does not violate the privacy of the users.

2 Nonfunctional aspects of security: These aspects are nonfunctional in the sense that they do not directly relate to security. Instead, they are required to make sure that a security solution works well in an enterprise setting. These are:
   a Interoperability—This concern is specific to SOA, where different security solutions must not break compatibility of services that are otherwise compatible.
   b Manageability—This concern is bigger for SOA, as a security solution needs to protect many different services.
   c Ease of development—This concern is common for any security solution. Be it SOA or traditional application development, complexity reduces adoption of any security solution.

In the next section, we will study the functional aspects of security. In each subsection, we will look into a specific security concern, see how it is traditionally solved, and see why the same approach will not work for SOA.

1.3 Functional aspects of security: With and without SOA

As explained earlier, the functional aspects of security in SOA are the same as in traditional applications. For the most part, they are well understood in the
context of traditional applications. While the security functions themselves do not change with SOA, the approach needed to address each functional aspect is certainly impacted by SOA. Let us discuss these in detail, one by one.

1.3.1 Authentication

Applications need to verify that only legitimate users are trying to use them. The process of verifying the identity of users is referred to as authentication. Broadly speaking, there are three kinds of evidence a user can present to an application to prove identity, as described in table 1.1.

<table>
<thead>
<tr>
<th>Evidence Type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>What you know</td>
<td>You present one or more secrets that the application expects you to know</td>
<td>You may present the username and password that you previously registered with. Alternatively, you may respond to a challenge question that can be only answered correctly by one who knows the secret</td>
</tr>
<tr>
<td>What you have</td>
<td>You prove that you are in possession of something the application expects you to have</td>
<td>You may prove that you are possess a special hardware token such as an RSA token by presenting the number displayed on the token at the time of authentication</td>
</tr>
<tr>
<td>What you are</td>
<td>You prove your identity with biometric evidence</td>
<td>You may present a fingerprint or retina scan</td>
</tr>
</tbody>
</table>

First, let us see the strategy that applications have traditionally used for authentication. Next, we will see how that strategy is inadequate for securing SOA.

*Traditional authentication strategy*

No matter which type of evidence users present to an application, the burden of examining that evidence and validating it traditionally fell on the application.

To fulfill this responsibility, applications need to possess a master copy of the evidence legitimate users are going to present. For example, in order to verify a username/password (or a retina scan), an application needs to know all the correct username/password combinations (or what retina images of all legitimate users look like) up front. Or, if users are using an RSA token, the application needs to know an algorithm that can verify whether the number presented by the user matches the number that is displayed on the user’s RSA token at the time of authentication.
In other words, every single application that needs to authenticate user identity will need a copy of true user credentials. To reduce the administrative complexity, most enterprises store these credentials in a directory server, a repository for user credentials. Most enterprise applications readily integrate with directory servers using a standard protocol named Lightweight Directory Access Protocol (LDAP). LDAP allows applications to consult the directory server for validating user-provided evidence of identity.

Even with the use of directory servers, the onus of ensuring that authentication is carried out still lies with the applications. Based on configuration, most applications contact one or more directory servers and deny access to unauthenticated users.

**Authentication strategy in the context of SOA**

If a service is invoked in different ways, how can we do authentication? For instance, if the service is invoked within the same enterprise, we can use the corporate directory. But, if it is invoked from outside the enterprise, that repository is of no use.

Moreover, when a service uses another service, how does it provide the credentials? Can we trust the authentication done by one service and reuse it? How do we make sure we communicate the results of authentication between services? All these questions are complex, and have no answers that work well in every situation. Table 1.2 presents a few scenarios and possible authentication strategies in each scenario.

**Table 1.2 Possible authentication strategies in different service invocation scenarios**

<table>
<thead>
<tr>
<th>Scenario ID</th>
<th>Scenario description</th>
<th>Possible authentication strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Service is directly invoked by a client app belonging to the same enterprise as the service</td>
<td>Authenticate against the corporate directory</td>
</tr>
</tbody>
</table>
| 2           | Service is invoked by another service or composite app in the same enterprise          | If the caller is another service with the same authentication requirements, the called service can rely on the authentication carried out by the calling service. Otherwise, called service may re-authenticate user identity against the corporate directory.  
If the caller app is a composite app, the called service may rely on the composite app for authentication or do authentication by itself against the corporate directory. |
| 3           | Service is invoked by a partner’s app                                                | Rely on partner app’s assertion of user identity                                                     |
As you can see, the traditional authentication strategy will not work for SOA. Next, we see what authorization means, how it is tackled traditionally, and why the same mechanisms will not work for SOA.

