Global Information Assurance Certification Paper

GIAC Security Essentials Certification (GSEC) Practical Assignment

version 1.4b Option 1

Submitted by Richard Rabinowitz

Title: Web services - why all the talk about security?

Abstract

Web Services security has been a popular topic of discussion in the IT industry recently. Many of those in the information security field have more of an infrastructure background than an application development background. As a result, much of the discussion about XML, Web Services, and related security issues may be hard to grasp without the necessary basic foundation of knowledge about this emerging technology. This research paper introduces XML, SOAP, and Web Services, discusses security issues, and reviews important established and emerging standards for Web Services security. The intended audience are IT professionals who may not be well versed in XML, Web Services, and SOAP. The references provided will help interested professionals delve into the topic further.

Introduction

You can't open a computer magazine these days without finding an article related to Web Services security. Many IT professional careers have moved along an infrastructure/security track rather than an application development track. As a result, many IT professionals interested in security do not have a background in XML, SOAP, and Web Services. The focus of article is to give these professionals a basic level of knowledge of Web Services, related security issues, and emerging security standards. The references and sources cited in the article will point them to further sources so that they may work intelligently with application developers to securely build and implement Web Services.

What are Web services?

Despite all the buzz about them, many security professionals are not clear on exactly what Web Services are. We will define and describe Web Services to begin the discussion. The best and most succinct definition that I've seen is posted on PerfectXML.com (http://www.perfectxml.com/WebSvc1.asp) :

> A Web Service is programmable application logic accessible using standard Internet protocols. Web Services combine the best aspects of component-based development and the Web. Like components, Web Services represent black-box functionality that can be reused without worrying about how the service is implemented. Unlike current component technologies, Web Services are not accessed via object-model-specific protocols, such as DCOM, RMI, or IIOP. Instead, Web Services are accessed via ubiquitous Web protocols (ex: HTTP) and data formats (ex: XML).[1]

Basically, Web Services help to make components available to users, systems, or other objects via Web protocols without any requirements to use object-model protocols such as those defined in COM+, CORBA, J2EE etc. This allows components to be accessed by systems using common, standard, platform-independent, and well-known methods such as Web servers, browsers, and other XML clients.

OK, so what's this SOAP I've been hearing so much about?

Whatis.com (http://www.whatis.com) gives the following definition of SOAP:

> Simple Object Access Protocol (SOAP) is a way for a program running in one kind of operating system (such as Windows 2000) to communicate with a program in the same or another kind of an operating system (such as Linux) by using the World Wide Web's Hypertext Transfer Protocol (HTTP)and its Extensible Markup Language (XML) as the mechanisms for information exchange. Since Web protocols are installed and available for use by all major operating system platforms, HTTP and XML provide an already at-hand solution to the problem of how programs running under different operating systems in a network can communicate with each other. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and pass it information. It also specifies how the called program can return a response[2]

Although our previous definition of Web Services mentions that Web Services can use HTTP and XML to access objects, it doesn't specify how to use them. SOAP was developed and approved by a group of vendors in order to specify a simple protocol used for communication (e.g. by other systems, clients, other objects etc.) with objects using HTTP and XML. SOAP is probably the most common method used today for Web Services communication. This is due to its ease of use and acceptance by a large number of vendors. SOAP has been submitted to the W3C as a proposed standard and further development (http://www.w3.org/TR/SOAP/).[3]

Why all the talk about security?

Web Services security discussions are all over the media these days, why is that? To security-minded people like us, the answer is probably obvious. After all, we just explained that SOAP is a standard to *allow others (other users, systems, servers, objects etc.) to execute code on a system.* What we've just described is in fact every hacker's dream. This is a story that seems to keep repeating itself in the technology

field. In application developers' headlong rush to implement new technology, functionality, and openness guess what they forgot to think about? That's right- security! As often happens (developers are often focused on functionality, proper operation, interoperability, and deadlines) security has become an afterthought, an add-on to Web Services, rather than being built in from the ground up.

