Web Service Authentication Methods

Brandon Russel Faust, Liza Gonzales, Shengyu Jin, and Jason Paul Stajduhar
July 17, 2006

Abstract

Web services are an integral part of many Service Oriented Architectures. They facilitate inter- and intra-system communication, allowing for data exchange and shared functionality among internal and external consumers. However, while a system can open access to its resources, security must be considered: Who has access to this service and how can a consumer prove his identity? How can a server prove the same? This paper aims to analyze several methodologies of authentication in web services and provides suggestions for enhancements.

Introduction

"Web service" is a term that has become more prevalent in discussions of Service Oriented Architectures and network communication between software applications, in general. They can operate over numerous protocols such as HTTP, FTP, and SMTP. Exchanged messages can simple (such as raw text) or can be formatted in more structured XML formats. Communicants need only know the format of these messages, not the environments of the systems with whom they communicate. Thus, "interoperability [across] … disparate platforms" becomes a simple process rather than an impossible challenge. ("Web service")

Through web services, software systems share resources with other systems or subsystems. For instance, they might share data or metadata; they might also share a function in a way similar to that of developers sharing functions in object-oriented classes. However, paradoxically, this inherent openness of web services must be tempered with the practical concern of security. Unless the objective of a particular web service is to provide a public resource for all requestors, it must limit access to those whom it (or some entity acting on its behalf) has authorized.

With this challenge of authorization comes that of authentication. How does a web service determine with whom it is communicating, and further, how can it trust that this identity is genuine? The rising use of web services has prompted the birth of several methodologies for authentication. Among these are Extensible Markup Language (XML) Digital Signatures, the XML Key Management Specification (XKMS), and the Security Assertions Markup Language (SAML). The remainder of this paper introduces these protocols, as well as analyzes their strengths and weaknesses. Where problems are found, attempts are made for possible improvements.

XML Digital Signatures

Background

XML digital signatures play an important role in Web Service security. XML digital signature is the specific XML syntax used to represent a digital signature (PKI cryptography) over any arbitrary digital content ("XML Signature"). Due to the distributed and heterogeneous nature of Web Service, it is very important to validate a user's identity in certain transactions. XML signatures (a joint effort of the W3C and IETF) provide message level authentication in web services. The signing of XML documents takes place at the application layer. This means that the signature does not get dropped when the transport layer is left, like it is with SSL/TLS after a TCP session ends.

XML Signature defines the processing rules and syntax to wrap message integrity, message authentication, and user authentication data inside an XML format. An XML signature is itself an XML document. It is a compound of four objects, two of which are mandatory: <SignedInfo> and <SignatureValue>. The <SignedInfo> sub-document specifies the data to be signed. <SignatureValue> yields the digital signature itself. There are also two optional elements, <KeyInfo> and <Object>. The <KeyInfo> element contains information about the key used to sign the document. <Object> elements carry any other information needed to support the signature. The following is the basic XML digital signature generation and verification process.

Reference Generation

For each resource (all of part of XML file, HTML, or JPG) being signed,

1. Apply transforms.
2. Calculate the digest value of the transformed resource using <DigestMethod>, and place the resulting value in <Digest Value>.

Signature Generation

3. Apply the <CanonicalizationMethod> and <SignatureMethod> to the <SignedInfo> element. The <CanonicalizationMethod> element indicates the algorithm used to canonize the <SignedInfo> element. The <SignatureMethod> identifies the algorithm used to produce the signature value.
4. Put the result from step 3 into the <SignatureValue> element. It contains the base-64 encoded value of the digital signature.

