Securing Simple Object Access Protocol (SOAP) Requests

Bruce Lane
July 15, 2003
Assignment version: GSEC 1.4b

Abstract

In the last few years, Applications have become significantly more complex and increasingly more distributed. Web Services, or SOAP based requests, are emerging as a mechanism to connect the functionality from many different platforms and application architectures into a single service. Unfortunately, there is not a standard way to address security within this model. This paper will present one approach to securing SOAP based requests.

Defense in depth is one of the key attributes of the solution presented. There are many factors involved in an application approach that joins different application architectures on different hosts across different networks. The approach presented explains how security is addressed as each layer of the environment is encountered and how the various approaches are correlated into a comprehensive and manageable solution. It also presents a holistic approach to Web Services by providing correlation between the two key components: the message creator and the message sender. It does this using standards based tools, including SSL/TLS, XML-Signature, and PKI certificates.

Introduction

Application security used to be easy. Applications ran on a single platform under a single application architecture on the internal network so there was only one security model to manage. With the advent of distributed applications, security became much more difficult. Security had to be managed across multiple platforms. The middleware for the distributed application architectures (DCOM, IIOP, DCE, RMI, etc…) had to provide the security mechanism to support integration of the security context. Now we are moving into yet another era of distributed computing, the era of Web Services, or Simple Object Access Protocol (SOAP) based requests. Web Services make it possible to create an application by assembling services from multiple distributed systems. The 'middleware' in this environment is the Hyper Text Transfer Protocol (HTTP). This allows combining services from any platform running in any application architecture that supports SOAP based invocations into a single application context. It is no longer possible to rely solely upon the security features of the platform because the components that make up the application may be hosted on a number of different platforms. Neither is it possible to rely solely upon the application architecture security model because a Web Service may combine components from multiple application architectures (COM+, Java, .Net, EJB, Perl, legacy, etc…). The middleware in this environment (HTTP) doesn't support its own security model other than course-grained support for user authentication. [9] There needs to be a way to tie together the security context of the application across all Web Service invocations, regardless of platform, application architecture, or network. This paper will address the end-to-end security requirements of SOAP based requests by attaching security context to the request being made.

Simple Object Access Protocol

SOAP stands for Simple Object Access Protocol. It is a middleware technology that allows component-based services to be accessed via HTTP (or other transport protocols) using the Extensible Markup Language (XML) as the serialization format. This allows any application that can make an HTTP request the ability to utilize the services that are being exposed. These services are also known as Web Services. The SOAP 1.1 specification is currently "a submission to the World Wide Web Consortium (W3C) to propose the formation of a working group in the area of XML-based protocols." [5] SOAP is described by the W3C as "a lightweight protocol for the exchange of information in a decentralized, distributed environment." [5] In essence, SOAP is a Remote Procedure Call (RPC) mechanism that allows services to be combined together in a loosely coupled manner over a distributed, heterogeneous environment. SOAP uses XML to describe the service invocation and is transmission protocol independent, though the version 1.1 specification explicitly describes the HTTP binding. For the purposes of this document, only the HTTP binding will be considered.

There are two requirements for the HTTP headers of a SOAP request. The media type must be "text/xml". This is specified using the "Content-Type" header.

Example:

```
Content-Type: text/xml; charset="utf-8"
```

The other requirement is that the SOAPAction header must be present. This header is used to indicate the intention of the SOAP request and can be used by firewalls or the destination server to filter SOAP requests.