See the below quote written by a developer on MSDN, it's very telling about the difference in perspectives between information security professionals and application development professionals:

> Firewall Woes
>
> Currently, developers struggle to make their distributed applications work across the Internet when firewalls get in the way. Since most firewalls block all but a few ports, such as the standard HTTP port 80, all of today's distributed object protocols like DCOM suffer because they rely on dynamically assigned ports for remote method invocations. If you can persuade your system administrator to open a range of ports through the firewall, you may be able to get around this problem as long as the ports used by the distributed object protocol are included.
>
> To make matters worse, clients of your distributed application that lie behind another corporate firewall suffer the same problems. If they don't configure their firewall to open the same port, they won't be able to use your application. Making clients reconfigure their firewalls to accommodate your application is just not practical.
>
> Since SOAP relies on HTTP as the transport mechanism, and most firewalls allow HTTP to pass through, you'll have no problem invoking SOAP endpoints from either side of a firewall. Don't forget that SOAP makes it possible for system administrators to configure firewalls to selectively block out SOAP requests using SOAP-specific HTTP headers.[4]

Did the hairs on the back of your neck stand up while you read the above? As a security-minded IT professional, they should have. The author is saying that an inherent reason that SOAP was created is to make it easy to bypass security controls that are currently in place! This developer views firewalls as an impediment. In his view, SOAP over HTTP was developed so that he doesn't have to deal with the current procedures for allowing traffic in and out of a corporate network. However, the final line of the quote illustrates the fact that to implement secure web services security professionals must implement additional controls than are currently in place. Now that everyone has woken up to the potential threats that Web Services can introduce, several efforts are under way to develop security standards.

As an aside, Skonnard says that most firewalls can selectively block out SOAP requests using SOAP headers. Although today's firewalls do have some SOAP-checking functionality, unless there are policies and procedures set and the firewall administrator properly configures the firewall (and is made aware of XML-SOAP applications), the

firewall will not be of much help regarding Web Services security, even at the perimeter of an organization. As we will see, much more than perimeter security is required for a secure Web Services implementation.

Why not use standard Web security?

The World Wide Web has been around for a while now. There are an incredible number applications that use the de facto standard SSL for encryption of data in transit, signing of data in transit, and site verification (i.e. X.509 server-side digital certificates). The vast majority of sites today use username and password authentication over SSL to verify identity- a decent number of sites go even further and use client-side certificates, tokens, and other better authentication methods. Why not just use these exact same methods and practices to secure Web Services? There is an article on this topic by Sarah Evans and Olwyn Dowling at Webservices.org (http://www.webservices.org/index.php/article/articleview/529/1/24/).[5]

We will follow Evans and Dowling's discussion closely in this section. First, let us look at the differences between common Web sites and Web Services implementations. Web sites today are typically oriented toward individual users. If it is a retail site it can usually described as B2C. If it is a company portal for clients or employees it is there to provide information or self-service administrative functions. In general each transaction is of relatively low risk: credit card liability is low for the consumer, retailers always must afford some fraudulent transactions, and a company's administrative changes can usually be rolled back if an error or fraud has been found. The point is that although the threat of a few malicious transactions may be high, the overall risk of damage is fairly low (e.g. some books are fraudulently obtained from xyzpublishing.com , a single employee's records are maliciously changed etc.) Current web sites are not as concerned about client authentication due to the lower risk on individual transactions and low transaction volume per user. Most sites use usernames and passwords and make sure that transmission of them are performed using SSL-encrypted communications. This makes it hard for eavesdroppers to steal authentication information and forces them to resort to other more detectable methods such as brute force and dictionary attacks to try to obtain this information. One reason that client-side SSL certificates have not significantly penetrated the market is that they require that certificates be distributed and properly configured at each client host. In most cases the effort required to use client side certificates for authentication outweighs potential benefits when taking the risk level into account.

However, with Web Services, the goal is usually to open a company's internal systems, processes, and data for interoperability with other systems. Often the goal is to open Web Services beyond the enterprise over the Internet. Generally Web Services are seen as way to allow quick, loosely coupled integration between entities and applications without a large effort. This is a very different model for use and the risks can be much higher. For example, a Web Service could expose a company's customer financial information (revenue, debt, etc.) for legitimate use, but if there was a breach of

security a malicious individual could quickly access all of the customers' information and blackmail the company by threatening to expose that information. This situation could potentially put a company out of business. This risk level requires that better security measures be put in place for Web Services. As most Web Services are implemented, it is very important that strong authentication of the requester of the service is implemented to prevent malicious and damaging activity. Client-side SSL certificates could provide this stronger authentication to Web Services. The problem is still the onerous task of installing, configuring, and maintaining these certificates across each potential requester of Web Services. With Web Services' goal of permitting integration with a minimum of set up, configuration, and indeed even knowledge about the workings of the specific Web Service client-side SSL certificates don't seem practical in most cases.