To verify the signature of the <SignedInfo> element, recalculate the digest of the <SignedInfo> element (using the digest algorithm specified in the <SignatureMehod> element>, then use the public verification key to decrypt the value of the <SignatureValue> element. If two results match, then the signing person is who he claims to be.

```
<Signature>
    <SignedInfo>
        <SignatureMethod>
        <CanonicalizationMethod>
        <Reference>URI
            <Transforms>
            <DigestMethod>
            <DigestValue>
          </Reference>
    </SignedInfo>
    <SignatureValue>

    <KeyInfo>   <optional element>
    <Object>    <optional element>
</Signature>
```

Figure 1. XML Template Using XML Digital Signature (Simon)

Enhancements

As mentioned earlier, one of the signature components, KeyInfo, is an optional element. For the most part, this flexibility is a positive feature. However, leaving information out of the signature means that there can be problems with signature verification at a later time or if the verifier changes. Although XKMS addresses the distribution and registration of public keys and ensures all the keys up-to-date, including signature parameters in the signature element is the preferable solution. While implementation-specific S8in the s)5(i)-1(e sgnaturs)]TJ0.00

along the tree to identify nodes of interests using recursion. However, it is possible that recursion will cause stack overflows. This can bring down a B2C or B2B server in a denial of service attack. Future research should be done to address this problem (Weissmann).

XKMS

Background

When performing web service authentication, dealing with cryptographic keys is essential. Public keys must be registered somewhere so anyone can access them, and those keys must be recovered and validated so they can be used for decrypting digital signatures. An infrastructure that works with SSL to provide key management that had to be addressed for true advancement of web service security was PKI. PKI, which stands for Public Key Infrastructure, allows users to securely exchange information over an unsecured channel through the use of public and private cryptographic keys. These keys are acquired from a trusted third party known as a Certificate Authority (CA). The CA issues certificates that are utilized to authenticate users over the Internet. Thus, PKI is a vital part of authenticating people or systems when dealing with web services, especially in the realm of e-commerce, where PKI can be used to exchange money.

However, there are problems when applying PKI to web services. PKI operations used for authentication, such as public key validation, registration, recovery, and revocation are very resource intensive. Furthermore, since the client has to perform these functions, some applications, particularly small ones embedded on mobile devices, cannot utilize PKI. And since cryptographic key processing is necessary for web service authentication, only using PKI would exclude all of these applications. Also, when performing this authentication with SSL and PKI, there is a "difficulty in managing client certificates" (Loftus). This problem arises again because of resource consumption, as well as the fact that there are many different implementations of PKI. XKMS aims to solve these problems by creating a single key management standard that moves the processing of the PKI operations to a trusted remote server (Shin). Transferring the PKI operations to the trust server almost eliminates the resource consumption problem on the client and also makes the client code much simpler with a greater degree of interoperability.

XKMS, which stands for XML Key Management Specification, is based on a dedicated XKMS server performing trust services (the aforementioned PKI operations) for clients. These trust services are accomplished through web services running on the XKMS server, and are invoked by SOAP messages sent to the server from the XKMS client (Shin). XKMS is comprised of two pieces: XKISS, the XML Key Information Service Specification, and XKRSS, the XML Key Registration Service Specification. XKISS is the protocol through which an application passes "the processing of key information associated with an XML Signature" over to a service (Hallam-Baker and Mysore, Specification). This processing includes the two major operations of locating a key on the server and performing validations. Keys must be located so they can be used to encrypt data or verify signatures. Validation is done to prove that a received key is a legitimate one as well as grant rights to clients with certain privileges (Gralla). XKRSS is the protocol through which a client registers a key pair with an XKMS server for future use with XKISS. These protocols enable

the remote processing of PKI operations through KXMS. Also, they are designed to be compatible with all underlying PKIs.