The SOAP specification is made up of three parts, the Envelope, the encoding rules, and the Remote Procedure Call (RPC) representation. The Envelope, which contains the payload of the request, is made up of two parts, the Header, which is optional, and the Body, which is mandatory. The Body typically represents the application data associated with the request. The serialization format of the data that is exchanged is specified in the encoding rules. The RPC representation does not supply the services normally associated with an RPC mechanism, such as transaction semantics or security considerations. The Envelope Headers are extensible and can be used to supply context information such as transaction context or security credentials. Because the primary goals of SOAP were simplicity and extensibility, [11] security and transaction semantics were not defined and it was assumed that a SOAP implementation would use the underlying transport protocol to supply the additional services needed in the context of remote invocation. The transport, however, only supplies half of the authentication requirements of a SOAP request. It authenticates the sender of the request, but not the creator of the message. Since the transport mechanism and message format are completely decoupled they must be authenticated independently. In addition, once the message creator and message sender have been authenticated, there must be a way to correlate the two authenticated identities. [1]

Problems with SOAP and security

A security model specific to SOAP is not defined. Until consensus on standards for Web Services security is reached and the standards are implemented within each application architecture, each implementation will need to provide its own model or delegate to an external mechanism (HTTP security, application level security, etc…). Without a consistent methodology for securing SOAP calls, different implementations will open up different vulnerabilities.

The WS-I (Web Services Interoperability) group is defining interoperable standards for Web Services. They have published a “Basic Profile” which is a set of non-proprietary specifications to promote interoperability. [14] This specification recommends the use of SSL/TLS, optionally with client side authentication, to provide security for Web Services until a standard security model for SOAP requests is established. They have also established a working group to deliver a set of guidelines for adherence to security protocols that will eventually allow interoperability between Web Services implementations. [15] One of the standards being considered is the WS-Security specification which is being developed by OASIS. [13] WS-Security is part of a larger strategy being proposed by IBM and Microsoft for the purpose of providing security to Web Services. The “Web Services Security Roadmap” specifies a number of protocols that will all work together eventually to enable secure Web Services. [17] Though the Web Services Security Roadmap may eventually become a standard for Web Servcies security, to date only one of the protocols is defined and industry adoption has been slow. The focus of this document is to outline a solution for securing Web Services that can be implemented today based upon current standards.

Web Services use the internet as their delivery mechanism which means they are expected to be accessible through the perimeters of an organization. Current distributed object protocols are prevented from operating through firewalls without opening additional ports. SOAP is designed to work through port 80 or 443 (over HTTP or HTTPS) so that it can pass through firewalls. Bruce Schneier’s comments on SOAP assert that firewalls are intended to prevent distributed application protocols such as DCOM that may come from untrusted sources and the ability to circumvent firewalls is undesirable. [4] However, as we move to a truly global distributed application environment utilizing the public Internet to expose services we must look instead at a holistic approach to verify the authentication and authorization of requests that are allowed through the perimeter.

Authentication Checkpoints

The first item to consider when securing Web Services is the number of hurdles that the request must pass before it is processed. Addressing security at each access layer will help to provide a strong ‘defense-in-depth’ strategy. Within an organization, a Web Service will typically pass through the following layers of access control:

- Network Perimeter
- Operating system controls
- Web Access Control (WAC)
- Application controls

Network Perimeter

The network perimeter is typically made up of firewalls and routers that control what type of traffic can penetrate the outer bounds of the organization's systems based upon predefined rule sets which are derived from the company's security policy. Access is controlled in terms of what kind of traffic can come through the firewalls, what ports that traffic can come through, what IP address the traffic came from or is destined for, and, in some cases, the format of the data based upon the expected protocol for the port (i.e., HTTP traffic is expected on port 80, so the traffic on port 80 can be inspected to make sure it conforms to the specified format for HTTP).

Platform Security

The operating system where the code will run is the next layer of control. The identity that the request runs as must have permission to run the code on that system. If the application needs access to any resources on the system (files, registry keys, named pipes, etc…), then the identity of the request must be authorized to access them as well.