Another point to consider is Web Services and SOAP relative immaturity as technologies. In general when technologies are first adopted they have not had the rigorous testing and trials required to make them secure. If you survey incidents that occur today, many occur because of product vulnerabilities such as buffer overflows, SQL injection, malformed URLs and other unintentional problems with code or applications rather than ill-designed security schema. While many of these vulnerabilities have occurred in widely implemented, 'mature' technologies (i.e. IIS, OpenSSL, OpenSSH, Microsoft SQL server), the risk of these type of vulnerabilities is even greater in newer, sparsely implemented technologies such as Web Services and SOAP. Evans and Dowling give a good example of a vulnerability in SOAPLite, an implementation of SOAP in Perl, which allows a requester (through SOAPLite) to simply execute any Perl function on the target system even though it is not meant to be offered as a web service. The examples Evans and Dowling cite[6] (derived from a vulnerability posting at phrack http://www.phrack.com/show.php?p=58&a=9)[7] can use the common Perl function **sendfile** to download any file from the target simply by changing the SOAP URL syntax from the allowed function (name and namespace) to an assumed function that you want to try and execute. In the example, the request is easily made to download a UNIX password file or a Microsoft SQL master database file, even though the capability to use the **sendfile** command was never enabled or intended.

Unfortunately, even strong authentication and SSL encryption does nothing to protect against this type of attack as well as the common buffer overflow and malformed URL attacks. Ultimately, message-level monitoring and validation mechanisms for security will be required to offer a robust, preventative, defense in depth approach to Web Services. Additional SOAP message-level logging functions should be implemented in order to provide detective and corrective capabilities when an attack does occur. Taking all of the above information into account, it becomes clear that current web security methods and practices are inadequate to ensure Web Services security.

At this point I'd like to address a misconception that I've seen written about time and again regarding Web Services security. Often, articles are published and statements are made lamenting the shortcomings of Web Services security. Usually in the same statement or breath it is mentioned that this is holding back Web Services deployment beyond most enterprises' perimeter firewalls. This implies that it is safe and prudent to

deploy Web Services within your security perimeter. As anyone knowledgeable on the topic will tell you, the majority (most estimates put it at 70-80%) of all security breaches occur within an organization's perimeter rather than from the outside. Considering that internal systems are often less secured (i.e. many enterprises only use firewalls, IDS, and encryption on outward-facing systems), deploying Web Services exclusively for internal use can still comprise a huge threat to security. In any case a risk assessment must be done even before deploying Web Services on an internal network.

Standards – how many do we need?

As it became obvious that additional security functionality would be required to securely implement Web Services, standards were developed. As is so often the story with 'standards', there are several parallel (and sometimes competing) efforts toward the same goal. In the case of Web Services, XML, and SOAP there are three major standards bodies currently working on standardization: the W3C, the IETF, and OASIS. These bodies sometimes work to develop complementary standards but often work independently of each other and develop overlapping or competing standards as well. Currently there are no less than 13 Web Services related standards or draft standards in various stages of development and ratification by these bodies.[8] Making sense of them is a daunting task. Let's take a look at some of them and assess their complementarities or overlap.

IETF

The first standard we'll look at is SASL or Simple Authentication and Security Layer. SASL is an IETF standard and its description is available in RFC2222 (http://www.ietf.org/rfc/rfc2222.txt?number=2222)[9]. SASL is a standard which allows the insertion, i.e. a 'shim', of an additional security layer to an existing, connection-based protocol. If SASL is used, it allows a server to authenticate a user as well as to protect subsequent communications between the client and server (i.e. for privacy and integrity of transmissions). The higher level protocol must implement a command to authenticate a user to a server and can include a command for the negotiation of protection of protocol transmissions.

SASL is designed in a non-specific and extensible manner so that it can be used with virtually any authentication and protection methods. It provides a simple way to allow connection-based protocols to incorporate security protocols with little or no changes to the higher level protocols or the connection. The example used in the RFC shows how SASL can be used to insert a security layer between IMAP4 and TCP to enable the IMAP server to authenticate a user and to optionally encrypt and sign further IMAP client/server communications if negotiated. The SASL layer performs the security functions like authentication, encryption, and decryption without the higher-level protocol's knowledge or understanding.