The current status of XKMS is that on June 28, 2005, the World Wide Web Consortium approved XKMS 2.0 as a W3C Recommendation. This version was generated by the W3C XML Key Management Working Group. Then, in December 2005, XKMS 2.0, as well as its Bindings and Requirements were officially completed (W3C News Archive). According to the WC3 documentation, XKMS 2.0 "[s]pecifies protocols for distributing and registering public keys, suitable for use in conjunction with the W3C Recommendations for XML Signature and XML Encryption." (Hallam-Baker and Mysore, Specification). The Bindings describe the necessary security requirements for using the XKMS protocol (Hallam-Baker and Mysore, Bindings), and the Requirements provide "design principles, scope and requirements" for XKMS specifications and protocol implementations (Hallam-Baker and Mysore, Requirements). With the publication of all of these documents, the W3C XML Key Management Working Group was closed, and currently there are no intentions for it to reconvene (W3C XKMS Activity).

Many products that provide security for web services are currently utilizing XKMS. Two examples are the Phaos XKMS and the SQLData XKMS 2.0 Client. Phaos XKMS implements XKMS based on the standard created by W3C. It is a toolkit for performing the cryptographic key operations associated with XKMS in order to authenticate "user identities across businesses and applications" (Phaos). This functionality addresses the authentication limitations SSL has with B2B web services. It also aims to help developers create code for performing these operations that is portable and simple. SQLData XKMS 2.0 Client also employs XKMS according to the WC3 specification. With SQLData, the "XKMS client is implemented in C/C++ as a COM object," and the key functions can be accomplished simply through method calls (SQLData). These products show how XKMS can currently be used in real-world applications for web service authentication in the form of XML signature verification.

Enhancements

XKMS addresses many of the problems that plague PKI usage with web services. It provides a single standard to use for PKI operations, while removing much of processing from the clients. However, we have identified several areas where improvements could be made.

One of the biggest advantages of using XKMS is that client resources are not used up to perform key management operations. However, even when using XKMS, the client is still responsible for major cryptographic functions (Udell). A possible extension that could alleviate this issue further would be to enable XKMS servers to perform the decryption of signed information on the document being transferred. This action could happen in multiple ways. One approach could have the client sign the document and send it to the web service. Then, instead of having the web service recover and validate the client's public key using the XKMS server, the web service could pass the document (or simply the signed material) to the XKMS server using extensions to the XKMS language. Then the XKMS server could perform the decryption of any signed material and send the document back to the web service along with a message expressing the results of the authentication. Another method would be

to have the client send the signed document directly to the XKMS server and include the destination address of the web service that should receive the document. Then the XKMS server could perform the decryption and forward the document to the web service along with results of the authentication. The XKMS server could also potentially handle document signing for the client in cases where it stores a copy of the client's private key. This functionality would further the XKMS benefits of decreasing resource consumption on the clients and allowing for simpler client code. However, these operations would greatly extend the scope of XKMS, and require more passing of the documents. Further investigation would have to be done to find the performance increase these services could provide, and weigh them against the difficulties in implementing them.

The following scenario brings to light a possible problem with XKMS. A client could sign a document with a private key, where the corresponding public key is registered with an XKMS server, and send the document to a web service. Then, before the web service successfully validates the client's public key through the XKMS server in order to perform the authentication, the client could revoke its public key that is stored on the XKMS server. This situation would result in a false negative for the authentication, which could cause serious problems for the client. The chances of this situation happening are small, but it is possible. The message from the client to the web service could be delayed in transport or it may have to wait to be processed if the web service is busy when the document arrives. In either case, the time that the client could revoke its public key before authentication is performed would increase. Also, clients that deal with very sensitive data may need to change their public keys frequently. These events would increase the probability of this situation occurring. To decrease the likelihood of this series of events, the XKMS servers could be required to continue to store all the information for public keys that have been revoked. Then, if a web service attempts to recover a public key but it is not found in the active list, the revoked list could be checked as well. This solution poses a new problem though. With many previous public keys being stored for a client, there could be confusion as to which one the web service is attempting to retrieve, and a client may be authenticated when it should not be based on an invalid key. To address this trouble, timestamps could be used. The client would add a timestamp to the document it signed and sent to the web service, as well as to the revocation request it sends to the XKMS server. When the XKMS server revokes the public key and moves the data to the revoked list, it would also store the time this operation took place. Then when the web service sends the validation request to the XKMS server it would also send the timestamp of when the document was signed. The XKMS server would only successfully validate a key from the revoked list if the timestamp from the document signature was earlier than the revocation timestamp. Finally, storing all the revoked keys would pose a memory problem, so the XKMS server should only store the revoked key information for a set limit of time before deleting the data completely.