Web Access Control

The Web Access Control (WAC) can be services built into the web server or it can be an add-on privilege management product. Either way, the WAC is an abstraction at the web server interface that provides a 'yes' or 'no' decision based upon the credentials supplied when a URL is requested. Most web servers provide basic authentication, digest authentication, and certificate-based authentication (the latter under Transport Layer Security, which is the successor to Secure Socket Layer, referenced as SSL/TLS for the remainder of this document) in their base implementation. Microsoft's servers also provide native Windows authentication (NTLM or Kerberos, depending upon the environment and configuration). Privilege management products usually hook into the web server to supply the same functionality plus additional features, including forms based authentication, which is authentication based upon HTML forms and usually managed through session cookies. Forms based authentication would normally be handled at the application level, but since the privilege management products hook into the web server services it is provided at the infrastructure level. The WAC will ultimately authenticate a user based upon credentials provided. If the user passes the test for authentication then a second test will be made to see if the authenticated user is authorized for the resource they are requesting, in this case a URL. If the user passes the authorization test as well then the web server will allow the user to have the resource they have requested. Depending upon the environment and configuration, the web server will either access the requested resource for the user using the user's credentials and acting as the requesting user, or it will access the resource on behalf of the user with a predefined identity by using a set of credentials owned by the server (aka, "service identity").

Application Security

The last hurdle a Web Service must pass is the application security. Most application environments have a security model built into the application architecture. When a request is made to the application it must be made based upon a specific user's identity. This prevents the code from being run indiscriminately by any arbitrary user. A critical decision for Web Services is whether the actual credentials of the user will be passed to the application so that the application runs in the context of the requesting user, or if the Web Service will run the application with a set of predefined credentials. There are arguments for each approach. Using the end user's credentials simplifies auditing but adds a great deal of administration overhead since the application must be configured for each possible user. Using a service set of credentials improves performance by limiting the context in which the application must run, allowing for the possibility of resource pooling and reducing administration, but it may be harder to track on whose behalf the application was run.

Corrrelation

In addition to passing all layers of control, there must be some way to correlate the verification at each layer so there is one holistic security model that can be administered in a centralized manner, rather than trying to manage and administer the authentication and authorization requirements of the request at each layer of access.

XML-Signature

The next portion of this document will outline the steps involved in authenticating a SOAP call at the various points of interception during it's path through a distributed system. However, XML-Signature is a critical part of the following solution so the next few paragraphs will briefly describe the details of XML-Signature.

XML-Signature is a Recommendation by the W3C (World Wide Web Consortium) and the IETF (Internet Engineering Task Force) that specifies how an XML document can be signed with a digital signature. [3] The specification allows for the signing of both XML and non-XML content. Basically, almost any resource that is URI addressable can be signed using XML-Signature. This applies regardless of whether the object is local or remote, text or binary. The protocol also supports multiple signatures on a single document and the ability to sign only a portion of an XML based document. XML-Signature was designed to be an application independent standard in order to provide interoperability between applications as well as platforms. One of the problems that XML Signature solves is the ability to treat portions of a document independently with regards to digitally signing. For example, an online order may need to have the ordering information signed to verify that the order has been placed, but other information may be dynamically changed, like preferences on how to receive other communications. Another scenario may exist in which different parts of a document need to be signed by different entities, as in the case of some workflow applications.

When creating a cryptographic hash from a document, any change to any bytes in the document will create a different hash. This is to confirm the integrity of a digital document. However, this creates some problems when working with XML. Two XML documents that may have some differences when compared on a byte-by-byte basis may still be logically equivalent. This is true because of issues of white space, comments, line delimiters, empty tags, and different ways of representing the same format. [12] The Canonical XML specification was created to address these issues so that reducing a document to its canonical form allows it to be compared with other documents that are logically equivalent but may not be physically equal. Therefore, when creating and validating XML signed documents, if the message digests are computed on the canonical version of the document, then the digests will be the same if the documents are logically equivalent, even if the text of the original XML documents has the variations noted above.