SASL achieves this utility by defining 'mechanisms' which are use when invoking a SASL command. The mechanism name is a string which signifies the methods used when SASL performs an authentication or other security function. Mechanisms must be

registered with the IANA. The list of registered SASL mechanisms can be found in the "Assigned Numbers" RFC issued by the IETF.

In RFC2222, several mechanisms are defined including: Kerberos, S/KEY, GSSAPI, and EXTERNAL. The EXTERNAL mechanism allows the server to use a mechanism external to SASL (such as IPsec) to provide authentication and other security functions.

Thus it is easy to see how Web services can use SASL to implement any of a number of authentication, encryption, and signing mechanisms to provide these security services. SASL provides many alternatives and in doing so allows Web Services access to effective user authentication techniques as opposed to simply using SSL. For example, currently defined mechanisms include NTLM and Securid authentication, both of which have made significant inroads to network infrastructures when compared with SSL client-side certificates. To get an idea of the true breadth of SASL mechanisms, look at the list at http://www.iana.org/assignments/sasl-mechanisms[10].

An important note on SASL is that it can provide user authentication and encryption/signing of transmissions between the client and host only for the duration a session; it does not differentiate regarding content or access control. Once the transmission of the information is complete it is no longer controlled or encrypted. In this sense SASL is similar to SSL.

In a complementary fashion, the W3C has been working on two standards that also specify mechanisms for encryption and digital signature, XML Encryption and XML Digital Signature. Let's take a look at these two initiatives and how they fit in.

W3C

XML Encryption is a recommendation of the W3C as of December 10, 2002 (see http://www.w3.org/TR/xmlenc-core/)[11]. The XML Encryption recommendation provides a process which allows the encryption (and decryption) of arbitrary data within an XML document. The encrypted data can be the entire document, an XML Element, or an XML Element Content, or other arbitrary data within the document. XML Encryption also provides for listing the methods of encryption and data hashing as well as for the encryption of encryption keys. XML Encryption gives developers a high degree of granularity when encrypting content- they can choose to encrypt an entire document or simply the content of on element in a document. It also provides for the encryption of different parts of a document with different keys so that only authorized viewers can access certain parts of the document.

In the W3C recommendation they give an excellent example of encryption options when using a credit card for an online payment (see section 2 of the recommendation)[12]. The recommendation illustrates how a developer can choose between the following scenarios:

1. Encrypting the entire credit card element. By doing so all information is encrypted including the fact that a credit card was used at all (see section 2.1.1 of the above)[13]. All references to credit cards are encrypted and contained in cipher data content. An eavesdropper or a document reader without the proper decryption keys will only know that there is some encrypted data but have no clue as to what the data is, the element is simply labeled as <Encrypted Data>.

2. Encrypting the content for the Credit Card element. In this scenario the element name (i.e. <Credit Card>) is left in plaintext (see 2.1.2 of the above)[14] but the number, issuer, and expiration date content and their respective element labels are encrypted. An eavesdropper or document reader without decryption keys can see that a credit card was used along with limit and currency information, but cannot read any other element content within the Credit Card element.

3. Encrypting the specific Credit Card element content for the Number element. Using this method only the actual credit card number (i.e. the Number element content) is encrypted (see section 2.1.3 of the above)[15]. The element name, issuer, expiration date, transaction amount, and card limit are all left in plaintext with only the actual credit card number being encrypted. An eavesdropper or document reader without decryption keys can tell that a credit card was used, and know all of the credit card information except the actual card number.

4. Encrypting the entire document. Using this method the entire XML document is encrypted (see section 2.1.4)[16]. Anyone without the decryption keys can only see that there is an XML document with encrypted data. All other information such as element names and element data content are encrypted in cipher data.

XML Encryption also allows for 'Super-Encryption" (see 2.1.5)[17]. Super-Encryption is an encrypted field in which the data is actually the encryption of an <Encrypted Data> or <Encrypted Key> element. When using super-encryption, the entire original <Encrypted Data> or <Encrypted Key> element must be used in the new <Encrypted Data> field- it is invalid to encrypt only content or child elements of these element types.

XML Encryption also allows the developer to include encryption method and key information as elements of the <Encrypted Data> element. It also makes use of the XML Signature (XML-DSIG) schema, especially with regard to encryption keys.