Another potential issue with XKMS is that according to its specification, "all XKMS requests MAY cause a change of state" (Hallam-Baker and Mysore, Specification). This statement means that if a request is repeated, it could result in negative consequences. For example, a user may send a key registration request to an XKMS server, then due to problems with the response message, it may not know the server successfully received the request. Then the client might resend the request, but when the server receives the duplicate, the

course of action and end state are unknown. XKMS should provide a way for the server to quietly disregard the second request, and make that course of action mandatory. Similar actions should be defined for duplicates of all other messages as well.

With that said, however, SAML can certainly bring on the benefits for many aspects. First, SAML can rid of the limitations related to browser cookies. A user's web browser will store cookies locally to preserve a state specific to that user. These cookies can only be shared among that domain, while SAML aims to transfer these authenticating properties across DNS domains.

Another benefit to consider is using SAML as an enabler for Single-Sign On (SSO). In the same manner this XML security technology improves the concept of cookies, SAML can sustain a user's authentication/authorization throughout several domains – a termed coined as Cross-Domain Single-Sign On (CDSSO). So, rather than resorting to proprietary methods of sustaining a user's status on an organization's intranet, SAML could assist in a seamless manner.

A third indication for improvement with utilization of SAML is its use with Web Services in general, as stated previously. Much like the overall strength of SAML, the standard will provide "the means by which authentication and authorization assertions can [be] exchanged between communicating parties" (CoverPages).

Enhancements

Though there is not much to be faulted towards SAML, this XML protocol could use some advancement to prolong its usefulness in web service authentication. For instance, SAML depends on SSL certificates to offer up digital signatures and encryption of its assertions. A wish (or suggestion) for improvement would be that SAML could become independent of the pains of SSL, but the "fine-grained security" that these two can create together seems unmistakably meant to be.

As long as SAML depends on SSL and other means of security on the transport layer, one could claim a flaw that would potentially defeat the purpose of SAML authentication within a web service. What if an anti-user injected a so-called *fake* SAML assertion request to gain access to certain web service privileges? Bob's archenemy could be ordering CDs with Bob's credentials - but without his consent. It would be safe to exclaim that if an encrypted, digitally-signed request was transmitted, a continuous conversation would be needed to maintain the integrity between the two partners attempting to authenticate the user. TSL or SSL 2.0 would be very pertinent in securing the process of transmitting SAML assertion requests and responses (Hirsch).

Another possible issue with SAML is the assertion requests corrupting at the point of authentication. If, somehow, there was data loss or nonconsensual modification to the request by a hacker, the authenticating party could either send back an invalid assertion response or would not respond at all. If this problem were consistent, authentication would apparently be rendered useless and the user would get nowhere. It would be recommended a sort of reliable data transfer would be in place to decrease the likelihood of assertion corruption or interruption.

Surely, the purpose behind SAML accomplishes plenty of authentication issues, especially for the benefit of Web Services. Its ability to improve the timeliness and integrity among one user across several servers is desirable. Provided that the transmission of SAML's assertion requests and responses are intact and secure, SAML shall flourish.

Summary

The need for security in web services has given rise to innovative authentication methods including XML digital signatures, XKMS, and SAML. Respectively, these allow signing of components of XML documents, authentication by mobile devices, and authentication over multiple servers. However, while these protocols solve one set of problems, another set becomes apparent as they become more widely used.

For XML digital signatures, a slight alteration of the specification (requiring a now-optional element) may reduce uncertainty in identity verification. The adoption of a common simple key exchange infrastructure and its implementation among end-users may provide complete (two-way) authentication. An alternative to XPath may reduce the likelihood of server attacks.