XML-Signature Representation

The following XML representation taken directly from “XML-Signature Syntax and Processing” by Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon [16] will be a useful reference during the following discussion of the makeup of an XML-Signature document:

```
<Signature ID?>
   <SignedInfo>
     <CanonicalizationMethod/>
     <SignatureMethod/>
     (<Reference URI? >
       (<Transforms>)?
       <DigestMethod>
       <DigestValue>
     </Reference>)+
   </SignedInfo>
   <SignatureValue>
  (<KeyInfo>)?
  (<Object ID?>)*
 </Signature>
```

Note: “?” denotes 0 or 1 occurrence, “+” denotes 1 or more occurrences, “*” denotes 0 or more occurrences. [16]

XML digital signatures are represented by an XML document hierarchy with the element named ‘Signature’ at the base. Under the <Signature> element is the <SignedInfo> element, which contains information about the item being signed including the canonicalization method used (<CanonicalizaitonMethod>), the method used to create the signature (<SignatureMethod>), and the reference element (<Reference>). The <Reference> element gives specifics about the signed item including any transformations that were applied (<Transforms>), the method used to create the digest value (<DigestMethod>), and the digest itself (<DigestValue>). The next element under the <Signature> element is the <SignatureValue> element, which is the actual signature that was generated when the data was signed with the user’s certificate. The next sibling to <SignatureValue> is the optional <KeyInfo> element, which provides information about the key that should be used to validate the signature.

The steps to generate an XML signature include creating the <Reference> elements and the <SignatureValue> over the <SignedInfo> element. The first step to create the <Reference> element is to apply any transforms as required by the application. Next, the digest over the result must be calculated. Now a <Reference> element can be created which includes the <DigestMethod> and <DigestValue> along with other optional elements. The next step is to generate the signature. First, a <SignedInfo> element is created with the signature method, canonicalization method, and the references. Next the canonicalization is applied to the <SignedInfo> element and the <SignatureValue> is calculated. From that information the <Signature> element can be constructed.

There are two steps to validating a signed XML document. The first step is to validate the <Reference> and attached digest; the second is to validate the signature. To validate the <Reference>, first the <SignedInfo> element must be canonicalized using the canonicalization method specified. Next, the data that was digested must be obtained and the digest specified in the <Reference> specification must be applied. The resulting digest must be compared to the <DigestValue> in the Reference element under <SignedInfo> to make sure they are the same. If they don't match then the validation has failed. If they match then the next step is to validate the signature. First, the validation key information is obtained from the <KeyInfo> element or an external source. Next, the canonicalization specified by the <CanonicalizationMethod> is applied to the <SignedInfo> element. Finally, the <SignatureValue> is validated using the signature method specified in the <SignedInfo> element.

Authentication of SOAP Requests

With an understanding of XML-Signature in place, the following steps outline a solution to authenticate SOAP based requests in a distributed system.

Network Perimeter

In the context of Web Services, it is impractical to authorize the originator of the request at each point of authorization, i.e., the perimeter, the WAC, all applications, and all platforms. The performance of the request would suffer and administration of the services would be unmanageable. Instead, the request should be verified at the perimeter as an allowed service based upon the standard firewall controls. These include verifying the protocol (TCP vs. UDP) and verifying it is an allowed port (80 or 443 for Web Services). In addition, the content can be checked for correct conformance to the HTTP specification. Web Services can be identified by a Content-Type of text/xml. When a Web Service is identified, the SOAPAction header should be checked against a known set of allowed Web Services. When using Secure Socket Layer (SSL/TLS), this will need to be done at the receiving server, since the firewall will not be able to read the encrypted data. Depending upon the security needs, it may make sense to verify the requesting IP address as an additional course grained authorization check. The authentication of the credentials of the requester doesn't occur until the request reaches the WAC. At that point, the identity of the requester is authenticated based upon the credentials supplied.

Web Access Control

At the WAC, the requesting entity must be authenticated to the session. HTTP supplies a number of methods for authenticating a user before satisfying the request. Most web servers support at least basic authentication, digest authentication, and certificate-based authentication, the last option requiring SSL/TLS. Microsoft's Internet Information