The XML Signature (XML-DSIG) specification is a W3C Recommendation and can be found at http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ [18]. It was produced by a combined IETF/W3C working group in order specify syntax and processing rules for the use of digital signatures with XML. The XML Signature recommendation describes a standard method of digitally signing arbitrary data for use with XML data. XML Signature can be used to sign either an entire XML document or elements within an XML document. The signature is related to a specific data object which either can be part of the same XML document or referenced via a URI. The signature can either be enveloped or enveloping. An enveloping signature means that the signed data is contained within an element of the actual signature object. An enveloped signature

means that the signature is contained as an element of the XML content. In the enveloped case it is important that the signature does not take itself into account while signing operations are performed.

XML Signature provides a method for two standard functions: the document reader can verify that the content was signed by a particular authority, the document reader can verify that the content is the same as when the data signed.

One feature of XML Signature worth noting is that it provides message authentication, meaning that it detects attacks (by using checksums) where both the content and signature have been changed in order to fraudulently change data and make the receiver believe that it has been signed. This provides verification of the source of the data and the data integrity of the content.

You should note that XML Signature provides canonicalization for data to be signed. This concept is not always present in discussions of encryption, signature, and message digests. Canonicalization is a process where an XML document is converted into a physical representation of the document. Several slight variations of a particular document may all correspond to the same canonical representation. This is necessary because although two XML documents may be logically the same, i.e. equivalent, they may be slightly different due to differences in line feed characters, empty tags, hex values substituted for names within certain elements or attributes etc. These slight variances in equivalent documents can occur because of operating system variations on different systems, format translation, data transfer methods or other operations that are not intended to alter data within a document. If a document was directly signed in this manner two logically equivalent documents would fail signature tests in certain cases, causing the document reader to think that the document was invalid. In order to avoid this situation, documents or data objects are put into a canonical form which is a physical representation which takes these slight variations into account so that two logically equivalent documents are represented in an identical manner. There is recommendation on Canonical XML available at http://www.w3.org/TR/xml-c14n .[19]

XML Signature remains agnostic with regard to what algorithms can be used for message digest, signing, and canonicalization. For example you can use commonly defined algorithms such as SHA-1 for message digests, RSA for signing, and XML-C14N for cannonicalization. XML Signature does however have several required algorithms that all implementations must support and several recommended algorithms which most implementations will support.

It is easy to see that the use of XML Encryption and XML Signature can allow XML/SOAP applications additional security functionality required in a Web Services implementation. When used properly, these recommendations (along with appropriate supporting elements) can provide the following security functions:

Confidentiality – Only authorized viewers of a document or document element can decrypt information. If a document viewer does not have access to the necessary keys, they can't view any encrypted data.

Integrity – Digital signatures ensure that the signed document or document elements are preserved in an equivalent state to when they were sent. Signatures also allow the reader to identify who signed the document and therefore who approved its content.

XML Encryption and XML Signature provide the syntax necessary to implement most of the common functions for encryption and digital signature within an XML framework. XML Encryption and XML Signature provide some of the necessary functionality to implement application-level security functions and defense-in-depth as far as Web Services are concerned. However, due to the recommendations' open natures they still rely on common services for key and certificate management (e.g. a PKI) which are vital to proper encryption and digital signature functionality. Unfortunately, key management systems are not commonly implemented within organizations today and are rarely implemented between organizations. These are precisely the situations (i.e. B2B) where Web Services are expected to be most valuable. So similar impediments exist when implementing XML Encryption/Signature as exist for other PKI infrastructure systems like X.509 or PGP.

In an effort to provide specifications to address key management for XML Encryption and XML Signature, the W3C has introduced the XML Key Management Specification (XKMS). The specification can be found as a note at http://www.w3.org/TR/xkms/.[20] The working draft of XKMS (XKMS 2.0) can be found at http://www.w3.org/TR/xkms2/.[21] The initial note is listed for discussion by the W3C and has no official endorsed status. The XKMS 2.0 specification is still under discussion and is by no means complete. We'll look at the XKMS 2.0 specification since this will most likely evolve into a standard in the future.

XKMS 2.0 defines protocols for the distributing and registering keys to be used in conjunction with XML Encryption and XML Signature standards. It is comprised of both the XML Key Information Service Specification (X-KISS)[22] and the XML Key Registration Service Specification (X-KRSS)[23]. X-KISS is a protocol to allow an application to delegate the processing of key information to a service. It is used to retrieve public keys as well as identification information that is bound to those keys. X-KRSS is a protocol that allows a key-pair owner to register that key-pair for use with the X-KISS or other trust assertion protocols.