In XKMS, moving computationally intensive cryptographic operations from client to server may make this protocol more efficient. Better error handling with regard to lost messages between server and client will eliminate false duplicate requests for keys.

SAML may benefit from an uncoupling from SSL. Guarding against forged requests and corresponding invalid responses may become necessary.

In conclusion, a wall of security has been built to protect web services against unauthorized use. A key foundation of this wall is the stability of current authentication methodologies. However, while these are able to handle the current implementation of web services, enhancements may be necessary to make them more efficient or secure in future implementations. In this way, web applications may be better able to control with whom it shares its resources.

Works Cited

Cohen, Frank. "Debunking SAML myths and misunderstandings." IBM developerworks. http://www-128.ibm.com/developerworks/xml/library/x-samlmyth.html. July 8, 2003.

Hirsch, Frederick, Rob Philpott, and Eve Maler. "Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) 2.0". OASIS Open 2005. http://xml.coverpages.org/SAML-sec-consider-20-os.pdf. March 15, 2005.

Gralla, Preston. "How XKMS Works". SearchWebServices.com. http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci822709,00.html. May 14, 2002.

Hallam-Baker, Phillip. Mysore, Shivram. "XML Key Management Specification (XKMS 2.0)". W3C. http://www.w3.org/TR/xkms2. June 28, 2005.

Hallam-Baker, Phillip. Mysore, Shivram. "XML Key Management Specification (XKMS 2.0) Bindings". W3C. http://www.w3.org/TR/2005/REC-xkms2-bindings-20050628. June 28, 2005.

Hallam-Baker, Phillip. Mysore, Shivram. "XML Key Management Specification (XKMS 2.0) Requirements". W3C. http://www.w3.org/TR/2003/NOTE-xkms2-req-20030505. May 5, 2003.

Loftus, Jack. "The pros and cons of securing Web services with SSL". SearchWebServices.com. http://searchwebservices.techtarget.com/qna/0,289202,sid26_gci995388,00.html. July 26, 2004.

Phaos. "Cryptography and XML Products". http://www.w3.org/2001/XKMS/Activity. 2004.

Polivy, Daniel J. and Roberto Tamassia. "Authenticating Distributed Data Using Web Services and XML Signatures". ACM Workshop on XML Security. November 22, 2002. Fairfax, VA

"SAML". Wikipedia. http://en.wikipedia.org/wiki/SAML. July 1, 2006.

"Security Assertion Markup Language (SAML)". CoverPages. http://xml.coverpages.org/saml.html. June 20, 2006.

Shin, Sang. “Secure Web Services”. Java World. http://www.javaworld.com/javaworld/jw-03-2003/jw-0321-wssecurity.html. March 18, 2003.

Simon, Ed, Paul Madsen, and Carlisle Adams, “An Introduction to XML Digital Signature”. http://www.xml.com/pub/a/2001/08/08/xmldsig.html

SQLData Systems, Inc. “SQLData XKMS 2.0 Client”. http://www.sqldata.com/xkmsclient.htm. November 29, 2005.

Udell, Jon. “XML-Style PKI”. InfoWorld. http://www.infoworld.com/articles/pl/xml/02/09/16/020916plspecs.html. September 13, 2002.

W3C. W3C Home Page News Archive. http://www.w3.org/News/2006. January 17, 2006.

W3C. XML Key Management (XKMS) Activity Statement. http://www.w3.org/2001/XKMS/Activity. April 14, 2006.

“Web service”. Wikipedia. http://en.wikipedia.org/wiki/Web_service. July 15, 2006.

Weissmann, Markus. “Review of XML Signatures”. http://www.mweissmann.de/downloads/XML-Signatures-Paper.pdf

“XML Signature”. Wikipedia. http://en.wikipedia.org/wiki/XML_Signature. July 15, 2006.