Server (IIS) supports integrated Windows authentication (NTLM or Kerberos, depending upon environmental factors) as well. In addition, forms-based authentication can be used at the application level or by third party WAC add-on products. Though there are several methods of authentication possible at the WAC, the best approach for Web Services is certificate-based authentication with SSL/TLS. This can be accomplished through SSL/TLS by using mutual authentication of the SSL/TLS certificate. Normally in an SSL/TLS session only the server is authenticated, but SSL/TLS also has the option of requesting a certificate from the client during protocol negotiation and using that certificate to verify the authentication of the client. Using certificate-based authentication under SSL/TLS provides a stronger case for non-repudiation once the user is authenticated. This is because the user is authenticating with a private certificate that (presumably) is only accessible by them. During the SSL/TLS handshake, the client tells the server it would like to use SSL/TLS for the communications channel. The server then sends its certificate including its public key to the client. The client verifies the certificate (assuming it already has a trust relationship with the root certificate authority). It then creates a session key, encrypts the session key with the server's public key and sends it back to the server. If the server has specified that it supports or requires client authentication then the client also sends the server a copy of its certificate. The server verifies the client's certificate and, assuming it has a trust relationship with the root certificate authority of the client's certificate, the two can now communicate with a high level of assurance that each is who they say they are. [8]

SSL/TLS also provides confidentiality to the message during transport by encrypting the data that is being exchanged. One other feature provided by SSL/TLS is the use of Message Authentication Codes (MACs). The MAC associated with a message will verify that the message is complete and that the integrity is intact, i.e., the message received has not been altered. So, the advantages of using SSL/TLS with client authentication are threefold:

- The message is encrypted for confidentiality during transport
- The message is verified using MACs during transport for integrity
- Both ends of the session can be verified using strong, certificate-based authentication

One consideration when using SSL/TLS is that, since the data is encrypted end-to-end, it is not possible to do verification of the SOAP headers to the SOAP payload/message at the firewall. In this case the SOAP HTTP headers (SOAPAction) should be verified against the actual SOAP message when it is received at the server.

If certificate-based authentication under SSL/TLS is not possible then basic authentication over SSL/TLS is a good next choice. In the case of basic authenticate, SSL/TLS must be used because the user id and password are passed in the HTTP headers in the clear for basic authentication. [9] Most SOAP or Web Services clients support HTTP basic authentication in their implementation. Forms based authentication is another possible alternative but be aware that this form of authentication will usually require the client application to manage a set of authentication cookies and may require an additional log-on step.

Message Authentication

The next step is to authenticate the message creator. This is accomplished by using XML-Signature to sign the payload of the SOAP request. XML-Signature will allow the sender to authenticate the message by signing it with their private key. On the receiving end, the receiver of the message can verify the identity of the sender of the message by verifying the digital signature with the sender's public key. See the previous discussion of XML-Signature for details of how this works.

Once the sender of the message is authenticated and the creator of the message is authenticated the message authentication and sender/receiver authentication need to be correlated. [1] This is most easily accomplished by using the same certificate for SSL/TLS client authentication that is used for signing and subsequently verifying the SOAP payload. Assuming that the message channel was authenticated using a client certificate under SSL/TLS, the certificate can be extracted during the SSL/TLS verification and then used to verify the SOAP payload that has been signed via XML-Signature. If another method of authentication is used for the channel (basic authentication, forms-based authentication, etc…) then the receiver will need to look up the public certificate of the sender based upon the credentials provided during channel authentication. This method of verification is not as strong as having the user authenticate directly with the signing certificate and will not supply the same level of non-repudiation as the previous method.

When using SSL/TLS and XML-Signature together to authenticate the message sender and message creator some other precautions should be taken to subvert a malicious sender or recipient. A unique nonce may be added in the SOAP payload to all requests to make them distinct and resistant to replay attacks – possibly with a verifiable timestamp. The receiver of the messages will need to maintain some form of history of each nonce as it is used so that they can be checked against any type of replay attack. In addition, the intended recipient may be included in the SOAP payload of all requests. The receiver of the message may then verify that they are indeed the intended recipient of the message, again avoiding a replay attack. [1]

Platform and Application Security

Having established strong, non-reputable, certificate-based verification of the message transport and the message content, the transaction flow can proceed with a high level of assurance regarding the authenticity of the requester. This makes it possible for the SOAP request to invoke its services to run in the context of an identity specified by the server that has only enough privileges for the specific request, including access to any host resources that are needed in the course of the transaction. This allows for the application of the 'principle of least privilege' and aids in the centralization and reduction of administration.