X-KISS allows an application (an X-KISS client) to delegate public key processing functions to an external Trust service. In this way, applications can be built without the development of complex and intricate PKI functionality, the application can access an external PKI infrastructure with simple XML syntax. Additionally, X-KISS does not require the use of a particular PKI specification. An application can be built in an open manner and can take advantage of several types of PKI's such as X.509, PGP, etc.

X-KRSS allows an application (an X-KRSS client) to register public key information and optionally bind certain information to that public key. For example, a client can request that a public key be bound to identification information such as a name so that the public key information can be used to verify the identity of the source of signed data elements. The key-pair can be generated by the client in advance of registration or can

be generated by the service when requested by the client. In the latter case the protocol also provides the manner in which the service can communicate the private key to the client. Additionally, the protocol can be used for private key recovery at a later time.

XKMS 2.0 defines a three tier implementation model that allows applications to decide what level of services it wishes to take advantage of. In the Tier 0 Service Model the application performs key processing functions without delegating any functions to a trust service. In Tier 1, the application retrieves the key from the trust service but validates the public key itself rather than delegating validation to a trust service. In Tier 2, the application behaves as in Tier 1 but also gets validation from the trust service about what data the key is bound to, for example a name or identity could be bound to the key.

Regardless of which Tier of XKMS service is used, XKMS provides a simple, common XML syntax for the retrieval of keys from directories, checking key revocations status , and operations required for trust chains.

OASIS

The other major organization working on standards within for XML is OASIS. OASIS has several specifications in various status of development/ratification that are specifically targeted at securing web services. Many of these specifications either build on or are complementary to the specifications created by other bodies such as the W3C and IETF.

The first OASIS specification we will discuss is Security Assertion Markup Language (SAML). We discuss the SAML Specification 01 of May 31, 2002. The core specification can be found at http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf [24]

SAML is a framework for exchanging security information via XML. SAML defines **subjects**, **assertions, and authorities.** Subjects are defined as an entity to which an identity is assigned for security purposes. Often this subject is a specific person or machine that is identified within a system or application. SAML conveys security information via assertions.

Assertions are security statements made about the subject and conveyed from a SAML Authority. Assertions can contain information about security authentications performed by subjects, security attributes of subjects, and authorization decisions about subjects authority to access resources. SAML Assertions are in XML format and can be nested, so a particular assertion can contain any and/or all of the above types of the above security information in it.

SAML clients can request an assertion from an authority and receive a response from the authority. Only authorities can create assertions, so clients request and receive assertions but can't create them. Authorities can use a variety of sources to create responses. These sources can be other assertions as well as external policy stores, so a SAML Authority can request and create assertions. SAML Authorities are categorized

as authentication authorities, attribute authorities, and policy decision points. Once a client receives an assertion from an authority the client can present the assertion to request access to a system or data or verify his identity.

SAML defines the necessary XML protocol to request assertions and send responses. This protocol can be bound to many underlying communications and transport protocols. However, the specification only defines and sanctions a single binding to SOAP over HTTP (see the [SAMLBind][25] specification for details).

SAML was designed to enable single sign-on so that a user could only authenticate once even if they were using resources in different security domains. This greatly simplifies security from the end-user standpoint and makes cross-domain operations and management easier from the systems administration standpoint. However, care must be used to ensure that single sign-on is implemented properly and does not introduce untrusted subjects into a domain.

OASIS has a document that discusses security and privacy issues specifically for the SAML specification. It is titled, "Security Privacy Considerations for Oasis Security Assertion Markup Language" (http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-01.pdf).[26]

Several key issues are discussed in this document. Most important is the fact that SAML rides atop several other key components. SAML does not define how authentication, attributes, or policy are implemented or managed- SAML is dependent on underlying services such as a PKI, encryption methods and ciphers, and signature algorithms. So the SAML assertions are only as reliable as the underlying methods used to manage authentication, encryption, signature, and policy enforcement.

SAML itself is especially susceptible to several well-known methods of attack and measures must be taken to minimize the risk of these attacks. SAML assertions are returned to a client and are then out of the control of the issuing authority. You must be careful about what information is contained in an assertion as it may be stored persistently at a remote system and used for malicious purposes. Once the assertion is returned to the client he may share (knowingly or inadvertently) these assertions with any number of other unknown subjects or systems even if the issuer takes care to properly sign and encrypt this information while in transit. Additionally, SAML by its nature exposes behavioral information by requiring the client to request assertions from an authority. A client can be identified as part of a select security domain by the fact that it regularly requests assertions from a specific authority. For example, if a specific machine often makes requests from the XYZ Company Authentication Authority, someone can observe this and surmise that that client has a relationship with XYZ Company.

