Security Issues with SOAP

Jim Clune and Dr. Adam Kolawa
ParaSoft Corporation

Formerly known as Simple Object Access Protocol, SOAP is rapidly becoming the standard for building Web services and connecting disparate systems in a loosely coupled fashion with complete platform independence. However, some of the very features that make SOAP attractive, such as its flexibility and its compatibility with HTTP, also provide opportunities for security breaches. This article discusses SOAP security issues and how they can be addressed.

SOAP (formerly known as Simple Object Access Protocol) is a lightweight protocol for exchanging structured and typed information in a decentralized, distributed environment. It is an XML-based protocol that consists of three parts:

- An envelope that defines a framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

SOAP makes possible a universal platform for Web-based applications that transcend the boundaries of a specific programming language and/or specific platform. With SOAP users/adaptors growing by the day, SOAP is rapidly becoming the standard for building Web services and connecting disparate systems in a loosely coupled fashion with complete platform independence.

SOAP was developed by Microsoft, DevelopMentor, and Userland Software and proposed as an XML protocol to the World Wide Web Consortium (W3C). The name reflected the idea that SOAP would be used to express serialized object graphs, enabling object-oriented systems to perform functions such as remote procedure calls while preserving objects and their relations. However, in the W3C's latest working draft (version 1.2), SOAP became the name and is no longer an acronym. This reflects a shift in thinking about SOAP from a serialization framework for object-oriented systems to a more general XML-based messaging paradigm, where the messages do not necessarily contain objects.

SOAP is a key technology enabling the development of *Web services*, which is a term that has emerged to describe a software module deployed on the Web and intended for use as a component in one or more applications distributed across the Internet. The promise of Web services is to facilitate the creation of open distributed systems that leverage networks by aggregating multiple services and providing higher levels of functionality.

Unlike the distributed computing protocols that preceded it, SOAP is simpler, more flexible, and facilitates looser coupling between the components. For example, the Object Management Group's Internet Inter-Object Request Broker Protocol (IIOP) is the underlying transport mechanism used by the Common Object Resource Broker Architecture (CORBA). Microsoft's Distributed Component Object Model (DCOM) is a distributed computing protocol that extends Component Object Model (COM). Whereas IIOP is tightly coupled to CORBA's heavyweight architecture and infrastructure, and DCOM is tied into Microsoft's COM architecture, SOAP is not tied to any corresponding architecture or infrastructure.

"Depending on how your software is configured, a remote operator could access your system and provide instructions to your server."

SOAP is a stateless, one-way messaging paradigm. Although more complex interaction patterns can be built on top of SOAP, the protocol is not tied to objects or an infrastructure managing them. SOAP is built on XML, which lends itself to cross-platform interoperability. For the transport layer, SOAP commonly uses HTTP, another text-based communication protocol that has gained wide acceptance, as is evident from the state of the Web today.

However, the very attributes that make SOAP so attractive also give cause for some concern. The ports that serve as integration points for business partners can serve as entry points for unwanted elements, such as hackers and viruses. Depending on how your software is configured, a remote operator could access your system and provide instructions to your server. This is a security issue. SOAP's openness and flexibility, the very things that make it so powerful, can enable attackers to wreak havoc on your system.

How can you protect yourself? The rest of this article explores and addresses the security issues that are relevant to existing technologies. Some of these challenges are applicable to a number of existing protocols, but for the scope of this article, we are focused specifically on SOAP. After addressing these issues, a discussion of the challenges beyond the mainstream solutions will follow, as well as ideas for meeting these challenges.

Security Issues and Priorities

Security is not a single problem, but rather a host of interrelated issues. For any given application, some of the issues will be critical, while others may be of lower priority or even irrelevant. Here are some facets of security that are worth considering when deploying SOAP services:

- Privacy: For many services it is important that messages are not visible to anyone except the two parties involved. This means traffic will need to be encrypted so that machines in the middle cannot read the messages.
- Message Integrity: This provides assurance that the message received has not been tampered with during transit.
- Authentication: This provides assurance that the message actually originated at the source from which it claims to have originated. You may need to not only authenticate a message, but also prove the message origin to others. This is called non-repudiation. Non-repudiation is a legal

concern as well as a technical issue, and achieving it is beyond the scope of this article.