Conclusion

In summary, securing applications in the Web Services space presents challenges that are unique in regards to integration of heterogeneous platforms, application security models, and networks. This paper has outlined an approach to provide strong certificate-based authentication at the transport layer, and to provide authentication of the message creator directly within the message itself, which can then provide a high level of assurance for subsequent layers of execution to trust the request.

References

[1] Satoshi Hada, “Soap Security Extensions: Digital Signature”, August 2001, http://www-106.ibm.com/developerworks/webservices/library/ws-soapsec/

[2] Murdoch Mactaggart, “Enabling XML Security”, September 2001, http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html

[3] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, Ed Simon, “XML-Signature Syntax and Processing”, February 2002, http://www.w3.org/TR/xmldsig-core/

[4] Bruce Schneier, “Crypto-Gram Newsletter”, June 2000, http://www.counterpane.com/crypto-gram-0006.html#SOAP

[5] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, “Simple Object Access Protocol (SOAP) 1.1”, May 2000, http://www.w3.org/TR/SOAP/

[6] Aaron Skonnard, “Understanding SOAP”, March 2003, http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us//dnsoap/html/understandsoap.asp

[7] Martin Gudgin, Timothy Ewald, “The Slippery Soap”, October 2001, http://www.xml.com/pub/a/2001/10/17/slippery-soap.html

[8] T. Dierks, C. Allen, “The TLS Protocol”, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, January 1997, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2068.html#sec-15

[10] Allen Brown, Barbara Fox, Satoshi Hada, Brian LaMacchia, Hiroshi Maruyama, “SOAP Security Extensions: Digital Signature”, February 2001, http://www.w3c.org/TR/SOAP-dsig/

[11] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, “Simple Object Access Protocol (SOAP) 1.1 – Design Goals”, May 2000, http://www.w3.org/TR/SOAP/#_Toc478383487

[12] J. Boyer, “Canonical XML Version 1.0”, March 2001, http://www.ietf.org/rfc/rfc3076.txt

[13] Bob Atkinson, Giovanni Della-Libera, Satoshi Hada, Maryann Hondo, Phillip Hallam-Baker, Johannes Klein, Brian LaMacchia, Paul Leach, John Manferdelli, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, John Shewchuk, Dan Simon, “Web Services Security (WS-Security)”, April 2002,

http://www-106.ibm.com/developerworks/library/ws-secure/

[14] Keith Ballinger, David Ehnebuske, Martin Gudgin, Mark Nottingham, Prasad Yendluri, “Basic Profile Version 1.0”, May 2003, http://www.ws-i.org/Profiles/Basic/2003-05/BasicProfile-1.0-WGAD.htm#security

[15] Martin LaMonica, “Group eyes Web services security”, April 2003, http://news.com.com/2100-1012-994938.html

[16] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, Ed Simon, “XML-Signature Syntax and Processing – 2.0 Signature Overview and Examples”, February 2002, http://www.w3.org/TR/xmldsig-core/#sec-Overview

[17] Authored jointly by IBM and Microsoft – key contributors: Giovanni Della-Libera, Microsoft; Brendan Dixon, Microsoft; Joel Farrell, IBM; Praerit Garg, Microsoft; Maryann Hondo, IBM; Chris Kaler, Microsoft; Butler Lampson, Microsoft; Kelvin Lawrence, IBM; Andrew Layman, Microsoft; Paul Leach, Microsoft; John Manferdelli, Microsoft; Hiroshi Maruyama, IBM; Anthony Nadalin, IBM; Nataraj Nagaratnam, IBM; Rick Rashid, Microsoft; John Shewchuk, Microsoft; Dan Simon, Microsoft; Ajamu Wesley, IBM, “Security in a Web Services World: A Proposed Architecture and Roadmap”, April 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/

Other sources

[18] James Kobielus, “XML Digital Signatures”, The Burton Group: Network Strategy Report October 2000.