Risks arise in SAML from the request-response protocol inherent in the specification. First, a denial-of-service attack is possible because the request and response for assertions can be computationally expensive. Another risk is that of replay at the SOAP

level for the purpose of DOS. The attacker would not need to understand the content of the replayed information so even XML Encryption does not prevent this attack.

Thus SAML can allow a security domain to trust another domain's assertion. This allows each security domain to maintain their own authorities and underlying systems (e.g. PKI, authentication mechanisms, policy enforcement, etc.) and through SAML assertions pass information between security domains for the purposes of authentication, attributes, and policy decisions. If the receiving security domain trusts the source of the assertion, it can then act on requests made by the client without requesting further (and possibly duplicate) information from the client. This can be used to introduce single sign-on, access control and many other functions in web services. SAML addresses the need where inter-domain functions are necessary for authentication, key exchange/verification, and especially in the case of B2B Web Services which are the focus of many if not most Web Services today.

Another important specification is the WS-Security specification which is currently an interim draft specification published by OASIS (available at http://www.oasis-open.org/committees/wss/documents/WSS-Core-08-1212-merged.pdf).[27] WS-Security is an extension of SOAP protocols to provide message integrity and single message authentication. It provides a facility for associating security tokens with messages. WS-Security builds on the XML Encryption and XML Signature definitions so that a token or several tokens can be sent with an XML message in order to prove a message source is as claimed and/or that the message has not been altered. These tokens are deliberately defined in a generic manner so that they can support many technologies. A security token is defined as a collection of one or more claims. Some examples of a what a security token could be are: a Username, a certified X.509 key, or a Kerberos ticket.

WS-Security defines a message header block which can be included in XML messages in order to convey security information. WS-Security defines how this information (i.e. a token) can be used to implement message privacy, authentication of the author/sender, and message integrity.

The WS-Security specification also defines a method to encode binary security tokens, a framework for XML-based tokens, and how to include opaque encryption keys in a message.

WS-Security is designed to be open and supports multiple security models, multiple security token formats, multiple signature formats, multiple encryption formats, and multiple trust domains. It supports but is not limited to the use of PKI (X.509), Kerberos, and SSL technologies. An implementation can use the three mechanisms, namely: token sending, message integrity, and message confidentiality together or selectively on a given message. It is important to note that WS-Security provides message-level security and not just transport level security.

Conclusions

Although Web Services initially lacked the necessary tools to implement secure applications, significant progress is now being made. The three major standards bodies, W3C, OASIS, and the IETF have been working together and in mostly complementary fashion to propose new specifications that will ultimately become standards in order to give Web Services developers the tools that they need to build and implement secure services. The standards bodies have taken a practical approach by developing drafts and specifications that are comprised of open, flexible, and extensible recommendations which in turn support current technologies that are widely deployed. In this way rather than forcing Web Services developers to reinvent the wheel they allow them to use parts of the security infrastructure that are currently in place.

As the current draft standards become ratified and more widely implemented, Web Services developers will be able to use standards such as SASL, XML Encryption, XML Signature, XML Key Management, SAML, and WS-Security along with currently implemented technologies such as SSL, PKI, Kerberos, firewalls, and content filters to implement defense in depth at the application, system, and network levels. Functionality including encryption,digital signature, single sign-on across security domains, and key management are now available to Web Services developers- at least in the draft forms of standards. Hopefully all of these specifications and proposed standards will find quick acceptance so that Web Services applications can be deployed securely while living up to their potential in the near future.