1.3.2 Authorization

Once a user is authenticated with or without the help of a directory server, an application needs to determine whether the identified user is *authorized* to access the functionality she is requesting. Authorization is also commonly referred to as *access control*.

The decision to grant access may depend on multiple criteria, such as the action that is being requested, the resource on which the action is being requested, and the groups to which the authenticated user belongs or the roles that the user plays. For example, the superuser or the administrator may access all the files in a system, but a user belonging to the HR group can access only those files that are allowed for that group.

You can avoid the confusion between “authorization” and “authentication” this way: authentication establishes who you are (like your photo ID may establish who you are) and authorization determines what you are allowed to do (your age and local laws may determine whether you can legally take a drink).

**Traditional authorization strategy**

Most applications come with their own access control model; that is, the logic for deciding whether or not to grant access for a particular action on a resource is hard-coded into the application. Some of the information used to decide whether to grant access is often pulled from a directory server or a configuration repository.

The two most common access control models are *Role-Based Access Control (RBAC)* and *Access Control List (ACL)*.

In RBAC, permissions for each action on a resource are granted to one or more *role*. For example, in an e-learning application, a teacher role is required to grade a test. Information on what roles are granted to which users can be maintained in an LDAP directory.

ACL-based access control works differently. Administrators associate a list of rules with each resource. A rule declares whether to grant or deny permission for a specific action on a specific resource. For instance, in an e-learning application, each test may be associated with a set of rules describing which user can do what actions.
Authorization strategy in the context of SOA

Think of a composite application that stitches together the capabilities of multiple services. As an action in the composite app may consist of multiple actions in constituent services, the composite app should ideally check the access control rules of all constituent services before initiating an action. But this is only possible if the access control rules of each constituent service are also available to the composite application. This is not possible, in general; traditionally, access control rules are built into each application in an opaque way.

There is another reason why we cannot hard-code a specific access control strategy into each application. The reusability of the service may be drastically reduced for use cases that require a different access control model. Any authorization strategy for SOA will have to address these issues.

Next, we will see how data confidentiality is impacted by SOA.

1.3.3 Data confidentiality

Data exchanged over a network needs to be safeguarded from prying eyes. Unauthorized parties can otherwise gain access to data over the network by using sniffer tools.

Traditional strategy to ensure data confidentiality

Encryption is the standard technique used to safeguard confidentiality of data exchanged over a network. Without encryption, web commerce would never have succeeded, as it would have been unsafe to transmit credit card information over the wire. Part II of this book will introduce the fundamentals of encryption. For now, it is enough to note that encrypted data can only be understood by parties that know the encryption scheme and the decryption key.

Traditionally, applications that care about data confidentiality establish a secure channel for data exchange using Secure Sockets Layer (SSL)/Transport Layer Security (TLS). SSL/TLS encrypts all the data exchanged over a channel.

Data confidentiality protection strategy in the context of SOA

SOA makes it easier for enterprises to integrate their service offerings with those of partners, and this of course fulfills a real business need. There is now additional complexity: customer information will now cross enterprise boundaries. That will

---

2 Data stored by an application—be it in memory or in files/databases—also needs to be secured against unauthorized access. We don’t discuss this aspect here, as SOA does not in any way impact the strategies used to secure stored data.
make the traditional data confidentiality strategy less useful. Consider the example shown in figure 1.5.

In this example, the ACME brokerage firm is integrating with a bank to make it easier for customers to pay for their stock purchases. Customers who wish to take advantage of this integration will have to trust ACME with their bank account information, which is not a good idea, in general. It imposes security risks, responsibilities, and even legal obligations that those enterprises (the ACME brokerage, as well as the bank in this example) are not willing to accept.

Observe that SSL/TLS is not enough to address the data confidentiality concerns in this use case. SSL/TLS can protect the confidentiality of a customer’s

Figure 1.5  John, a customer of the ACME brokerage firm, is placing an order. As ACME has integrated its applications with the payment services offered by John’s bank, John can pay for his order directly from his bank account. John attaches his bank account information to his order and sends both securely to ACME. ACME then places a money transfer request with John’s bank. Note the security hole in this arrangement. ACME knows the details of John’s bank account and an administrator at ACME may be able to misuse this information.
message when it is passing over the wire, but once the message reaches the brokerage application, SSL/TLS’s responsibility ends and the application is free to read and use all the data in the message. Clearly, we need better techniques to ensure data confidentiality.

Now, we will examine how the data integrity aspect of security becomes complex with SOA.

1.3.4 Data integrity and nonrepudiation

When an application receives a message, it needs to make sure that the message received is exactly what the sender sent and not something that is fabricated/tampered with by a man in the middle. In other words, applications are responsible for verifying integrity of data received over a network. Furthermore, the sender should not be able to “repudiate” or deny having sent a message.

These requirements are important for both senders as well as receivers of data over a network. Suppose you authorized the withdrawal of a certain amount from your bank account for purchases at an online merchant. You, the sender, must be sure that the merchant, or even the bank, did not modify the amount in your authorization.

From a recipient’s point of view, you must protect yourself, as a merchant, against customers who falsely report fraud claims against you. As a recipient of a sensitive email, you must be sure that the message was indeed sent by the person whose address appears in the mail headers.

Traditional strategy to ensure data integrity and nonrepudiation

SSL/TLS, used by most applications to ensure confidentiality of data exchanged over a network, also helps in verifying the integrity of data received over the network and ensuring nonrepudiation. In part II we will describe the underlying techniques that help verify data integrity and guarantee nonrepudiation.

Data integrity protection strategy in the context of SOA

As with data confidentiality, the traditional strategy of using SSL/TLS to protect data integrity and nonrepudiation is not enough when higher-level services bring together lower-level services from different parties. To understand this easily, consider the use case illustrated in figure 1.5. A secure channel provided by SSL/TLS cannot prevent the brokerage application from claiming a different amount from the bank than what the customer authorized. We need new and better ways of ensuring data integrity and nonrepudiation.
1.3.5 Protection against attacks

Every network application needs to be protected against attacks. Broadly, attacks seek to exploit three kinds of vulnerabilities:

- Vulnerabilities in application code: For example, failure to vet user input before using it in an SQL query can allow attackers to trick the application into executing arbitrary SQL code.\(^3\)

- Vulnerabilities introduced by poor administrative practices: For example, if an administrator does not change the default superuser password for a publicly exposed server application, attackers can walk right in through the front door and take charge.

- Vulnerabilities inherent in computing/networking infrastructure: TCP/IP has many known weaknesses. In addition, TCP/IP implementations and operating systems often come with their own set of vulnerabilities that can be exploited by attackers.

Let’s see how we can address these vulnerabilities in traditional applications as well as in SOA.

**Traditional strategy for protection against attacks**

Traditional strategy for protection against attacks is multifold

- Using firewalls, isolate publicly exposed applications from internal applications as much as possible.

- Run applications within sandboxes (such as chroot jails) to limit damage of compromise.

- Carefully audit application code.

- Use intrusion detection systems to monitor application activity.

How well do these techniques work for SOA?

**Strategy for protection against attacks in the context of SOA**

With SOA, more and more code gets exposed—directly or indirectly—to external parties, and attack opportunities multiply. The very openness of services can make it vulnerable to denial of service (DoS) attacks. There is nothing inherently special about SOA that makes it more vulnerable to these kinds of attacks. We will not focus on much on this kind of security concerns in this book.

---

\(^3\) This kind of attack is known as SQL Injection attack and is described in more detail in chapter 10.
1.3.6 Privacy protection

Businesses are often prohibited by law or by contract from disclosing the personal/private information of customers/partners to third parties. This prohibition applies as much to applications as it does to humans. In other words, business applications must be carefully designed and managed to avoid leakage of users’ private information.

Traditional strategy to protect privacy

Leakage of private information happens primarily due to two simple reasons.

- Flaws in access control rules: for example, an e-learning application screen that exposes the social security numbers of students enrolled for a class should not be openly accessible; only administrators should be allowed. Traditionally, flaws in access control rules were eliminated by careful reviews and security audits.

- Vulnerabilities exploited by attackers: vulnerabilities may lie in application code or may result from poor administrative practices. For example, failure of application code to vet user input may be exploited by attackers to inject SQL that queries sensitive information from a database. Or, the failure of an administrator to change the default superuser password may open up all the data managed by an application. Traditionally, damage from vulnerabilities is contained by good security management practices, security audits, and intrusion detection tools.

In addition to these two simple ways, there are several complex ways (such as triangulation) in which privacy can be compromised. We will not go into them in this book.

Privacy protection strategies in the context of SOA

With SOA, privacy protection is lot more challenging. Since SOA makes it easier to compose services from multiple parties, the chances of unwittingly sharing your customers’ private information with external parties are increased.

Privacy protection is not a topic we focus on too much in this book, so we will give away right here the basic idea behind privacy protection strategies in SOA.

When invoking a third-party service, service providers can the mask real identities of users using pseudonyms. Holding back real identities is not enough. It is possible to guess identity based on patterns of usage. Protecting against such leaks is difficult and is not often attempted.
We have now completed a review of the different functional aspects of security, all of which are impacted by SOA. New strategies are needed to address these aspects in the context of SOA.

The impact of SOA on security extends beyond the functional aspects of security. SOA forces us to pay more attention to some nonfunctional aspects of security as well. We will describe this impact next.

1.4 Nonfunctional aspects of security

For a security solution to be successful, it is not enough to take care of all the functional aspects of security. We need to address a few nonfunctional concerns as well, so that the solution is scalable, manageable, and usable. While these concerns exist even for the traditional applications, for SOA, they are very pronounced. We will explain why in the next subsections.

1.4.1 Interoperability

As described in section 1.1.1, one of the basic tenets of SOA is that technology differences must not matter. A service should be usable by any service consumer, regardless of what technology platforms the service and service consumer run on. In other words, interoperability of services and service consumers is a major promise of SOA. Security approaches we adopt for SOA must not break that promise. If everyone adopts a different approach for securing services and service consumers, interoperability will certainly break. But even with standardized approaches, interoperability is not guaranteed because, for good reasons, standards have to leave some choices to implementers. For example, the choices of authentication mechanism and cryptographic algorithm have to be left to the implementers, otherwise the standards run the danger of not being applicable in all situations.

Traditional strategy to ensure interoperability

As interoperability between applications was itself not guaranteed, interoperability of security implementations was traditionally not a topic of great interest. Interoperability needs have been addressed to varying degrees in traditional security strategies. For example:

- Most applications supported only one form of authentication.
- HTTP supported a few standard authentication schemes.
- SSL/TLS allowed for dynamic negotiation of cryptographic algorithms.
As you can see, this issue is not very important in application security as it is self-contained.

**Interoperability needs in SOA security**

With SOA, interoperability not only becomes an important goal, but also a tougher goal to accomplish. Expectations will be that a service implemented on any platform can be secured without affecting interoperability with a wide variety of service consumers. Also, as SOA allows composition of services from a number of parties, the likelihood of interoperability problems will be higher.

Another important nonfunctional aspect of security is manageability, which is of particular importance to SOA. Let us examine that next.

### 1.4.2 Manageability

Manageability refers to the ease with which a security implementation can be managed. By “managing,” we mean all aspects of dealing with a security implementation, including configuration, provisioning, auditing, and maintenance.

**Traditional strategy to ensure manageability**

Manageability of application security solutions has traditionally been poor to fair. The most pressing pain points in application security management have been user account provisioning and password synchronization. The common strategy to tackle these problems is:

- Where possible, modify applications to authenticate against a common credential store such as a directory
- Otherwise, use tools that synchronize user accounts and passwords across multiple applications

One additional management concern that has become commonplace recently is auditing. Recent regulations in the U.S. and Europe have made it necessary to not just secure but also prove that enterprises are doing all that they can to secure their applications. Traditionally, there is no easy solution to this problem. It has proved quite expensive for enterprises to install auditing controls in existing applications.

**Manageability needs in SOA security**

As interactions between applications multiply with SOA, it becomes difficult to prove that the security decisions being made by all the applications are correct
and comprehensive. In other words, security audits now become much more difficult unless we improve on the traditional approaches to manageability.

In addition, provisioning of users and user privileges in applications becomes difficult due to the increase in the number of ways applications can be used.

1.4.3 Ease of development

Usually, security is seen as an esoteric topic that is better left to experts. While it is true that security requires expert attention, everybody, including software developers, designers, architects, IT administrators, and managers, needs to do tasks that require a good understanding of security topics. If a security strategy is not easy to understand and develop, it cannot be successful.

Traditional strategy to ease burden on developers

Fortunately, traditional security techniques have been around long enough for designers, developers, and architects to understand and apply them in practice. In addition, a number of libraries, frameworks, and other facilities reduce the burden on developers. For example, J2EE application containers assume a lot of the responsibility for security.

Ease of development needs in SOA security

If you followed this far, you understand that any solution for SOA security is bound to be more complex than any traditional security solution. Therefore, it becomes more important than ever to make sure that SOA security solutions are easy to use. Thankfully, this is not a subjective measure; there are specific steps, which we’ll show in the next section, to make SOA security solutions easy for developers to use.

Now that we have discussed the different ways SOA security can be more complex than traditional security, it is time to offer a few words of optimism. Despite all these difficulties, SOA security is not impossible—thanks to the support of standards, tools, and technologies. As tools and frameworks mature, SOA security becomes easier to understand and implement.

In the next section, we will describe three approaches to SOA security. They solve different aspects of security, and can be combined. In fact, any framework or solution you develop or buy will have some elements of these approaches.
1.5 **New security approaches for SOA**

It is clear from the discussion so far that traditional approaches to various aspects of application security will no longer suffice. By lowering long-standing barriers between applications, SOA forces us to rethink how we approach security. At the same time, SOA fortunately allows a few new approaches, thanks to the standards it supports, that fit the changed requirements of application security discussed previously. In this section, we describe three such new security approaches:

- Message-level security
- Security as a service
- Policy-driven security

Let’s start with a description of message-level security.

1.5.1 **Message-level security**

As you might recall from sections 1.2.3 and 1.2.4, SOA brings changes in the requirements for data confidentiality and data integrity. As illustrated in figure 1.5, when a message sent to party 1 (brokerage) contains parts intended for party 2 (bank), we need the ability to differently encrypt and/or sign the part that is intended for use only by party 2. Clearly, traditional transport layer security mechanisms such as SSL/TLS are not good enough here, as they cannot stop party 1 from reading and/or tampering with the message part intended for party 2.

Message-level security (as opposed to transport-level security) is a new approach to solve this problem. With this approach, different parts of a message can be protected differently, to make them usable only by intended parties in the message path. Figure 1.6 illustrates this idea in the context of the brokerage and bank example introduced by figure 1.5.

In this figure, John separately protects a part of the message he is sending to the brokerage, as he wants that part to be usable only by his bank. This idea of protecting different parts of a message differently to make each part usable only by intended parties is what is commonly referred to as message-level security.

Part II describes how a standard named WS-Security allows message-level security to be implemented with SOAP.

Remember that wish we made at the beginning of the chapter? We wanted our SOA security solution to be in line with the basic principles of SOA—the security solution itself should be usable as a service. We will describe this approach next.
CHAPTER 1
SOA requires new approaches to security

1.5.2 Security as a service

In sections 1.3.1 and 1.3.2, we discussed how applications can no longer be in charge of authentication and authorization, for they might not know the context in which they are being used. The natural question is: who else can do authentication and authorization? The answer, in keeping with the spirit of SOA, is a security service.

A security service can offer applications the ability to authenticate, authorize, encrypt/decrypt messages, sign messages/verify signatures, and log messages. It may also scrub messages to protect applications against known and unknown vulnerabilities. Applications might still need to know a little bit about security—for example, they may need to know how to invoke a security service and use the information provided by the security service in return—but the meat of the security logic can be executed by a central security service.

Figure 1.6  The message from John to the bank is in a different envelope inside the envelope to the ACME firm. ACME can only open its envelope, so the letter to the bank is safe from everybody else.
The idea of a security service is in some ways similar to the idea of an application service, and in some ways different. Like an application service, a security service should be usable by any application; technology differences should not be a barrier. Unlike an application service, a security service is infrastructural and may come into play even if it is not explicitly invoked. For example, as shown in figure 1.7, the security service may be implemented as part of the ESB or by application-aware network devices.

As a security service is central, and not a part of any application, its security model can evolve in-line with business needs, without affecting any of the applications. Treating security as a service helps offload much of the burden of security enforcement from developers of business services, allowing them to focus on business logic.

Implementing security as a service was not possible until recently because of the absence of technologies and standards for communicating with a security service. While LDAP did enable applications to rely on a central directory server for authenticating users and retrieving user attributes, applications still had to implement their own logic for authorization and other aspects of application security that we discussed earlier in this chapter.

![Figure 1.7](image_url) Security can be implemented as an infrastructure service, for example, offered by the ESB. This approach avoids the problems that arise when particular security models are hard-coded into applications. See also figure 1.4.
CHAPTER 1

SOA requires new approaches to security

SOA supports standards such as Security Assertion Markup Language (SAML) and WS-Trust that can be used to implement such a service. Chapter 8 describes SAML, WS-Trust, and other technologies that help offer security as a service in web services implementations.

No matter how good a solution is, unless it is easy to use and manage, it will not be successful in the enterprise. In the next section, we will describe an emerging approach to make SOA security easy to develop and manage.

1.5.3 Policy-driven security

In sections 1.4.1 to 1.4.3, we discussed the nonfunctional needs of SOA security solutions. Policy-driven security is a new approach that is evolving to address those needs.

The idea behind policy-driven security is simple. Security requirements and mechanisms must not be hard-wired into applications. Instead, security requirements of an enterprise should be declared separately as a “security policy.”

A security policy declaration becomes handy in many ways: It separates security logic from business logic, leaving the former to security specialists. It becomes easier to ensure consistency of security enforcement across multiple applications. Most importantly, interoperability is enhanced as policies of involved parties can be compared to figure out how to make their security implementations compatible.

Chapter 9 describes the WS-SecurityPolicy standard that is emerging to make this new approach possible.

In this section, we have introduced three new approaches that address the impact of SOA on various aspects of security. As a SOA practitioner, you need to know how you can put these approaches into action to tackle SOA security. If you are an application developer using a web services toolkit in Java, .Net, or a legacy platform, you need to know if and how your toolkit supports these approaches. If you are an IT administrator or an enterprise solution architect, you need to know about products that help you implement frameworks based on these approaches across the enterprise. We will discuss what’s available in current generation SOA products next.

1.6 Current SOA security implementation choices

This book will teach you the details of each of the three new approaches described in the previous section. You will see sample Java code that helps you to understand how most of the relevant standards can be implemented.
While you can definitely write custom code based on the samples shown in this book, there are better options. Table 1.3 lists the options you have under various scenarios.

### Table 1.3  Currently available options for implementing SOA security in different scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Options</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Securing a single service built using a web services toolkit that supports policy-driven security</td>
<td>Select one of the precrafted security policies offered by your web services toolkit, if it meets your needs.</td>
<td>BEA WebLogic Server (WLS) 9.1 provides three “simple” policies and .NET WSE 3.0 provides five policies (called turnkey security profiles). While prepackaged policies in WLS 9.1 support authentication using username/password and digital certificates, those in WSE 3.0 also support authentication using Kerberos tickets. Both restrict encryption/signing to the whole message body; if you want to encrypt/sign different parts of the message differently, you cannot use the prepackaged policies. You also cannot use other forms of authentication (such as SAML assertions, introduced in chapter 8) in either case.</td>
</tr>
<tr>
<td>If none of the precrafted policies offered by your web services toolkit meets your needs, declaratively specify your own security policy using a toolkit vendor-specific policy language that is usually a variation of WS-SecurityPolicy (described in chapter 9). Toolkit vendors also allow you to extend what you can enforce with a policy by coding your own assertions.</td>
<td></td>
<td>BEA WLS 8.1 and .NET WSE 2.0 are examples of web services toolkits that offer this level of support.</td>
</tr>
<tr>
<td>Securing a single service built using other web services toolkits</td>
<td>Look for extensions or code your own extensions.</td>
<td>If you are using Apache Axis, an open source web services toolkit, publicly available extensions named Apache WSS4J (for Axis 1.x) and Apache Rampart (for Axis2) will let you handle, with a combination of declarations and code, usernames/passwords and encryption/signing of entire message bodies. For everything else, you will have to code your own extensions.</td>
</tr>
</tbody>
</table>
CHAPTER 1
SOA requires new approaches to security

These are only some of the options. No doubt as the tools and frameworks mature, you will have even more options. You will need a good understanding of security issues and the strengths and weaknesses of different security strategies in order to pick the right tools and technologies. That understanding is what we seek to provide in this book.

1.7 Summary

SOA is changing long-standing assumptions about how applications are used and combined, and these changes impact the ways in which we can secure applications.

Traditionally, functional capabilities of an application were by and large only available within the context of that application. Other applications could not easily reuse them, either because they were not designed to be reusable or because technologies used by different applications did not work well together. This fact slowed the pace at which IT departments in enterprises can respond to ever-changing needs of businesses.

SOA fixes this problem by asserting that the capabilities of an application must be exposed as “services” that can be easily reused by other applications. But in the process, SOA opens a new can of worms with respect to security. As we explained in this chapter, traditional approaches to addressing different aspects of security do not work well in a world where application, technology, and enterprise boundaries are rendered insignificant by SOA.

Security becomes a serious concern for enterprises seeking to adopt SOA on a large scale. To guarantee security, people naturally tend to be conservative and impose cumbersome restrictions on the use of services. Such an approach negates

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Options</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Securing a large number of services within an enterprise</td>
<td>Consider deploying security infrastructure that takes care of security for your services. A natural place for such security infrastructure is in your ESB. But, not many ESB implementations currently offer comprehensive implementations of SOA security standards. You will need to augment your ESB with a security appliance that specializes in SOA security. Or, you can extend the vendor implementations with your own code.</td>
<td>Examples of ESB are BEA AquaLogic Service Bus and IBM WebSphere ESB. Example of specialized security appliances are IBM DataPower XML Security Gateway, Cisco AON Modules, Reactivity XML Gateway, Layer 7 SecureSpan XML Firewall.</td>
</tr>
</tbody>
</table>
the benefits of SOA. Fortunately, there are new approaches enabled by SOA that can ensure security without diluting the advantages SOA can bring.

In this book, we present three new approaches to security. The first uses message-level security instead of transport-layer security to ensure confidentiality and integrity of data even when multiple parties are involved in a message exchange. The second looks upon security itself as a service in order to decouple security logic from applications. The third uses declarative security to guarantee interoperability and manageability. Elements of these approaches are present in most commercial, open-source, or home-grown solutions.

Naturally, you need to know about the new tools, standards, and technologies that make these three new approaches possible. In the rest of the book, we will introduce all of these with working examples. Along the way, we will also describe several simple techniques that solve specific small problems.

In the next chapter, we will cover the basics of SOA, where you will create and execute your first service. As explained earlier, you will learn the theory followed by actual code that you can modify and see in action.

**Suggestions for further reading**

- Apache WSS4J, a publicly available extension of Apache Axis 1.x to add limited support for security, is available at http://ws.apache.org/wss4j/.
CHAPTER 1
SOA requires new approaches to security

- Apache Rampart, a publicly available extension of Apache Axis2 to add limited support for security, is available at http://ws.apache.org/axis2/modules/rampart/1_0/security-module.html.


- DataPower (acquired by IBM), Sarvega (acquired by Intel), Layer 7 Technologies, Reactivity (acquired by Cisco), and Forum Systems are some of the firms selling XML security devices. The home pages of these firms are at http://www.data-power.com/, http://www.sarvega.com/, http://www.layer7tech.com/, http://www.reactivity.com/, and http://forumsys.com/ respectively.
To secure an SOA solution you cannot just secure the participating applications—you must also ensure security of the services that underlie the whole thing.

But, how do you do that while keeping services open and easy to use by both internal and external applications? This is one of the many questions answered in this book.

**SOA Security** is written for architects, designers, developers, and IT managers. Part 1 covers the basics of SOA—best practices, toolkits, and techniques. You will find it useful, even if you are familiar with SOA. Part 2 deals with the building blocks of SOA security: authentication, authorization, nonrepudiation, and more. Part 3 shows how to build industrial-strength solutions, introducing you to real-life problems and frameworks to solve them. It is accessible to IT managers and particularly useful to architects.

This book is down-to-earth. It covers many kinds of security situations, from casual to complex. It combines concepts and practice. Its examples are in Java and the Apache Axis toolkit, but .NET and other users should have no trouble getting the message. And for those using commercial security solutions, it provides valuable theoretical background for the practical issues they encounter, including why and when they may need to implement a missing functionality.

**Dr. Ramarao Kanneganti** is CTO at HCL EAI Services. He has worked at Bell Labs in databases and large programming systems, and currently advises enterprise clients on SOA strategies. Rama works out of Grosse Pointe Woods, Michigan.

**Prasad Chodavarapu** is a manager at HCL EAI Services, Bangalore, India. He leads teams designing and deploying integration solutions at enterprises across the globe.

For more information, code samples, and to purchase an ebook visit www.manning.com/ SOASecurity