- Authorization: Clients should only be allowed to access services they are authorized to access. Note that authorization requires authentication, because without authentication, hostile parties can masquerade as users with the desired access.

The first step in implementing security is to determine which aspects are important for your organization. It is also helpful to have some idea of their priorities with respect to each other as well as in terms of non-security related goals such as quality and performance.

Implementing Security with Current Technology

SOAP is a young technology, so the tools and techniques for using it are still evolving. However, engineers under a deadline need to understand what is available *now*.

In order to achieve security goals, we will first look at HTTPS, which is HTTP over the secure socket layer (SSL). The SSL is widely used on the Internet today. It performs public key encryption to address the privacy aspect of security. It also performs a message integrity check using a keyed message authentication code.

When it comes to authentication, the story gets a little more complicated. SSL uses certificate-based authentication, and the certificates for the server and the client may be controlled independently. By far, the most common usage is for the server to have a certificate and the client not to have a certificate. In this scenario, the client has assurance of the server's identity, but the server does not have assurance of the client's identity. However, SSL may also be configured so that both the server and the client require trusted certificates.

Certificates for SSL are often, but not required to be, in a chain with the root certificate from a well-known trusted authority such as Verisign, Thawte, Entrust, etc. HTTPS can be used in conjunction with other types of authentication such as HTTP Basic and Digest authentication. By itself, Basic authentication is an extremely weak protection scheme since it involves sending username and password in base64 encoded plain text. For some applications this is sufficient. Digest is considerably better because it involves a challenge/response mechanism where the password is not sent directly. Combining either of these with HTTPS makes for a significantly more secure connection since encryption provided by the SSL prevents hackers from spying the passwords and reusing them.

Once you have established a means of authentication, you need to establish authorization procedures. It is helpful to think of authorization in two broad categories: *declarative* and *programmatic*. Declarative authorization typically involves specifying which groups various users belong to and which groups can access each service. Here the membership of each group determines the complete specification, so coding is either trivial or nonexistent. Programmatic authorization involves obtaining the user, group, or role at runtime and using that information to perform some logic about what to do next. Programmatic authorization is more flexible in that you have more options about what criteria to use to reject a request as well as what action to take if a request comes from a non-trusted party. The trade-off is that programmatic authorization is more complex, involves writing code, and provides more opportunities for error.

"Breaches in security are often the result of false assumptions. The most dangerous are the ones that are implicit and unspoken."

Example: A Financial Institution

Although SOAP-based Web services can be deployed in a wide variety of languages and platforms, the principles used are best illustrated by a concrete example. In our discussion, we will be developing our services in Java, utilizing the Apache SOAP implementation, and deploying them over HTTP using the Apache Tomcat servlet container. Each of these tools is freely available for commercial use.

We will use the example of a financial institution using SOAP-Remote Procedure Call (RPC) for business-to-business integration. Before we jump into the security issues, we will briefly review the relevant pieces for implementing this scenario. A simple method that a financial institution may want to expose is: getAccountBalance(account Number), which takes an account number as an input and returns the current account balance. The Java method signature may look like this:

```
public int getAccountBalance
(int accountNumber) throws
InvalidAccountException {
// Perform database query and
return result.
...
}
```

This describes the interface for a Java application, but for SOAP we need to translate this into appropriate SOAP terms. The current approach is to create a Web Service Description Language (WSDL) document. WSDL is another XML-based language that has been proposed to the W3C. It is used for describing services as a set of endpoints operating on messages. This WSDL then becomes the published interface for business partners utilizing the service. Deploying the implementation requires configuring our components (in our case, Tomcat, Apache SOAP, and our implementation Java class working together).

Deploying the service over HTTPS requires only minor modifications to the HTTP. First, you will need a certificate for the server. This certificate is the identifier that enables clients to authenticate the server. Java manages private keys and their associated certificates in keystores. Conceptually, keystores are databases of key entries and trusted certificate entries, though they are often implemented in files rather than relational databases. Java also provides a tool for managing keystores called keytool. Keytool generates self-signed certificates as well as certificate signing requests, which are sent to a certificate authority.

Next, both the interface and the implementation must be modified to reflect the change in protocol. For the interface, change the WSDL to specify the HTTPS protocol in the RPC router, which indicates where to route the remote procedure call. For the implementation, enable an HTTPS connector in the Tomcat server configuration file.

The client will also need to make some minor changes. The URL changes to reflect the use of HTTPS instead of HTTP. (This will happen automatically if the client is WSDL-aware.) For accepting certificates, Java again uses a keystore. The keystore should contain the trusted certificate entry corresponding to the certificate associated with the server. This assures the client that the response really comes from our financial institution and not a malicious party intercepting the request. The keystore can be configured statically using keytool or it can be accessed and manipulated programmatically at runtime in the client.

Beyond the Fundamentals: Everything You Know Is Wrong

Breaches in security are often the result of false assumptions. The most dangerous assumptions are the ones that are implicit and unspoken because these are made subconsciously, so they are never directly challenged and scrutinized. Here we present some ideas to challenge the reader's assumptions. In some respects these ideas are simply common sense. However, in the words of the French philosopher Voltaire, "Common sense is not so common."

There Is No Guaranteed Security

Public key cryptography systems are a very good technology, but they are not a panacea. What takes 50 years to break using brute force on today's most powerful supercomputer may take three seconds after an unexpected breakthrough in quantum computing. More pointedly, it does not matter how long brute force takes if the hacker does not use brute force. The easiest way to circumvent public key encryption is to gain physical access to a computer containing the private key. When someone tries to tell you security is guaranteed, remember that whenever humans are involved, guarantees are an illusion, which only serves to prevent you from thinking about what can go wrong.

Security Through Obscurity Is Not Always Bad

The term *security through obscurity* is used to describe security measures that rely on secrets in the protocol or algorithm. This is in contrast to public key cryptography systems, which have secret (private) keys, but well-known algorithms. Despite the inherent weaker nature of the security through obscurity approach, if it is layered on top of strong encryption schemes, it can provide an additional deterrent for would-be attackers. In addition, this hybrid approach has an advantage over a strict, strong cryptography scheme since in a traditional scheme the attacker knows that all he needs is the private key. In an obfuscated scheme, however, he may not have any clue what parameters are relevant.

How does this apply to SOAP messages? SOAP is a very expressive protocol. There are in fact an infinite number of variations on how to say the same thing. For example, white space in certain contexts in XML documents is specifically defined as ignorable. Yet there is nothing to prevent you from requiring specific, obscure rules for white space in your SOAP messages. Attackers familiar with SOAP would assume that the ignorable white space is really ignorable, giving you the opportunity to turn the tables and capitalize on the hackers' assumptions. Another example would be to put constraints on the namespace prefixes used, another area where multiple solutions are possible. Outside the SOAP envelope, the HTTP header could also house additional constraints. The obvious side effect is that we have completely undermined the interoperability of SOAP.

"Whenever humans are involved, guarantees are an illusion, which only serves to prevent you from thinking about what can go wrong."

Interoperability Is Not Always Good

As mentioned in the beginning of this article, a major benefit of SOAP compared with other distributed computing technologies is the evidence that the promise of cross-platform and cross-language interoperability is finally being realized. However, security requirements like authentication and authorization can be reformulated as requirements to prevent interoperability with malicious parties. If you already know what implementation your intended clients are using, then being able to interoperate with other clients may be an asset or a liability, or both.

It Is All About Patterns

Another way to look at the security problem is to recognize that the whole process of deploying Web services, as well as the mechanisms of invoking SOAP RPCs and processing SOAP messages, is about manipulating structured data. Within this data are many patterns, some of which promote security, while others undermine security. In this sense, many security problems can be reformulated as a requirement to identify insecure patterns and prohibit them from infecting the system. Patterns in XML are especially important to SOAP because XML permeates the entire architecture. XML may appear in the following:

- Server configuration files.
- Deployment descriptors.
- WSDL.
- In the SOAP envelope.

When a security hole is introduced in any of these components, there is an XML pattern that corresponds to that security hole. For example, your server configuration file could be set to not only expose your service over HTTPS on port 443, but it could expose the same servlet over HTTP on port 80. In some cases, this is desirable, but only if you intentionally make the service available through both a secure and a nonsecure connection. Finding these types of patterns in XML is another technique to ensure security.

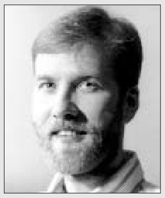
It Is All About Layers

Providing security in multiple layers increases robustness. If one security layer is compromised, the next security layer still provides protection. Layers can also be used to provide flexibility by encrypting some parts of an XML document differently than others. This allows users access to only the portions of the document related to them. Work in the area of encrypting parts of XML documents is under way in the XML Encryption working group of the W3C. A related area is the idea of providing means for signing XML data and verifying signatures, which is covered by the XML Signature working group of the W3C.

Conclusions

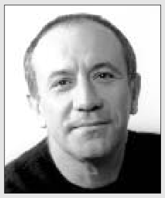
The potential exists for SOAP to allow you to set up very dynamic Web services highly customized to the specific needs of each of your customers. However, this new opportunity comes with the challenge of being able to consistently provide flexibility without compromising your security. Meeting the challenges of security needs requires knowing what the security needs and priorities are, what technologies can be used to achieve them, and above all, thinking clearly about your system's weaknesses.◆

About the Authors



Jim Clune is development manager for ParaSoft Corporation as well as technical lead for the SOAPtest development team. His professional experience includes software engineering, manufacturing engineering, and management. He has a master's of science degree in applied computer science and technology from Azusa Pacific University and a bachelor's of science degree in engineering from Harvey Mudd College.

ParaSoft Corporation
2031 South Myrtle Avenue
Monrovia, CA 91016
Phone: (888) 305-0041
Fax: (626) 305-3036
E-mail: jim.clune@parasoft.com



Adam Kolawa, Ph.D., is CEO and founding member of ParaSoft Corporation. Dr. Kolawa's experience with various software development processes allows him insight into the high-tech industry providing him the ability to successfully identify technology trends. Dr. Kolawa holds seven patents for the technologies behind several innovative commercial software tools. He is co-author of "Bulletproofing Web Applications" and was awarded the Los Angeles Ernst & Young's Entrepreneur of the Year Award in the software category. Dr. Kolawa has a doctorate degree in theoretical physics from the California Institute of Technology.

ParaSoft Corporation
2031 South Myrtle Avenue
Monrovia, CA 91016
Phone: (888) 305-0041
Fax: (626) 305-3036
E-mail: ukola@parasoft.com

WEB SITES

Defense Information Systems Agency

www.disa.mil

The Defense Information Systems Agency (DISA) is a combat support agency responsible for planning, developing, fielding, operating, and supporting command, control, communications, and information systems that serve the needs of the Department of Defense (DoD) and other government offices. DISA is a provider of integrated information solutions to DOD and non-DOD customers.

National Communications System

www.ncs.gov/n5_hp/n5_ia_hp/default.htm

The National Communications System (NCS) Information Assurance Branch was established to focus on the network and information security initiatives of the NCS under a common program branch. This was done to increase NCS efficiency and effectiveness, apply a coordinated direction, and increase the general awareness of the importance of network and information security to the NCS government and industry community. This site is a link to the NCS home page listing programs, publications, member organizations, and more.

Software Testing Institute

www.softwaretestinginstitute.com

The Software Testing Institute (STI) provides access to quality industry publications, research, and online services. STI offers the following professional resources: a software testing discussion forum, the STI Resource guide, the Automated Testing Handbook, the STI Buyer's Guide, and privileged access to STI's exclusive industry surveys, including salary and staffing practices, industry trends and more.

World Wide Web Consortium

www.w3.org

The World Wide Web Consortium (W3C) develops interoperable technologies to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding. On this page, you'll find W3C news as well as links to information about W3C technologies, including Simple Object Access Protocol (SOAP) 1.1, XML Encryption WG, W3C architecture domain, industry surveys on salaries, staffing practices, industry trends, and more.