References

1. Anonymous. URL: http://www.perfectxml.com/WebSvc1.asp (21 Jan 2003)
2. Anonymous. URL: http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci214295,00.html (21 Jan 2003)
3. Box, Ehnebuske, Kakivaya et al. “Simple Object Access Protocol (SOAP) 1.1”. W3C Note. May 8, 2000. URL: http://www.w3.org/TR/SOAP/ (21 Jan 2003)
4. Aaron Skonnard. “SOAP: The Simple Object Access Protocol”. Microsoft Internet Developer. January 2000. URL: http://www.microsoft.com/mind/0100/soap/soap.asp (21 Jan 2003)
5. Evans, Sara and Dowling, Olwyn. “Is SSL enough security for first-generation Web services?”. July 18 2002. URL: http://www.webservices.org/index.php/article/articleview/529/1/24/ (21 Jan 2003)
6. ibid
7. Anonymous author. “RPC without borders”. Phrack 58 download. December 28, 2001. URL: http://www.phrack.com/show.php?p=58&a=9 (21 Jan 2003)
8. Fontana, John. “Securing Web Services”. Network World. September 23, 2002. URL: http://www.nwfusion.com/buzz/2002/websec.html (21 Jan 2003)
9. Meyers, J. “Simple Authentication and Security Layer (SASL)”. RFC2222. October 1997. URL : http://www.ietf.org/rfc/rfc2222.txt?number=2222 (21 Jan 2003)
10. Anonymous. “SIMPLE AUTHENTICATION AND SECURITY LAYER (SASL) MECHANISMS”. August 17, 2001 URL: http://www.iana.org/assignments/sasl-mechanisms (21 Jan 2003)
11. Imamura, Dillaway, and Simon. “XML Encryption Syntax and Processing”. W3C Recommendation. 10 December 2002. URL: http://www.w3.org/TR/xmlenc-core/ (21 Jan 2003)
12. ibid, Section 2.1
13. ibid, Section 2.1.1
14. ibid, Section 2.1.2
15. ibid, Section 2.1.3
16. ibid, Section 2.1.4
17. ibid, Section 2.1.5
18. Bartel, Boyer, Fox, LaMacchia, Simon. “XML-Signature Syntax and Processing”. W3C Recommendation. February 12 2002. URL:

http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/ (21 Jan 2003)

19. Boyer, John “Canonical XML Version 1.0”. W3C Recommendation. March 15, 2001. URL: http://www.w3.org/TR/xml-c14n (21 Jan 2003)
20. Ford, Hallam-Baker, Fox, Dillaway, Brian LaMacchia, Epstein, Lapp. “XML Key Management Specification (XKMS)”. W3C Note. March 30 2001. URL: http://www.w3.org/TR/xkms/ (21 Jan 2003)
21. Hallam-Baker, Phillip editor. “XML Key Management Specification (XKMS 2.0)”. W3C Working Draft. March 18 , 2002. URL: http://www.w3.org/TR/xkms2/ (21 Jan 2003)
22. ibid
23. ibid
24. Hallam-Baker, Phillip and Maler, Eve. “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)”. Committee Specification 1. May 31,2002. URL:

http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf (21 Jan 2003)

25. Mishra, Prateek editor. “Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML)”. cs-sstc-bindings-00. April 19, 2002.

URL: http://www.oasis-open.org/committees/security/docs/cs-sstc-bindings-00.pdf (21 Jan 2003)

26. Moses, Mishra, Hodges, et al. “Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML)”. Committee Specification 1. May 31, 2002. URL:

http://www.oasis-open.org/committees/security/docs/cs-sstc-sec-consider-01.pdf (21 Jan 2003)

27. Hallam-Baker, Kaler, Monzillo, Nadalin et al. “Web Services Security Core Specification”. Working Draft 08. December 12, 2002. URL:

http://www.oasis-open.org/committees/wss/documents/WSS-Core-08-1212-merged.pdf (21 Jan 2003)

Other References for Web Services security study

Anonymous. “Web Services Zone”. IBM .URL: http://www-106.ibm.com/developerworks/webservices/ (21 Jan 2003)

Anonymous. “xwss.org – XML Web Services Security Forum”. URL: http://www.xwss.org/index.jsp (21 Jan 2003)

Anonymous. “Web Services Security: RSA Security Whitepaper”. URL: http://www.rsasecurity.com/products/cleartrust/whitepapers/WSS_WP_0802.pdf (21 Jan 2003)

Karpinski, Richard. “Reactivity Ships ‘Firewall’ for Web Services”. Internet Week. July 22, 2002. URL: http://www.internetweek.com/webDev/INW20020722S0010 (21 Jan 2003)

Doyle, Peter. “Security and Web Services”. SC Online Magazine. October 2002. URL: http://www.scmagazine.com/scmagazine/sc-online/2002/article/46/article.html (21 Jan 2003)

Fontanna, John. “Top Web Services Worry: Security”. Network World. January 21, 2002. URL: http://www.nwfusion.com/news/2002/0121webservices.html (21 Jan 2003)

Last Updated: July 21st, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS